Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Manuel Meyer

Inertiale Navigation mittels Deep-Learning

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Manuel Meyer

Inertiale Navigation mittels Deep-Learning

Masterthesis eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 27. November 2019

Manuel Meyer

Thema der Arbeit

Inertiale Navigation mittels Deep-Learning

Stichworte

inertiale Navigation, INS, Trägheitsnavigation, künstliche Intelligenz, Machine Learning, rekurrente Netze, Deep Learning, LSTM, Faltungsnetzwerke, multidimensionale Zeitreihen

Kurzzusammenfassung

Inertialsensoren spielen in der inertialen Navigation eine zentrale Rolle. Die Anforderungen die gängige inertiale Navigationssysteme mit sich bringen, können oftmals nicht alleine durch die Verwendung eines einzelnen Systems gestemmt werden, weshalb heutzutage vielfach auf integrierte Navigationssysteme gesetzt wird. Diese bringen aber einen hohen Implementierungs-, Wartungs- und Kostenaufwand mit sich. Allerdings hat die Entwicklung der letzten Jahre dazu geführt, dass sich kostengünstige Inertialsensoren in zahlreichen Bereichen und Anwendungen etabliert haben. Die kostengünstigen MEMS-Sensoren haben jedoch den großen Nachteil, dass sie mit der Zeit einen zunehmenden Schleppfehler verursachen, weshalb sie sich nur mit einem entsprechenden Referenzsystem (z. B. GNSS) zur Positionsbestimmung eignen. Ein Ansatz, mit dem es ermöglicht wird, auch den kostengünstigen Inertialsensoren wie sie mittlerweile in jedem Smartphone zu finden sind, eine Positionbestimmung zu erlauben, wäre daher wünschenswert. Gegenstand dieser Arbeit ist aus diesem Grund die Vorstellung eines neuartigen Lösungsansatzes einer Positionsbestimmung mittels Low-cost-Inertialsensoren auf Basis von tiefen neuronalen Netzen. Der Ansatz soll dabei umgebungsunabhängig und ohne ein entsprechendes Referenzsystem die Anforderungen der Zuverlässigkeit, Verfügbarkeit und Genauigkeit erfüllen können, sodass eine inertiale Navigation auch in abgeschatteten Bereichen wie der Indoor-Navigation garantiert werden kann.

Manuel Meyer

Title of the paper

Inertial Navigation Using Deep-Learning

Keywords

Inertial navigation, ins, artificial intelligence, machine learning, recurrent neural networks, deep learning, lstm, convolutional neural networks, multidimensional time series

Abstract

Inertial sensors play a major role in inertial navigation. The requirements of current inertial navigation systems often can not be solved by the use of a single system, which is the reason

why *integrated navigation systems* are often used nowadays. But these systems involve high implementation, maintenance and expenditures. However, the development in recent years has led to the establishment of *low-cost* inertial sensors in various fields and applications. The low-cost MEMS sensors, however, have the big disadvantage that they cause unavoidable errors to the entire system over time, so that they are only suitable with a corresponding reference system (e.g. GNSS) for position determination. An approach that makes it possible to allow position determination even with low-cost inertial sensors as they are now to be found in any smartphone, would therefore be desirable. For this reason, the subject of this thesis is the presentation of a novel solution of position determination by means of low-cost inertial sensors based on deep neural networks. With this approach it should be possible to meet the requirements of reliability, availability and accuracy regardless of the environment and without a corresponding reference system, so that inertial navigation can also be guaranteed in shadowed areas such as indoor navigation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Exponetial Linear Unit	Als **ELU** wird die Aktivierungsfunktion eines Neurons bezeichnet, die für positive Werte linear verläuft, für negative – exponentiell
Fingerprint	Fingerprinting beschreibt eine Methode, in der Signale (WLAN, Magnetometer etc.) an einem bestimmten Ort zu einer bestimmten Zeit aufgezeichnet werden. Dabei wird die Empfangsstärke der eingehenden Broadcastsignale (RSSI) als ein charakteristischer Fingerabdruck am Ort der Messung erstellt und in einer Tabelle abgespeichert
Hyperparameter	„Ein Hyperparameter ist ein Parameter des Lernalgorithmus (nicht des Modells)“ ([Géron (2017)]:28). Als Synonyme kann sowohl **Metaparameter**, **freie Parameter** als auch **tuning Parameter** verwendet werden
Inertiales Navigationssystem	Bei einem inertialen Navigationssystem (engl. Inertial Navigation System (INS)) oder auch Trägheitsnavigationssystem handelt es sich um ein 3-D-Messsystem mit einer inertialen Messeinheit (engl. Inertial Measurement Unit, IMU), diese besteht aus mehreren Beschleunigungs- und Drehratensensoren. Es bestimmt fortwährend die Position, Geschwindigkeit und Orientierung im Raum mittels Integration der erfassten Beschleunigung und Winkelgeschwindigkeit über die Zeit. Dieses Verfahren nennt man auch Koppelnavigation („Dead Reckoning“)

Metaparameter	Als Synonyme kann sowohl **Modell Hyperparameter**, **freie Parameter** als auch **tuning Parameter** verwendet werden. „Ein Hyperparameter ist ein Parameter des Lernalgorithmus (nicht des Modells)“ ([Géron (2017)]:28)
Multilayer Perzeptron	Als **MLP** wird eine Klasse von künstlichen neuronalen Feedforward-Netzwerken bezeichnet, die aus mehreren verdeckten Schichten bestehen
Polarkoordinaten	Beschreibt ein zweidimensionales Koordinatensystem oder auch Polarkoordinatensystem, in dem jeder Punkt in einer Ebene durch den Abstand von einem vorgegebenen festen Punkt und den Winkel zu einer festen Richtung festgelegt wird. Das Verhältnis zwischen zwei Punkten dabei durch Winkel und Abstände beschrieben und berechnet
Rectified Linear Unit	Als **ReLU** wird die Aktivierungsfunktion eines Neurons bezeichnet, die für positive Werte linear verläuft, für negative Werte 0 erzeugt
Trainingsparameter	Die für das Training zuständigen Parameter eines DL-Modell. Ein paar müssen vorher definiert werden und bleiben konstant, andere wiederum werden während des Trainings angepasst
Trainierbarer Parameter	Eine Modell interne Konfigurations-Variable, sie wird im Laufe des Trainings gelernt bzw. angepasst (Gewichte und Biase der Neuronen)

1. Einleitung

Maschinen lernen denken: Der Einsatz von Robotern, Sensortechnik, Big Data und künstlicher Intelligenz spielt eine entscheidende Rolle in der fortschreitenden Entwicklung von Wirtschaft und Gesellschaft. Maschinen werden smarter als zuvor und sorgen für einen grundlegenden Strukturwandel – denn diese technischen Systeme sind lernfähig und zunehmend in der Lage, ihr bereits erlerntes Wissen auf neue Situationen zu übertragen und ihr Verhalten intelligent anzupassen. Sie können lernen neue Prozesse zu verstehen und zu planen, Prognosen treffen und sogar mit Menschen und anderen Maschinen interagieren. Vor allem der Bereich des Machine Learnings (ML) konnte in den letzten Jahren enorme wissenschaftliche Fortschritte nachweisen. Der vielversprechendste Ansatz hierbei ist das Deep-Learning (DL) mit tiefen neuronalen Netzen. Durch ihre komplexe Architektur ist es ihnen möglich eine Hierarchie von Merkmalen aus Daten zu erlernen und eine Musterkennung oder Objektklassifizierung durchzuführen. Aus multidimensionalen Zeitreihendaten können sowohl temporale Abhängigkeiten als auch ein mögliches dynamisches Verhalten abgeleitet werden. Die Grundlage solcher DL-Modelle bilden die mehrschichtigen Multilayer Perceptrons (MLPs) oder auch Deep Feedforward Networks (DFF). Weitere derzeit verbreitete Varianten des MLP sind neben den Convolutional Neural Networks (CNN) (dt. „Faltungsnetze“) auch die Recurrent Neural Networks (RNN). Die Entwicklung der künstlichen Intelligenz der letzten Jahre hat dazu geführt, dass sie sich bereits in zahlreichen Gebieten und Anwendungsfeldern etablieren konnten.

Im *Predictive Maintenance* Bereich können bereits mithilfe von KI-Methoden Systeme überwacht und technische Probleme noch vor ihrem Auftreten identifiziert werden, dadurch wird es ermöglicht, Systeme und Strukturen rechtzeitig instand zu halten – Maschinen zu reparieren, noch bevor sie defekt sind. Das DLR Institut für Raumfahrtantriebe untersucht in diesem Zusammenhang bspw. neuronale Netze für den Entwurf und Betrieb von Flüssigraketentriebwerken [Waxenegger-Wilfing u. a. (2019)]. Dabei geht es u.a. um den Einsatz neuronaler Netze für die Regelung und Zustandsüberwachung von Raketentriebwerken. In der *Luft- und Raumfahrt* sind bereits sog. intelligente Astronautenassistenten im All im Einsatz [Karrasch (2018)]. Sie sollen die Astronauten bei ihren Arbeiten unterstützen, vor Gefahren warnen und sie vor

allem bei Routineaufgaben entlasten. Der intelligente Astronautenassistent CIMON (**C**rew **I**nteractive **MO**bile companio**N**) ist der erste Roboter mit Mensch-Maschine-Interaktion und künstlicher Intelligenz im All.

Die eben vorgestellten Einsatzbereiche und Anwendungsgebiete sollten lediglich einen ersten Einblick in die Notwendigkeit der Entwicklung und Erforschung intelligenter Maschinen und ihrer Anwendungen geben. Um das KI-Potenzial weiter voranzutreiben ist eine effektive Nutzung der von Sensoren erfassten Daten notwendig. Sensoren gelten damit als einer der wichtigsten Datenlieferanten, denn nur mit ihnen können Zustände erfasst und Aktionen ausgeführt werden. Sie werden immer kleiner, günstiger und energieeffizienter und sind aus dem Alltag nicht mehr wegzudenken. Insbesondere MEMS-Sensoren haben nach [Titterton u. a. (2004)] eine der interessantesten Entwicklung in den letzten 30 Jahren erfahren und konnten sich mittlerweile in zahlreichen Anwendungen und Bereichen etablieren. Vor allem im Bereich der *Navigation* bieten MEMS-Inertialsensoren mit ihren Eigenschaften ganz neue Möglichkeiten in der *Inertialnavigationstechnologie*. Allerdings wird die Qualität der Messergebnisse von MEMS-Sensoren im Wesentlichen auch von ihren Fehlereinflüssen beeinträchtigt. Einer der Hauptfehlerquelle stellt der Sensorparameter *Zero Bias* dar. Dieser wirkt additiv und führt in den Berechnungen zu einem Anwachsen der Fehler der Geschwindigkeit, Position und der Lage (vgl. [Wendel (2011)]).
In der *Landfahrzeugnavigation* beispielsweise können MEMS-Inertialsensoren bisher nur als Ergänzung zu GNSS (Global Navigation Satellite System) verwendet werden. Dabei funktioniert satellitengestützte Navigation für Landanwendungen nur solange, wie man sich im offenen Gelände aufhält. Diese optimalen Bedingungen sind nicht immer gegeben. In signalschwachen oder abgeschatteten Bereichen, wie z.B. Häuserschluchten, Fahrten durch Tunnel, Gebirge oder Strecken mit einer erhöhten Verkehrsdichte, kann es zu Signaldämpfungen, Signalverlusten oder Mehrwegeeffekten der empfangenen Satellitensignale kommen.

Die Ausnutzung von Trägheitsdaten für eine genaue und zuverlässige Navigation und Lokalisierung hat dabei nicht nur das industrielle Interesse geweckt, sondern spielt auch bereits in unserer Gesellschaft eine große Rolle. Denn die Mehrheit-, der sich im Umlauf befindlichen Smartphones sind mit kostengünstigen MEMS-Inertialsensoren ausgestattet. Eine inertiale Navigation auf Basis dieser Inertialsensoren wäre wünschenswert, gestaltet sich bisher aber schwierig. Eine Anwendung in der Navigation zur relativen Bestimmung einer Position ist zurzeit nur bei Durchführung einer vollständigen Kalibrierung der Sensoren kombiniert mit einer externen Referenzierung nach kurzen Messzeiten als Korrekturmaßnahme denkbar. Bei einem

Inertialnavigationssystem (INS) kommen Beschleunigungssensoren und Drehratensensoren zum Einsatz. Durch die erfassten Messwerte aus Beschleunigungen und Winkelgeschwindigkeiten über die Zeit, werden mittels Integration kontinuierlich die Geschwindigkeit, Position und Orientierung im Raum bestimmt. Ein Magnetometer kann zusätzlich wichtige Messungen über das Erdmagnetfeld liefern. In Kombination mit dem Beschleunigungssensor können zusätzliche Stützinformationen über die Lage gewonnen werden, was folglich zur Stabilisierung und Verbesserung der Genauigkeit beitragen kann (vgl. [Wendel (2011)]).
In verschiedenen Arbeiten [Moder (2011), Willemsen (2016)] wurden Low-cost-MEMS-Inertialsensoren mit gängigen Strapdown-Algorithmen untersucht. Die Ergebnisse machten deutlich, dass eine Positionsbestimmung ohne ein geeignetes Referenzsystem (bspw. satellitengestützte Navigation) zu ungenau ist und dass mit derartigen Sensoren keine brauchbaren Lösungen für eine Anwendung erzielt werden können.

Die Realisierung eines inertialen Navigationssystems mittels Low-cost-Sensoren kombiniert mit einem satellitengestützten Referenzsystem zur Korrektur, mag am Anfang den Anschein erwecken, die beste Wahl für eine zuverlässige, genaue und effiziente Navigation zu sein. Diese Kombination beinhaltet jedoch auch eine Vielzahl erheblicher Schwächen und technischer Herausforderungen.
In den nachfolgenden Kapiteln werden die Ziele und die Umsetzung der vorliegenden Arbeit näher erläutert und definiert.

1.1. Motivation

Die Inspiration der Arbeit entstand aus der Idee heraus mit dem Smartphone eine einfache inertiale Navigation entwickeln zu wollen. Die algorithmischen Herausforderungen und die Komplexität des Gebiets der Navigation bzw. der inertialen Navigation zusammen mit den Schwierigkeiten, die alleine Low-cost-Sensoren mit sich bringen, waren letztendlich Anreiz genug, um sich mit diesem Themengebiet noch näher zu befassen. Typischerweise verwendet man für eine zuverlässige und integre Positionsbestimmung in der Navigation neben den Inertialsensoren auch ein geeignetes Referenzsystem für zusätzliche Stützinformationen. Daher stellt die Kombination zweier sich ergänzender Systeme aus Inertialsensoren und einem satellitengestützten Navigationsverfahren eine logische Wahl dar.
Die GPS Technologie hat sich längst als weltweite Navigationslösung im Alltag etabliert, denn sie bietet stets aktuelle und der Nutzerumgebung passende Informationen. Allerdings weist die Technologie auch einige Defizite auf. Zum Beispiel belastet GPS neben der mangelnden

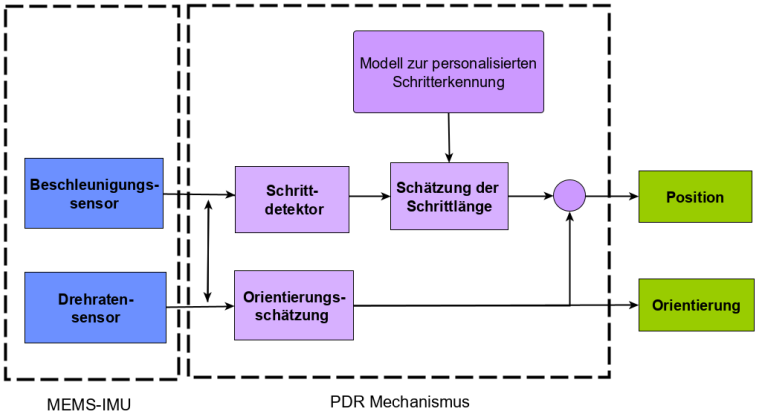
Zuverlässigkeit in signalschwachen oder abgeschatteten Bereichen zudem noch die Akkuleistung besonders stark. Hinzu kommt, dass einige Geräte bei der Verwendung von GPS häufig kollaborativ mit WiFi arbeiten, wodurch der Einfluss auf den Energieverbrauch zusätzlich belastet wird. MEMS Sensoren hingegen haben nur einen geringen Leistungsbedarf und einen niedrigen Energieverbrauch, dazu sind sie bereits permanent in Betrieb. Zur Umsetzung einer Navigationslösung müssen lediglich die Sensordaten effizient verarbeitet und visualisiert werden.

Im Rahmen einiger bereits im Studium erfolgreich durchgeführter Projekte im Bereich des Machine Learnings auf Basis von neuronalen Netzen-, entstand die Idee, dass Problem der inertialen Navigation mithilfe von Techniken bzw. Ansätzen aus der künstlichen Intelligenz zu betrachten.

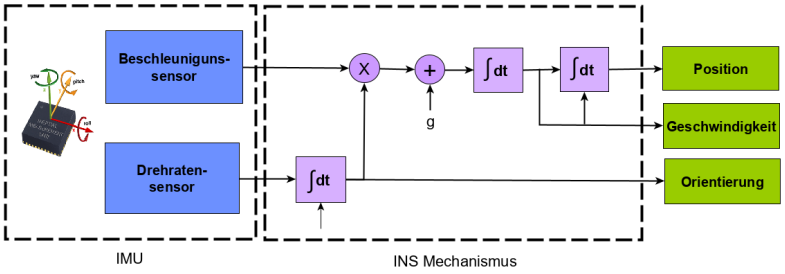
1.2. Zielsetzung

Das Ziel der vorliegenden Arbeit soll sein, auf Basis von Deep-Learning-Methoden einen Lösungsansatz zur inertialen Navigation mittels Low-cost-Inertialsensoren vorzustellen. Dazu soll ein entsprechendes Modell entworfen, implementiert und im Anschluss näher untersucht werden.

Eine inertiale Navigation auf Basis von kostengünstigen Inertialsensoren ist mit konventionellen Algorithmen, wie sie bei Strapdown-Systemen zum Einsatz kommen, nicht realisierbar. Die Fehlereinflüsse der Sensoren verursachen fehlerbehaftete Messwerte, welche bei Strapdown-Systemen zu einer exponentiellen Fehlerausbreitung führen. Vor allem die numerische Integration verursacht einen mit der Zeit zunehmenden Schleppfehler, wodurch sich die Fehler aufsummieren und folglich hohe Abweichungen erzeugen, was eine relative Bestimmung der Position unmöglich macht. Weitaus vielversprechendere Ansätze in der Navigation bezüglich Low-cost-Sensoren bieten die PDR-Algorithmen. Sie haben die Möglichkeit die Fehlereigenschaften der Sensoren zu kompensieren und eine Navigationslösung zu erlauben. Obwohl mit PDR-Algorithmen weitaus bessere Ergebnisse erzielt werden können, eignen sich die Algorithmen ohne ein geeignetes Korrektursystem eher für kleinere Strecken, da ansonsten die Fehler zu stark anwachsen würden. Die Funktionsprinzipien beider eben erwähnten Algorithmen sind in Abbildung 1.1 als Architektur übersichtlich dargestellt. Die Abbildung 1.1b präsentiert die einzelnen Phasen, die zu einem gewöhnlichen, schritt-basierten PDR-Algorithmus gehören. Hierauf-, wird aber auch in den nachfolgenden Kapiteln noch



(a) Pedestrian Dead Reckoning (PDR).



(b) Strapdown Inertial Navigation System (SINS).

Abbildung 1.1.: Funktionsprinzipien häufig verwendeter INS Methoden: PDR und SINS.

näher eingegangen. Die Abbildung 1.1a demonstriert stattdessen den Mechanismus eines konventionellen Strapdown-Systems. Für beide Algorithmen ist die Qualität der Sensoren bzw. Sensordaten ein wesentlicher Faktor.

Um dem Kreislauf der kontinuierlichen Fehlerfortpflanzung entgegenzuwirken, soll das Problem stattdessen als ein Problem aus dem Bereich der künstlichen Intelligenz betrachtet und mit lern-basierten Methoden durch Experimente zunächst evaluiert werden. Bei dem vorgeschlagenen lern-basierten Ansatz handelt es sich um ein Ensemble DL-Modell, bestehend aus den Convolutional Neural Networks (CNN) und den Recurrent Neural Networks (RNN). Für die Umsetzung des lern-basierten Ansatzes ist zudem eine ausreichender Menge beschrifteter Daten zum Trainieren, Validieren und Testen notwendig. Aufgrund begrenzter Verfügbarkeit von inertialen Navigationsdaten wurde im Rahmen der Arbeit ein eigener Datensatz erstellt. Dieser umfasst dabei die folgenden zwei Bewegungsformen:

- **Foot-Mounted:** ein am Fuß getragener Sensor
- **Handheld:** ein in der Hand getragener Sensor

Die Daten wurden insgesamt mit drei verschiedenen Sensoren aus unterschiedlichen Geräten aufgezeichnet: ein kostengünstiger IMU-9DoF-Sensor, iphone 5 und iphone 6. Bei den verwendeten Inertialsensoren handelt es sich um eine Kombination aus 3-achsigen Beschleunigungssensoren sowie Drehratensensoren und einem Magnetometer. Des Weiteren wurde besonders darauf Wert gelegt, dass der Datensatz sich an einem realen Anwendungsszenario orientiert, wo das Tragen der Sensoren auch den täglichen Gebrauch des Anwenders widerspiegelt. Der aufgezeichnete Datensatz wurde allerdings auf zwei verschiedene Laufstrecken begrenzt. Neben dem Sammeln ausreichender Trainingsdaten sind eine adäquate Vorverarbeitung der Daten und das entsprechende Labeln wesentliche Bestandteile der Arbeit. Wobei mit dem Labeln der Daten in dieser Arbeit die Erzeugung geeigneter Ground-Truths (Grundwahrheitswerte) beider Laufstrecken gemeint ist. Für jede Strecke wurden jeweils zwei Ground-Truths erstellt. Eine Ground-Truth beinhaltet jeweils die exakten Positionsdaten, während die andere Ground-Truth die aus den Positionsdaten abgeleiteten Orientierungsdaten (Heading Daten) enthält. Eine Trajektorie aus den Positionsdaten lässt sich aus den Positionsdaten ohne weiteres erzeugen, bei den Heading-Daten wird dafür zusätzlich die *Rho-Theta*-Technik nach [Hofmann-Wellenhof u. a. (2003)] zum Einsatz kommen. Rho-Theta beschreibt dabei ein Prinzip in der Navigation, wodurch mithilfe eine auf Distanz- und Richtungsmessung basierende Positionsbestimmung ermöglicht werden kann. Mit dem Hintergrund, den DL-Algorithmus vor zwei unterschiedliche lern-basierte Aufgaben zu stellen, welche schließlich zum selben Ziel führen, sollen die Resultate nach Beendigung der Lernphasen auf ihre Verallgemeinerungsfähigkeit hin untersucht, sowie im Hinblick, was die Praxistauglichkeit der entwickelten Lösungen betrifft, evaluiert werden.
Zum Schluss soll aus den (fehlerbehafteten) Rohdaten der Sensoren eine vollständige Trajektorie gewonnen werden. Diese soll als Resultat mit den entsprechenden Ground-Truths und einer auf PDR-basierten Methode verglichen werden. Die Abweichung wird als Differenz zwischen der vom Modell geschätzten Position und der tatsächlichen Position (Ground-Truth) gemessen. Des Weiteren soll eine Trajektorie Erstellung sowie eine kumulative Verteilungsfunktionen (engl. cumulative distribution function (CDF)) dazu beitragen die Ergebnisse adäquat zu vergleichen, zu veranschaulichen und im Anschluss zu Analysieren. Diese Arbeit soll dazu beitragen auch mit Low-cost-Inertialsensoren, wie sie mittlerweile in jedem Smartphone zu finden sind, eine relative Bestimmung der Position zu ermöglichen. Die Abbildung 1.2 soll den Umfang der Arbeit noch einmal als vereinfachtes Ablaufdiagramm verdeutlichen.

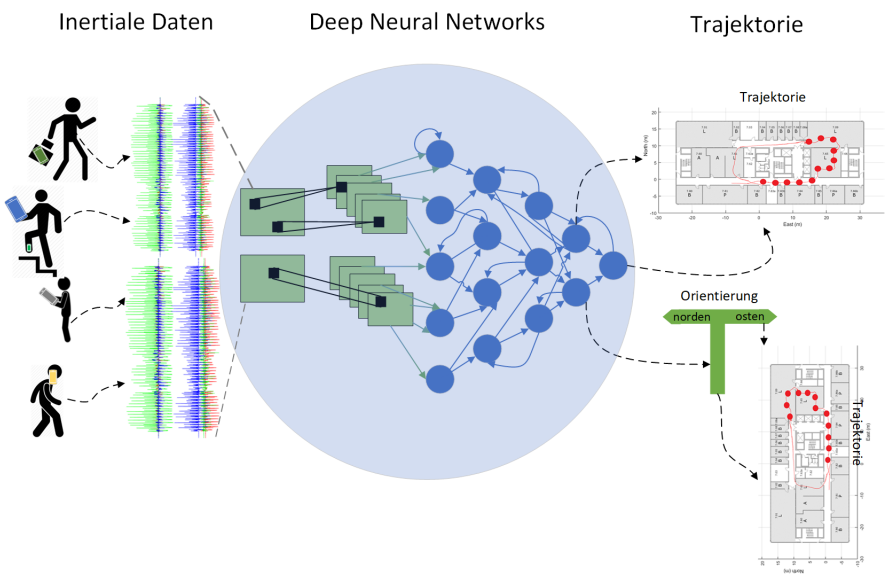


Abbildung 1.2.: Zielsetzung der Arbeit als Schematische Darstellung mit Deep-Learning Modell.

1.3. Inhaltlicher Aufbau der Arbeit

Die vorliegende Arbeit umfasst insgesamt acht Kapitel. Dabei soll der Leser systematisch entlang des roten Fadens durch die vom Autor durchgeführten Arbeitsschritte begleitet werden.

Kapitel 1 gibt zunächst eine grobe Einführung in das Thema und nähert sich nach und nach dem eigentlichen Kern der Arbeit. Im Anschluss folgt die Motivation und eine ausführliche Zielsetzung dieser Arbeit.

Kapitel 2 befasst sich mit dem Grundwissen, damit die einzelnen Arbeitsschritte in dieser Thesis nachvollzogen werden können.

Kapitel 3 gibt eine Übersicht über aktuelle verwendete PDR-basierte Verfahren, gefolgt von ersten DL-Ansätzen und wie sie die inertialen Navigationssysteme bereits unterstützen.

Kapitel 4 beschreibt die Erstellung, den Umfang sowie die notwendige Vorverarbeitung des Datensatzes. Außerdem wird ein schritt-basierter PDR Algorithmus implementiert, welcher für die Ground-Truth Erzeugung ein elementarer Bestandteil gewesen ist.

Kapitel 5 stellt die Deep-Learning Systemarchitektur vor, präsentiert dazu die einzelnen DL-Techniken und erläutert ihre Aufgaben und verdeutlicht den Nutzen ihrer strategischen Kombination.

Kapitel 6 beschreibt den Versuchsaufbau, die Durchführung der Experimente und präsentiert die Ergebnisse.

Kapitel 7 diskutiert, interpretiert die Ergebnisse.

Kapitel 8 fasst die wichtigsten Erkenntnisse der Arbeit zusammen. Abschließend wird ein Fazit über die Ergebnisse gezogen und im Zuge dessen ein Ausblick über mögliche Weiterentwicklungen gegeben.

2. Theoretische Grundlagen

In diesem Kapitel wird zunächst auf die allgemeinen Grundlagen der inertialen Navigation eingegangen. Es folgt eine Einführung in die MEMS-Sensorik mit den für ein Inertialesnavigationssystem wichtige Sensoren und deren Fehlereigenschaften. Zum Schluss folgt eine grundlegende Beschreibung der in dieser Arbeit verwendeten Deep-Learning Modelle.

2.1. Inertiale Navigation

Ein inertiales Navigationssystem (INS) bestimmt fortlaufend die Position, Geschwindigkeit und Orientierung im Raum mittels Integration der erfassten Beschleunigungen und Winkelgeschwindigkeiten der zurückgelegten Strecke über die Zeit.
Der Ursprung der Navigation geht bis ins 13. und 14. Jahrhunderte zurück, denn schon damals wurden Navigationsinformationen bei einer Vielzahl von Anwendungen benötigt. Vor allem in der Seefahrt, war man darauf angewiesen. Den Kurs und die Position genau zu kennen spielte schon damals, eine wichtige Rolle. Die Konsequenzen fehlerhafter Informationen in der Navigation führten zum Verlust unzähliger Schiffe. In dieser Zeit benutzte man eine frühe Form der Koppelnavigation: Die zurückgelegte Strecke wurde mit einem Log und der Kurs mit dem Kompass bestimmt. Schon damals summierten sich dabei die Fehler mit der Zeit auf und es wurden Stützinformation benötigt. So wurde zusätzlich versucht mittels Sextant anhand des Sonnenstandes den Breitengrad zu ermitteln (die Bestimmung des Längengrades blieb in der damaligen Zeit noch lange ein Problem). Ab Mitte des 20. Jahrhunderts gewann die *inertiale Navigation* kontinuierlich an Bedeutung. Auch bei der *inertialen Navigation* handelt es sich um ein Koppelnavigationsverfahren. Bei der Koppelnavigation (engl. dead reckoning) werden kontinuierlich die Position, die Geschwindigkeit und Orientierung mittels Integration der erfassten Beschleunigungen und Winkelgeschwindigkeiten über die Zeit bestimmt. Hierbei summieren sich die Fehler ebenfalls mit der Zeit auf und es werden zusätzliche Stützinformationen benötigt. Aus diesem Grund werden heutzutage vielfach sog. *integrierte Navigationssysteme* eingesetzt, die dadurch gekennzeichnet sind, dass die verschiedensten Navigationsverfahren und Sensoren kombiniert werden. Ziel ist hierbei, durch die Vorteile des einen Navigationsverfahrens die Nachteile eines anderen Navigationsverfahrens zu kompensieren und eine gewisse Redundanz

zu schaffen. Dabei ist die Kombination von inertialer Navigation mit GPS weit verbreitet, da sich diese Verfahren ausgezeichnet ergänzen (vgl. [Wendel (2011)]).

Die zugrundeliegende Technik der Inertialnavigationssysteme (INS) sind die sog. Strapdown-Systeme, die ab Mitte der 60er Jahre mit dem Aufkommen der Ringlaserkreisel (RLG) erst technisch realisierbar wurden. Die Basis des Strapdown-Systems bildet eine Inertialsensoreinheit, bestehend aus drei Beschleunigungs- und drei Drehratensensoren. Die Inertialsensoreinheit wird auch als *Inertial Measurement Unit (IMU)*[1] bezeichnet. Die Drehratensensoren dienen zur Erfassung der Lageänderung, welche durch Integration der Drehratensensordaten Lageinformationen eines körperfesten Koordinatensystems liefern. Diese gilt es in ein Koordinatensystem mit raumfesten Koordinatenrichtungen, am häufigsten ein Nord-Ost-Unten-Koordinatensystem, umzurechnen. Die Beschleunigungen werden im Anschluss zu Geschwindigkeiten integriert und zur Bestimmung der Position erneut integriert. Die Kombination der Messungen zur Berechnung von Position, Geschwindigkeit und Orientierung erfolgt mit dem Strapdown-Algorithmus. Die vollständige Rechnung des Strapdown-Algorithmus ist in ([Wendel (2011)] S.45) beschrieben. Eine vereinfachte Form des Strapdown-Algorithmus ist als Blockdiagramm in Abbildung 1.1a dargestellt.

Im Fall von Low-cost-MEMS-Sensoren sind die gängigen Strapdown-Systeme kein geeignetes Mittel für eine inertiale Navigation. Hierfür kommen weiterhin nur hochwertige Inertialsysteme infrage, da die Einsatzfähigkeit solcher Inertialnavigationssysteme maßgeblich von der Qualität der verfügbaren Sensordaten abhängt. Aus diesem Grund werden die wesentlichen Fehlercharakteristiken von MEMS-Sensoren und INS im nachfolgenden Kapitel 2.2 näher erläutert.

Obwohl Strapdown-Systeme heutzutage den Standard in der inertialen Navigation darstellen, eignen sie sich in Verbindung mit Low-cost-Sensoren nicht, stattdessen bieten sich alternativ das sog. *PDR (Pedestrian Dead Reckoning)* als Verfahren an. Bei *PDR* handelt es sich um ein Koppelnavigationsverfahren zur relativen Positionsbestimmung.
Die Voraussetzung von PDR ist die korrekte Übertragung bzw. Erfassung menschlicher Körperbewegungen auf den verwendeten Sensor. Anhand dieser Bewegungen können Drehungen und Geschwindigkeiten registriert und aufgezeichnet werden, welche im Anschluss eine rela-

[1] Gelegentlich wird noch zwischen Inertial Sensor Assembly (ISA), d. h. dem eigentlichen Inertialsensor und Inertial Measurement Unit (IMU), d. h. der Sensorik in Verbindung mit der Elektronik zur Bereitstellung einer Schnittstelle (einem Analog-Digitalwandler mit entsprechender Elektronik zur Vorverarbeitung der Signale wie bspw. Ausreißerdetektion o. ä.), unterschieden.

tive Bestimmung der Position ermöglichen. Während der Navigation muss die Ausrichtung des Sensors beibehalten werden.Es existieren zahlreiche Möglichkeiten und Methoden einen PDR-Algorithmus erfolgreich umzusetzen. Dabei kann zwischen einem inertial-basierten oder einen schritt-basierten Algorithmus unterschieden werden. Der schritt-basierte Ansatz wird in drei Phasen unterteilt:

- Schrittdetektion
- Schrittlängenschätzung
- Richtungsschätzung.

Obwohl der PDR-Algorithmus in der Indoor-Navigation eine wirksame und beliebte Technik darstellt, besteht bei dieser Methode die Gefahr, dass sich Fehler schnell ansammeln (z. B. fehlerhafte Kalibrierung der Schrittlängenschätzung des Läufers). Jede Phase umfasst eine große Anzahl von Parametern, die sorgfältig abgestimmt werden müssen, weswegen das PDR-Verfahren auch ein gewisses Fehlerpotential beinhaltet. Für die *Ground-Truth* Erzeugung war es notwendig einen solchen PDR-Algorithmus zu implementieren. In Kapitel 4.3 wird dieser vorgestellt und näher erläutert. Zwei konventionelle PDR-Algorithmen, deren Umsetzung und Implementation nachvollziehbar und verständlich erläutert wird, sind in [Jin u. a. (2011), Park und Suh (2010a)] zu finden. Weitere optimierte Varianten des PDR werden in Kapitel 3 vorgestellt.

2.1.1. Inertiale Navigationssysteme

Die Basis eines inertialen Navigationssystems beruht auf der Grundidee, die für eine Koppelnavigation notwendigen Informationen durch Messungen von Beschleunigungen und Drehraten zu gewinnen. Dafür werden drei orthogonal zueinander angeordnete Beschleunigungssensoren und Drehratensensoren benötigt.

Die ersten inertialen Plattformen waren kardanisch gelagerte Systeme oder auch Gimbalsysteme. Dabei wurden Beschleunigungssensoren auf einer kardanisch gelagerte, stabilisierte Plattform montiert, wodurch die Achsen von den Bewegungen des Fahrzeugs o. Ä. entkoppelt wurden und in Richtung *Norden*, *Osten* und *unten* zeigten. Die Drehratensensoren hatten die Aufgabe, Lageänderungen der Plattform zu detektieren. Dabei wurden nur kleine Drehraten gemessen, da die Plattform der Bewegung selbst entgegenwirkt.

Strapdown Inertial Navigation Systems (SINS) oder einfach Strapdown-Systeme wurden ab Mitte der 60er Jahre mit dem Aufkommen der Ringlaserkreisel (RLG) technisch erst realisierbar.

Dank der Strapdown-Systeme waren Plattformsysteme nicht mehr notwendig gewesen, was wesentliche Vorteile mit sich brachte. Es gab keine komplexe oder wartungsintensive Mechanik mehr und wegen des geringen Platzbedarfs gab es auch einen erheblichen Gewichtsvorteil. Damit stellen Strapdown-Systeme heute den Standard der inertialen Navigation dar.

Bei AHRS (Attitude Heading Reference System) handelt sich um stark vereinfachte inertiale Navigationssysteme. Im Gegensatz zu Strapdown-Systemen sind bei AHRS keine hochwertigen Inertialsensoren notwendig. AHRS können zur 3D-Orientierung sowie zur Positionsbestimmung von Benutzern und Geräten verwendet werden. Es kommen dabei üblicherweise Beschleunigungssensoren, Drehratensensoren und Magnetometer zum Einsatz. AHRS-Algorithmen liefern vor allem in Umgebungen mit potenziellen externen Störungen eine zuverlässige bzw. robuste Leistung. Um ein AHRS zu implementieren, existieren diverse Ansätze von Fusionsalgorithmen [Madgwick u. a. (2011)]1582367wendel2011integrierte. Die meisten dieser Ansätze basieren dabei entweder auf einem Kalman-Filter oder einem Komplementärfilter. Diese Filter machen es möglich die Fehlercharakteristiken von Sensordaten einzuschränken und zu kompensieren, weshalb sie sich stark für den Einsatz in kostengünstigeren Sensoren empfehlen [Diaz u. a. (2015)].

2.1.2. Koordinatensysteme

Inertiale Sensoren erfassen physikalische Größen bezüglich eines Inertialkoordinatensystems, Koordinaten bezüglich eines erdfesten Referenzsystems sind gegeben und die Koordinatenrichtungen in *Norden*, *Osten* und *unten* werden benötigt. In der Navigation werden dafür eine Reihe von Koordinatensystemen benötigt, weshalb nachfolgend die dafür notwendigen Systeme kurz erläutert und in Abbildung 2.1 veranschaulicht sind.

- Die Achsen des **körperfesten Koordinatensystems (b-frame)** sind fest im Bezug zu einem Fahrzeug und weisen in die Fahrzeuglängsrichtung (x^b), nach rechts (y^b) und unten (z^b). Bei einer orthogonalen Ausrichtung der Achsen des Inertialsensors fallen diese auch exakt mit den Achsen des körperfesten Koordinatensystems zusammen. Damit fallen auch entsprechend die Messwerte des IMU in dieses Koordinatensystem.

- Der Mittelpunkt des **Inertialkoordinatensystems (i-frame)** hat den Ursprung im Mittelpunkt des Rotationsellipsoids, der sich der Erdgestalt annähert. Die (x^i) und (y^i) des Koordinatensystems liegen in Äquatorebene, die (z^i)-Achse fällt mit der Rotationsachse der Erde zusammen. Die IMU misst Beschleunigungen und Drehraten des körperfesten

Koordinatensystems bzgl. des Inertialkoordinatensystems.

- Der Ursprung des **erdfesten Koordinatensystem (e-frame)** entspricht demselben wie beim Inertialkoordinatensystem. Die Koordinatenachsen sind fest im Bezug auf die Erde. Die (z^e)-Achse fällt mit der (z^i)-Achse zusammen. Die Bezeichnung ω entspricht der Winkelgeschwindigkeit der (z^e)-Achsen-Rotation des erdfesten Koordinatensystems bezüglich des Inertialkoordinatensystems. Des Weiteren wird das erdfeste Koordinatensystem auch als earth centered, earth fixed (ECEF) Koordinatensystems beschrieben.

- Der Ursprung des **Navigationskoordinatensystems (n-frame)** fällt mit dem des körperfesten Koordinatensystems zusammen. Die (x^n)-Achse, sowie die (y^n)-Achse sind in Nord- bzw. Ostrichtung ausgerichtet und liegen in der Tangentialebene an dem Erdellipsoid. Die (z^n)-Achse zeigt nach unten und ist parallel zu der Schwerebeschleunigung. Die Komponenten eines Vektors des Navigationskoordinatensystems werden mit n,e und d (north, east und down) bezeichnet.

Weiterführende Literatur zu Koordinatensystemen und deren Transformationen können in [Wendel (2011)] sowie in der gängigen Literatur wie [Hofmann-Wellenhof u. a. (2003)] nachgeschlagen werden. Die Messdaten eines Inertialnavigationsystems werden meistens in einem

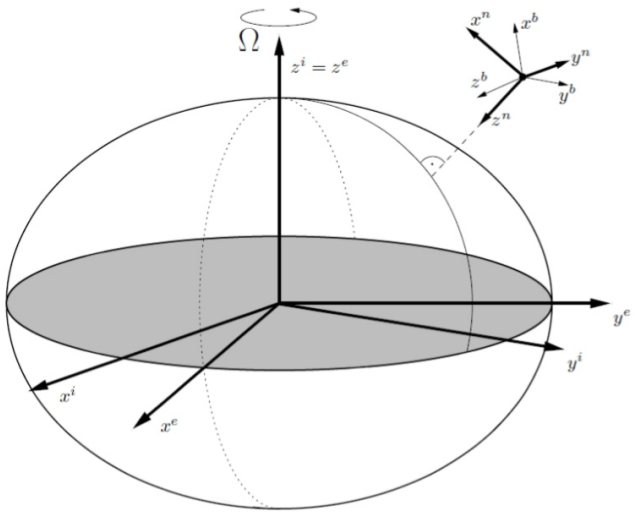


Abbildung 2.1.: Darstellung unterschiedlicher Koordinatensysteme für die Navigation (vgl. [Wendel (2011)], S.28)

b-frame gemessen, in einem n-frame berechnet und in einem entsprechenden e-frame dargestellt.

2.1.3. Lagedarstellung

Wie oben bereits erwähnt, erfassen inertiale Sensoren die Beschleunigungen und Drehraten bezüglich eines Inertialkoordinatensystems im b-frame. Die dafür notwendigen Koordinatensysteme wurden im vorherigen Kapitel beschrieben. Nun folgt eine beispielhafte Betrachtung zweier Koordinatensysteme und deren Achsenrotation zur Ausrichtung eines Koordinatensystems parallel zu den Achsen in ein anderes Koordinatensystem. Für diese beispielhafte Betrachtung werden das Navigationskoordinatensystem und das körperfeste Koordinatensystem gewählt. Die Lagen der Koordinatensysteme können dabei wie folgt unterschieden und beschrieben werden:

- Eulerwinkel
- Orientierungsvektor und Quaternionen
- Richtungskosinusmatrix

Eulerwinkel

Eine gängige Methode, um Rotationen im dreidimensionalen Raum zu beschreiben, ist die Angabe durch Eulerwinkel. Die drei Eulerwinkel *roll*, *pitch* und *yaw* beschreiben drei Drehungen, die nacheinander ausgeführt das Navigationskoordinatensystem in das körperfeste Koordinatensystem überführen. Sie sind auch als Roll-, Nick- und Gierwinkel bekannt. Der yaw-Winkel ψ, der auch auch als Azimuth bezeichnet wird, beschreibt eine Drehung um die z-Achse des Navigationskoordinatensystems. Außerdem spricht man bei yaw in der Navigation meist auch vom *heading*, weil dies auch direkt der Bewegungsrichtung eines Körpers entspricht.
Im Anschluss wird um die neue y-Achse des rotierenden Koordinatensystems, d. h. um den pitch θ bzw. Nick-Winkel gedreht. Zum Schluss erfolgt die letzte Drehung der neuen x-Achse um den roll Winkel ϕ.

Ein Nachteil, die Eulerwinkel mit sich bringen, ist das Auftreten von Singularitäten. So tritt bei der Berechnung der Änderung für den roll-Winkel eine Division durch 0 auf, wenn der pitch-Winkel $\pm$ 90 Grad annimmt. Der yaw-Winkel beschreibt eine Drehung um die lokale Vertikale. Wird im Anschluss um den pitch-Winkel von $\pm$ 90 Grad gedreht, handelt es sich wie bei der durch den roll-Winkel beschriebenen Drehung ebenfalls um eine Drehung um die lokale Vertikale. In diesem Fall sind die Eulerwinkel nicht mehr eindeutig. Das Problem ist

auch als gimbal-lock oder kardanische Blockade bekannt und tritt ebenfalls unvermeidlich bei Plattform-Systemen auf (vgl. [Wendel (2011)]).

Um eine Winkelbestimmung mittels Drehratensensoren durchführen zu können, ist eine vorherige Transformation der Drehraten in das entsprechende Bezugssystem notwendig. Die Umrechnung in ein anderes Koordinatensystem kann mithilfe der Richtungskosinusmatrix C_b^n erfolgen. Die Richtungskosinusmatrix C_b^n kann als Funktion der Eulerwinkel angegeben werden.

$$C_b^n = \begin{pmatrix} \cos\theta\ \cos\psi & -\cos\phi\ \sin\psi + \sin\phi\ \sin\theta\ \cos\psi & \sin\phi\ \sin\psi + \cos\phi\ \sin\theta\ \cos\psi \\ \cos\theta\ \sin\psi & \cos\phi\ \cos\psi + \sin\phi\ \sin\theta\ \sin\psi & -\sin\phi\ \cos\psi + \cos\phi\ \sin\theta\ \sin\psi \\ -\sin\theta & \sin\phi\ \cos\theta & \cos\phi\ \cos\theta \end{pmatrix} \tag{2.1}$$

Die Herleitung dieses Zusammenhangs wird in dieser Arbeit nicht näher erläutert, kann aber in der folgenden Literatur nachgeschlagen werden: [Titterton u. a. (2004)] und [Wendel (2011)]. Umgekehrt ist es auch möglich Eulerwinkel als Funktion der Koeffizienten der Richtungskosinusmatrix darzustellen:

$$\begin{aligned} \phi &= \arctan_2(c_{32,c_{33}}) \\ \theta &= \arcsin(-c_{31}) \\ \psi &= \arctan_2(c_{21,c_{11}}) \end{aligned} \tag{2.2}$$

Die zeitliche Änderung der Eulerwinkel aufgrund der Drehraten $\vec{\omega}_{nb}^b$ wird durch die folgenden Differentialgleichungen dargestellt:

$$\dot{\phi} = (\vec{\omega}_{nb,y}^b\ \sin\phi\ +\ \vec{\omega}_{nb,z}^b\ \cos\phi)\ \tan\theta\ +\ \vec{\omega}_{nb,x}^b \tag{2.3}$$

$$\theta = \vec{\omega}_{nb,y}^b\ \cos\phi\ -\ \vec{\omega}_{nb,z}^b\ \sin\phi \tag{2.4}$$

$$\psi = (\vec{\omega}_{nb,y}^b\ \sin\phi\ +\ \vec{\omega}_{nb,z}^b\ \cos\phi)\ /\ \cos\theta \tag{2.5}$$

Die entsprechende Herleitung kann ebenfalls in [Titterton u. a. (2004), Wendel (2011)] nachgeschlagen werden. Eine weitere Möglichkeit zur Beschreibung der Rotationen kann mittels Quaternionenalgebra erfolgen. Die Technik ist komplexer, jedoch eleganter und findet häufiger Verwendung wie bspw. in der Luft- und Raumfahrt, Schifffahrt als auch in der Robotertechnik (vgl. [Dam u. a. (1998)]).

Orientierungsvektor und Quaternionen

Die Lage zweier Koordinatensysteme kann durch einen Orientierungsvektor $\vec{\sigma}$ beschrieben werden. Dabei legt der Orientierungsvektor die Achsen im Raum fest, um die gedreht werden muss, um mit einer einzigen Drehung beide Koordinatensysteme ineinander überführen zu können. Der Winkel, um den gedreht werden muss, wird von der Länge durch die des Orientierungsvektors angegeben. Der Orientierungsvektor

$$\vec{\sigma}_1 = (\sigma_{1,x}\ \sigma_{1,y}\ \sigma_{1,z})^T \tag{2.6}$$

beschreibt mit der Länge $\sigma_1 = |\vec{\sigma}|$ die Lage zweier Koordinatensysteme. Ebenfalls beschreibt der Orientierungsvektor

$$\vec{\sigma}_2 = -\frac{\vec{\sigma}_1}{\sigma_1}\ (2\pi\ -\ \sigma_1) \tag{2.7}$$

dieselbe Lage. Der Hintergrund hierbei ist, dass eine Überführung der Koordinatensysteme sowohl durch eine Rechtsdrehung um den Winkel σ_1 als auch durch eine Linksdrehung um den Winkel $2\pi\ -\ \sigma_1$ möglich ist.
Um die Änderung des Orientierungsvektors in Abhängigkeit von Drehraten $\vec{\omega}^b_{nb}$ erfassen zu können, muss die Bortzsche Orientierungsvektordifferentialgleichung

$$\dot{\vec{\sigma}} = \vec{\omega}^b_{nb} + \frac{1}{2}\vec{\sigma} \times \vec{\omega}^b_{nb} + \frac{1}{\sigma^2}\left(1 - \frac{\sigma \sin\sigma}{2(1-\cos\sigma)}\right)\vec{\sigma} \times (\vec{\sigma} \times \vec{\omega}^b_{nb}) \tag{2.8}$$

gelöst werden. Der Orientierungsvektor kann als auf die Länge $|\mathbf{q}| = 1$ normiertes Quaternion gespeichert werden:

$$\mathbf{q}^n_b = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \cos(\sigma/2) \\ (\sigma_x/\sigma)\sin(\sigma/2) \\ (\sigma_y/\sigma)\sin(\sigma/2) \\ (\sigma_z/\sigma)\sin(\sigma/2) \end{pmatrix} \tag{2.9}$$

Bildet man nun die Quaternionen zu den Orientierungsvektoren Gl. (2.6) und Gl. (2.7), so findet man $\mathbf{q}_2 = -\mathbf{q}_1$. Die Lage zweier Koordinatensysteme kann sowohl durch $\mathbf{q}$ als auch durch -$\mathbf{q}$ beschrieben werden. Ebenfalls ist wie bei den Richtungskosinusmatrizen mit $\mathbf{q}^n_b = \mathbf{q}^n_e \bullet \mathbf{q}^e_b$

eine Verkettung von Drehungen möglich. Zur Transformation, eines Vektors in ein anderes Koordinatensystem, kann das Quaternion ebenfalls für genutzt werden:

$$\begin{pmatrix} 0 \\ \vec{x}^n \end{pmatrix} = \mathbf{q}_b^n \bullet \begin{pmatrix} 0 \\ \vec{x}^b \end{pmatrix} \bullet \mathbf{q}_n^b \tag{2.10}$$

Alternativ zu dieser Transformation, kann eine Transformation auch erfolgen, wenn zunächst aus dem Quaternion eine Richtungskosinusmatrix berechnet wird und im Anschluss der Vektor mithilfe dieser Richtungskosinusmatrix umgerechnet wird. Der Zusammenhang zwischen dem Quaternionen und der Richtungskosinusmatrix kann durch die folgende Matrix beschrieben werden:

$$\mathbf{C}_b^n = \begin{pmatrix} (a^2+b^2-c^2-d^2) & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & (a^2-b^2+c^2-d^2) & 2(cd-ab) \\ 2(bd-ac) & 2(cd-ab) & (a^2-b^2-c^2+d^2) \end{pmatrix} \tag{2.11}$$

Außerdem lassen sich Quaternionen aus Eulerwinkeln berechnen. Die folgende Gleichung soll den Zusammenhang zeigen:

$$a = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{2.12}$$

$$b = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{2.13}$$

$$c = \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \tag{2.14}$$

$$d = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2}. \tag{2.15}$$

Umgekehrt können Eulerwinkel auch aus Quaternionen berechnet werden, dafür muss lediglich mit Gl. 2.11 die zum Quaternionen gehörende Richtungskosinusmatrix gebildet und anschließend mittels Gl. 2.2 nur noch ausgewertet werden.
Quaternionen werden auch häufig mit Hilfe von imaginären Einheiten $\mathbf{i}, \mathbf{j}, \mathbf{k}$ dargestellt. Für diese gelten die folgenden Multiplikationsregeln:

$$\mathbf{i}\cdot\mathbf{i} = \mathbf{j}\cdot\mathbf{j} = \mathbf{k}\cdot\mathbf{k} = -1 \tag{2.16}$$

$$\mathbf{i}\cdot\mathbf{j} = -\mathbf{j}\cdot\mathbf{i} = \mathbf{k} \tag{2.17}$$

$$\mathbf{j}\cdot\mathbf{k} = -\mathbf{k}\cdot\mathbf{j} = \mathbf{i} \tag{2.18}$$

$$\mathbf{k}\cdot\mathbf{i} = -\mathbf{i}\cdot\mathbf{k} = \mathbf{j} \tag{2.19}$$

Mit der Quaternionendarstellung

$$\mathbf{q} = a + \mathbf{i}b + \mathbf{j}c + \mathbf{k}d \tag{2.20}$$

und den obigen Multiplikationsregeln können dann ebenfalls Quaternionenmultiplikationen und Vektortransformationen durchgeführt werden. Weitere Informationen zur Lagedarstellung mit dem Orientierungsvektor und Quaternionen sind u.a in [Titterton u. a. (2004), Wendel (2011) Jekeli (2001)] beschrieben.

Richtungskosinusmatrix

Die Richtungskosinusmatrix stellt eine weitere Methode dar einen Vektor von einem Koordinatensystem in ein anderes Koordinatensystem zu transformieren. In den vorherigen Kapiteln wurde die Richtungskosinusmatrix bereits kurz eingeführt. Die Richtungskosinusmatrix wird vor allem für kleine Drehungen verwendet, darunter fällt bspw. das Schätzen von Lagefehlern mittels Kalman-Filter. Es sollen somit lediglich kleine Drehungen betrachtet werden, sodass für die Drehwinkel die Näherungen

$$\sin\delta \approx \delta \tag{2.21}$$

$$\cos\delta \approx 1 \tag{2.22}$$

$$\delta \cdot \delta \approx 0 \tag{2.23}$$

begründet sind. Für das Quaternion aus Gl. 2.9 erhält man unter diesen Voraussetzungen die folgende Gleichung

$$\mathbf{q}_b^n = \begin{pmatrix} \cos(\sigma/2) \\ (\sigma_x/\sigma)\sin(\sigma/2) \\ (\sigma_y/\sigma)\sin(\sigma/2) \\ (\sigma_z/\sigma)\sin(\sigma/2) \end{pmatrix} \approx \begin{pmatrix} 1 \\ \sigma_x/2 \\ \sigma_y/2 \\ \sigma_z/2 \end{pmatrix} \tag{2.24}$$

Für kleine Eulerwinkel ϕ, θ, ψ erhält man anhand der Gl. 2.5 daher

$$\mathbf{C}_b^n = \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ \theta & \phi & 1 \end{pmatrix} \approx \mathbf{I} + \left[\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \times \right] \tag{2.25}$$

Des Weiteren lässt sich auch für die Richtungskosinusmatrix eine entsprechende Differentialgleichung angeben. Die Herleitungen sind in Titterton u. a. (2004) und Jekeli (2001) zu finden.

Die unterschiedlichen Darstellungen der Lage als Eulerwinkel, Orientierungsvektor und Quaternionen sowie die Richtungskosinusmatrizen sind bis auf die auftretenden Singularitäten bei den Eulerwinkeldarstellungen und Richtungskosinusmatrizen äquivalent und können ineinander umgerechnet werden.

2.2. MEMS-Sensoren

MEMS ist die Abkürzung für *Micro Electro Mechanicle Systems*, im deutschsprachigen Raum auch als *Mikro Elektro Mechanische Systeme* bekannt und seit Mitte der 90er Jahre nicht mehr wegzudenken. Bei MEMS-Sensoren handelt es sich um stark miniaturisierte Systeme, sog. Mikrosysteme, die Sensoren, Aktoren und Elektronik mittels mikrotechnologische Fertigungsverfahrens in Mikrogröße – von 1 mm bis 1 μm – zu einem System vereinen [Wild-Pfeiffer (2015)]. Sie können bestimmte physikalische oder chemische Messgrößen quantitativ erfassen und diese in elektrische Signale umwandeln (physikalisch z. B. Temperatur, Druck, Schallfeldgrößen, Helligkeit, Beschleunigung oder chemisch z. B. pH-Wert, Ionenstärke, elektrochemisches Potenzial). Die ersten MEMS-Sensoren kamen in den 1970er Jahren zum Einsatz und wurden bereits in Geräte und Maschinen eingebaut. Erst in den 90er Jahren konnte sich die Mikromechanik als Schlüsseltechnologie für die heutzutage üblichen Sensoren etablieren und die Funktionen konventioneller und hochwertiger Sensoren übernehmen [Marek (2007)]. Die gängigsten Sensoren sind bspw. Beschleunigungssensoren, Drehratensensoren, Temperatursensoren, Drucksensoren, Magnetfeldsensoren, Gassensoren etc. In [Wild-Pfeiffer und Schäfer (2011)] von Wild-Pfeifer und Schäfer werden MEMS-Sensoren bzw. Mikrsosysteme mit den folgenden Eigenschaften charakterisiert:

- mit klassischer Feinmechanik nicht realisierbar
- kompakte Bauweise
- geringes Gewicht
- hohe Ausfallsicherheit
- geringer Energieverbrauch
- kostengünstige Produktion
- parallele Fertigungstechnik vieler Mikrokomponenten auf einem Substrat (batch-fabricated).

Die steigenden Nachfragen nach MEMS-Sensoren führt aufgrund des Fertigungsverfahren und der Produktion in hohen Stückzahlen zu sinkenden Kosten. Dadurch sind MEMS-Sensoren in ihrer Anschaffung sehr kostengünstig, wodurch sich neue Einsatzmöglichkeiten ergeben.

MEMS-Sensoren werden daher auch gerne als Low-cost-Sensoren bezeichnet. Die Einsatzbereiche der Mikrosystemtechnik sind vielseitig und spielen in den verschiedensten Branchen eine Rolle. Nachfolgend werden einige MEMS Anwendungsfelder in ihren Teilgebieten vorgestellt [Wild-Pfeiffer und Schäfer (2011)]:

- IT-Branche: Notebook, Drucker
- Automobilindustrie: Airbagsysteme, Reifendrucksysteme, ESP-Systeme
- Konsumelektronik: Smartphones, Digitalkameras, Spielekonsolen
- Medizintechnik: Blutüberwachungssysteme, Herzschrittmacher, Implantate
- Industrie: Steuerung und Überwachung von Maschinen
- u. v. m.

Unter den MEMS-Sensoren erfahren insbesondere die Inertialsensoren eine hohe Nachfrage und haben sich bereits als Massenprodukt für zahlreiche Anwendungen etabliert. Unter einem Inertialsensor ist eine räumliche Kombination von Beschleunigungs- und Drehratensensor zu verstehen. Sie finden neben den bereits erwähnten Einsatzbereichen auch zunehmend in der Geodäsie und Navigation Anwendung. Dort werden sie vor allem zur optischen Bildstabilisierung, zur Neigungsbestimmung optischer Vermessungsinstrumente, zur Erfassung von Bauwerksschwingungen sowie für die Personennavigation verwendet.
Da sich die vorliegende Arbeit auf die inertiale Navigation fokussiert, werden entsprechend die dafür notwendigen MEMS-Sensoren in den nachfolgenden Kapiteln näher beschrieben.

2.2.1. Beschleunigungsmesser

Ein Beschleunigungsmesser ist ein Sensor, welcher die auf ihn wirkende Trägheitskraft F auf eine Masse m – eine sog. Probemasse –, ausgeübt durch eine Beschleunigung a, bzgl. eines inertialen Koordinatensystem misst.
Die nachfolgende Abbildung 2.2 soll das Funktionsprinzip eines Beschleunigungsmessers näher verdeutlichen. Sie zeigt dabei eine zwischen zwei Federn elastisch aufgehängte Masse in ihrer Nullposition (vgl. Abb. 2.2a oben). Durch eine Beschleunigung a wird diese Masse aus ihrer Nullposition gelenkt, dadurch ergibt sich eine Auslenkung Δl, welche sich proportional zur Beschleunigung verhält (vgl. Abb. 2.2a unten). Dieser Zusammenhang beschreibt damit das zweite Newton'sche Gesetz für die Trägheitskraft:

$$F = m \cdot a \tag{2.26}$$

bzw. für die Federkraft

$$F = k \cdot \Delta l \tag{2.27}$$

k ist eine Federkonstante. Durch das Gleichsetzen beider Kräfte bzgl. eines inertialen Koordinatensystem für die Beschleunigung a gilt:

$$a = \frac{k}{m} \cdot \Delta l \tag{2.28}$$

Weiterhin gilt für die tatsächlich erfasste Beschleunigung bzw. spezifische Kraft:

$$\overline{\mathrm{a}} = g - a \tag{2.29}$$

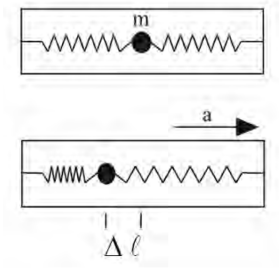
Des Weiteren lassen sich verschiedene in MEMS realisierte Arten von Beschleunigungsmessern unterscheiden:

- Piezoelektrisch: Messung der Kraft, die auf ein piezoelektrisches Material, einer Masse in Form einer Ladungsverteilung bzw. elektrischen Spannung wirkt. Zu den piezoelektrischen Materialien zählen Kristalle wie Quarz, Keramiken wie Blei-Zirkonat-Titanate oder Zinkoxid zur Dünnschichtherstellung.
- Kapazitiv: Messung der Änderung der Kapazität zwischen zwei festen Kondensatorplatten, durch Auslenkung der Probemasse unter den Einfluss der zu messenden Beschleunigung.
- Piezoresistiv: Messung der Widerstandsänderung, aufgrund einer Krafteinwirkung wie Druck, Feuchte oder Gase. Obwohl die Änderung bei jedem Materiel auftritt, ist dieser Effekt bei Halbleitermaterialien wie Silizium sehr viel ausgeprägter verglichen mit anderen Metallen.
- Vibration: Messung der Frequenzdifferenz (Schwebungsfrequenz zweier schwingender Quarze: Während der eine Quarz durch das Einwirken einer Beschleunigung von einer Probemasse gestaucht wird, der andere gedehnt.)

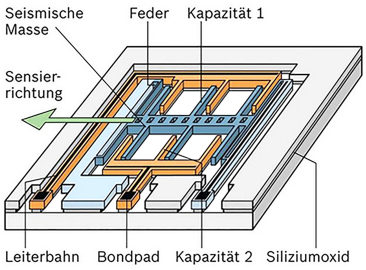
Für weitere Informationen über die Funktionweise wird folgende Literatur empfohlen [Titterton u. a. (2004), Hsu (2008)].

2.2.2. Drehratensensor

Ein Gyroskop oder auch Drehratensensor misst die Rotationsgeschwindigkeit einer Masse bzgl. eines inertialen Koordinatensystems. Generell lassen sich drei Typen von Drehratensensoren unterscheiden: der mechanische Kreisel, der optische Kreisel, basierend auf dem Sagnac-Effekt



(a) Funktion eines Beschleunigungssensors.



(b) MEMS-Beschleunigungssensors

Abbildung 2.2.: Grundprinzip eines Beschleunigungsmessers mit Feder-Masse-System (a) und die Realisierung auf Basis eines MEMS (b) [Bosch (2019)]

wie bspw. der Faserkreisel (FOG) oder Ringlaserkreisel (RLG), und der Vibrationskreisel, basierend auf dem Coriolis-Effekt. Im MEMS-Bereich wird die Technik der Vibrationskreisel realisiert. Sie zählt zu den kostengünstigsten, führt aber auch zu einer deutlich geringeren Genauigkeit. Eine Skizze eines MEMS-Kreisels ist in Abbildung 2.3 zu sehen. Sie soll als Basis dienen, um das Prinzip der Coriolis-Kraft näher zu verdeutlichen.
Durch elektrostatische Anregung werden zwei Probemassen kontrolliert gegenphasig in x-Richtung in Schwingung versetzt. Liegt nun eine Drehrate vor, wird eine Schwingung der Massen aufgrund der Coriolis-Kraft in z-Richtung verursacht. Die daraus resultierende Beschleunigung a_c und die damit verbundene Auslenkung können kapazitiv gemessen werden. Sie dienen als Maß für die Drehrate ω. Zusammen mit der Relativgeschwindigkeit v_{ref} und der Coriolis-Kraft F_c ergibt sich zusammen mit der Drehrate ω die Beschleunigung a_c:

$$a_c = \frac{F_c}{m} = 2v_{ref} \cdot \omega \tag{2.30}$$

Weiterführende Literatur zu MEMS-Kreiseln ist z.B. in [Wendel (2011)] zu finden.

2.2.3. Magnetometer

Wie bereits im Kapitel 2.1 beschrieben, liefert das Magnetometer Messungen des Erdmagnetfeldvektors.
In Deutschland beträgt die Feldstärke des Erdmagnetfeldes ungefähr 0,48 Gauss. In der inertialen Navigation kann das Magnetometer in Kombination mit dem Beschleunigungssensor zusätzliche Lageinformationen liefern und zur Stabilisierung des Gyroskops und folglich zur Verbesserung der Genauigkeit beitragen. Es gibt zwei verschiedene Arten zum Messen des

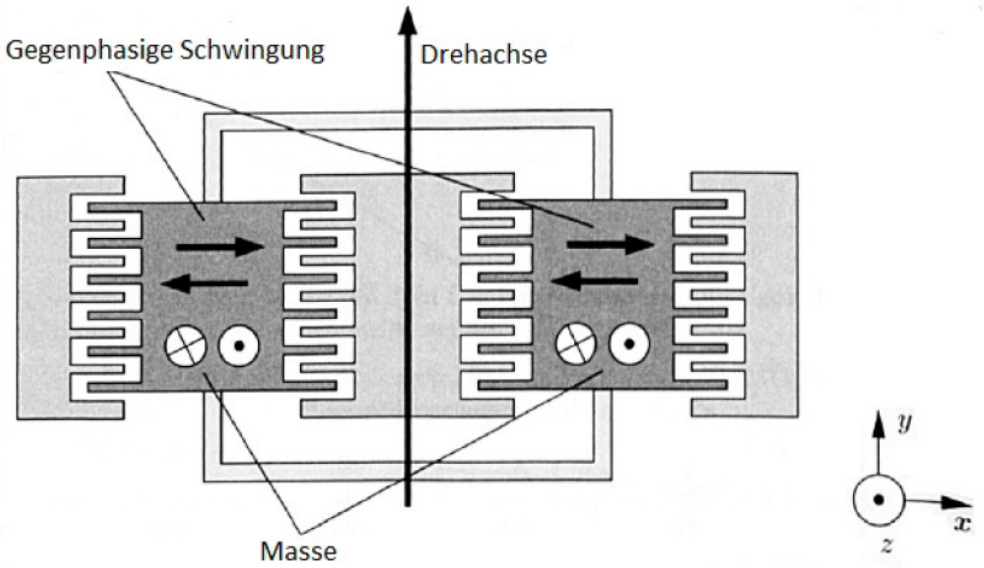


Abbildung 2.3.: Prinzipskizze eines MEMS-Kreisel (vgl. [Wendel (2011)], S.62)

Magnetfelds: zum einen mit dem *Hall-Effekt* und zum anderen mit dem von William Thomas entdeckten *anisotropen magnetoresistiven Effekt (AMR-Effekt)*. Die Mehrzahl der verbauten Magnetfeldsensoren in Smartphones basieren auf dem *Hall-Effekt*. Zudem geht aus dem Datenblatt [InvenSense (2019), Kasei (2019)] der in dieser Arbeit verwendeten Sensoren [Verknüpfung zur Tabelle in Kapitel mit den Sensoren] hervor, dass die verwendeten Magnetfeldsensoren ebenfalls auf dem *Hall-Effekt* basieren. Daher wird dieser nachfolgend näher erläutert.
Der Hall-Effekt beschreibt die Bewegung von Elektronen in einem stromdurchflossenen elektrischen Leiter. Die Basis des Messeffektes beruht auf der Lorenzkraft, die auf bewegte Ladungsträger wirkt. Die Abbildung 2.4 soll das Funktionsprinzip einer Hallplatte verdeutlichen. Bei der Präsenz eines Magnetfeldes B in z- Richtung und des Stromes I in x-Richtung, wird eine Hallspannung in y-Richtung U_H erzeugt.
Wird also ein stromdurchflossener Hall-Sensor einem zu ihm senkrecht verlaufenden Magnetfeld ausgesetzt, liefert er eine Ausgangsspannung. Setzt man einen konstanten Strom voraus, ist diese sogenannte Hall-Spannung proportional zu der magnetischen Feldstärke (= Hall-Effekt) (vgl. [Tille (2016)], S.237 ,S.260).

2.2.4. Fehlereigenschaften

Die Fehlereigenschaften inertialer MEMS Sensoren unterscheiden sich im Wesentlichen nicht von den klassischen Sensoren, weisen generell aber eine deutlich schlechtere Messgenauig-

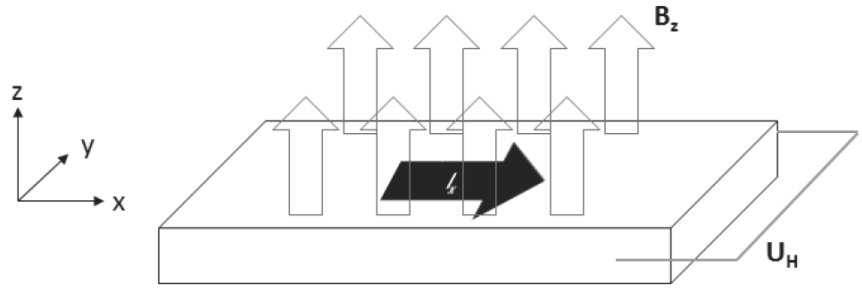


Abbildung 2.4.: Funktionsprinzip einer Hallplatte [Smits und Ballato (1994)].

keiten auf. Bei einer groben Genauigkeitsberechnung durch MEMS-Inertialsensoren können Hauptfehlereinflüsse wie *Bias* und *Rauschen* bereits nach kurzen Messzeiten von wenigen Sekunden große Abweichungen in der Orientierung, Geschwindigkeit und Position verursachen.

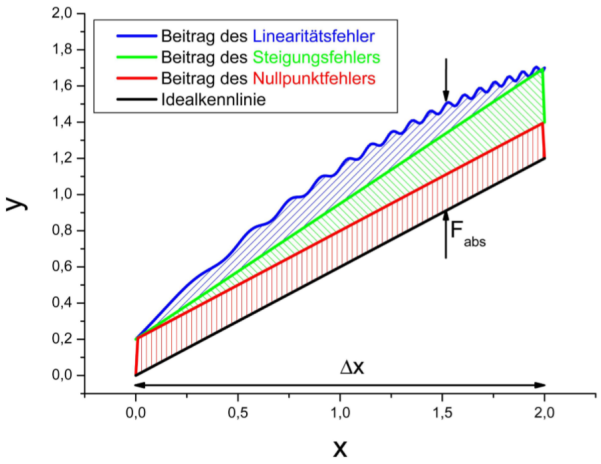


Abbildung 2.5.: Untergliederung des Fehlers in verschiedene Anteile [Kittel (2005)].

Die Skizze 2.5 soll beispielhaft verdeutlichen, wie sich der gesamte Fehler in verschiedene Anteile untergliedern lässt. Der absolute Fehler F_{abs} berechnet sich aus der Differenz von Istwert y_i und Sollwert y_s:

$$F_{abs} = y_i - y_s \tag{2.31}$$

Für die Ist-Kennlinie y_i, die Soll-Kennlinie y_s und den absoluten Fehler F_{abs} ergeben sich folgenden Gleichungen:

$$y_i(x) = y_{0i} + \frac{\Delta_{y_i}}{\Delta_x}(x - x_0) + F_{li}(x) \tag{2.32}$$

$$y_s(x) = y_{0s} + \frac{\Delta_{y_s}}{\Delta_x}(x - x_0) \tag{2.33}$$

$$F_{abs} = y_i(x) - y_s(x) = \underbrace{y_{0i} - y_{0s}}_{F_{nu}} + \underbrace{\frac{\Delta y_i - \Delta y_s}{\Delta_x}(x - x_0)(x)}_{F_{st}} + F_{li}(x) \tag{2.34}$$

Nachfolgenden werden die Fehlereinflüsse genauer beschrieben und weitere vorgestellt, mit denen man unter anderen bei inertialen Sensoren konfrontiert wird [Lawrence (2001)]:

- Bias (Nullpunktfehler)
- Steigungsfehler
- Rauschen
- Nichtlinearität
- Quantisierungsfehler

Beim *Bias* handelt sich um einen Drift der Nullpunktverschiebung, die durch Temperaturschwankungen hervorgerufen wird. Unter einer Nullpunktverschiebung ist eine Ausgabe eines Wertes ungleich Null zu verstehen, auch wenn keine reale Beschleunigung bzw. Winkelgeschwindigkeit vorliegt. Dementsprechend müssen die Messwerte um diese Nullpunktverschiebung korrigiert werden, um die reale Beschleunigung bzw. Winkelgeschwindigkeit zu erhalten.
Der *Steigungsfehler* oder *Skalenfaktorfehler* beschreibt das Verhältnis vom Eingangssignal zum Ausgangssignal über den Wertebereich. Sensoren besitzen oftmals eine direkte Proportion vom Eingangssignal zum Ausgangssignal, der Skalierungsfaktor entspricht damit einer Geraden.
Das *Rauschen* ist eine weitere Störgröße (z. B. verursacht durch Erschütterungen), sie wird auch als *Noise* bezeichnet. Sie ist nicht deterministisch und kann daher nicht durch eine Kalibrierung kompensiert bzw. entfernt werden. Allerdings ist es möglich, mittels Signalverarbeitung das Rauschen bspw. unter Verwendung der Allan-Varianz-Berechnung zu ermitteln und mit einer anschließenden Filterung zu verringern.
Die *Nichtlinearität* oder *Linearitätsabweichung* ist, wie in Abb. 2.5 dargestellt eine Abweichung der Ist-Kennlinie von einer gewünschten linearen Soll-Kennlinie.
Der *Quantisierungsfehler* beschreibt den Fehler, welcher bei der Umsetzung eines analogen in ein digitales Signal entsteht. Weitere Fehler, die bei inertialen Sensoren auftreten sind in [Lawrence (2001)] angegeben.

Beschleunigungsmesser

Messungen von Beschleunigungsmessern sind in der Regel immer mit Fehlern behaftet. Aus diesem Grund soll das folgende generische Fehlermodell eines Beschleunigungsmessers diese Fehler nochmals verdeutlichen (vgl. [Wendel (2011)], S.72):

$$\tilde{\vec{f}}_{ib}^{b} = M_{Acc} \cdot \vec{f}_{ib}^{b} + \vec{b}_a + \vec{n}_a \tag{2.35}$$

Die tatsächliche gemessene Beschleunigung entspricht $\tilde{\vec{f}}_{ib}^{b} = M_{Acc}$ und die reale Beschleunigung wird mit $\vec{f}_{ib}^{b}$ gekennzeichnet. Um die Beschleunigung messen zu können, sollten idealerweise die Achsen der drei Sensoren senkrecht zueinander stehen. Da das in der Realität nicht der Fall ist, kann eine Korrektur mit der Misalignment-Matrix M_{Acc} durchgeführt werden. In M_{Acc} befinden sich auf der Hauptdiagonalen die Skalenfaktoren für die Maßstabskorrektur und auf den Nebendiagonalen die Parameter für die Ausrichtung. Des Weiteren entspricht $\vec{b}_a$ dem Bias, d. h. den Nullpunktfehlern der Sensoren, der Vektor $\vec{n}_a$ beschreibt das sensorinherente Rauschen, welches als weiß, normalverteilt und mittelwertfrei angenommen werden kann. Durch einen Referenzwert lassen sich die Orthogonalität, der Maßstab und der Bias für die Erdbeschleunigung kalibrieren.

Gyroskop

Das Prinzip des inertialen Fehlermodells gilt ebenfalls auch bei den Drehratensensoren. Da beide Sensoren mit denselben Fehlereinflüssen behaftet sind, kann hier ebenfalls das folgende vereinfachte Fehlermodell der Form

$$\tilde{\vec{\omega}}_{ib}^{b} = M_{Gyro} \cdot \vec{\omega}_{ib}^{b} + \vec{b}_\omega + \vec{n}_\omega \tag{2.36}$$

gewählt werden (vgl. [Wendel (2011)], S.68). Die gemessene Drehrate entspricht $\tilde{\vec{\omega}}_{ib}^{b}$ und die reale Drehrate kann entsprechend mit $\vec{\omega}_{ib}^{b}$ bezeichnet werden. Im idealen Fall sollten ebenfalls die Achsen der drei Sensoren orthogonal zueinander ausgerichtet sein. Allerdings entspricht das selten der Realität, aus diesem Grund kann die vorliegende Missweisung der drei Achsen über die Nebendiagonalelemente der Misalignment-Matrix M_{Gyro} beschrieben und korrigiert werden. Das Misalignment wird nach dem Zusammenbau vermessen und im Anschluss innerhalb der IMU mathematisch kompensiert. Der Term $\vec{b}_\omega$ gibt wiederum die Biase des Sensors an. Mit $\vec{n}_\omega$ wird das sensorinherente Rauschen bezeichnet.

Magnetometer

Magnetometer sind mit denselben Fehlern behaftet wie Beschleunigungsmesser und Drehratensensor. Allerdings übersteigen dabei die magnetischen Abweichungen, welche den wahren Messwert überlagern, alle anderen Fehlereinflüsse. Zu den häufigsten magnetischen Einflüssen, die diese Abweichungen auslösen, zählen die *Hard-* und *Soft-Iron*-Effekte. Von einem *Hard-Iron*-Effekt ist die Rede, wenn statisch erzeugte Magnetfelder direkt und permanent in der unmittelbaren Umgebung (lokal) auf den Sensor wirken. Sie können beispielsweise durch die verbauten Komponenten auf der Platine verursacht werden. Dementsprechend wird von einem *Soft-Iron*-Effekt gesprochen, wenn das umgebende Magnetfeld (global) eine Beeinflussung des Sensors hervorruft. Durch eine Kalibrierung kann der Magnetfeldsensor auch in Umgebungen mit lokalen und globalen Magnetfeldern verwendet werden. Eine theoretische Einführung und praktische Anleitung für eine solche Kalibrierung kann in der Literatur [Ozyagcila (2015)] nachgeschlagen und durchgeführt werden.

2.3. Inertiale Navigation mittels Deep Learning

In dieser Arbeit sollen die Convolutional Neural Networks mit den Recurrent Neural Networks (in Kombination) als Ensemble verwendet werden. Aus diesem Grund beschreibt das vorliegende Kapitel die beiden DL-Techniken näher und gibt eine grundlegende Einführung.

2.3.1. Convolutional Neural Networks

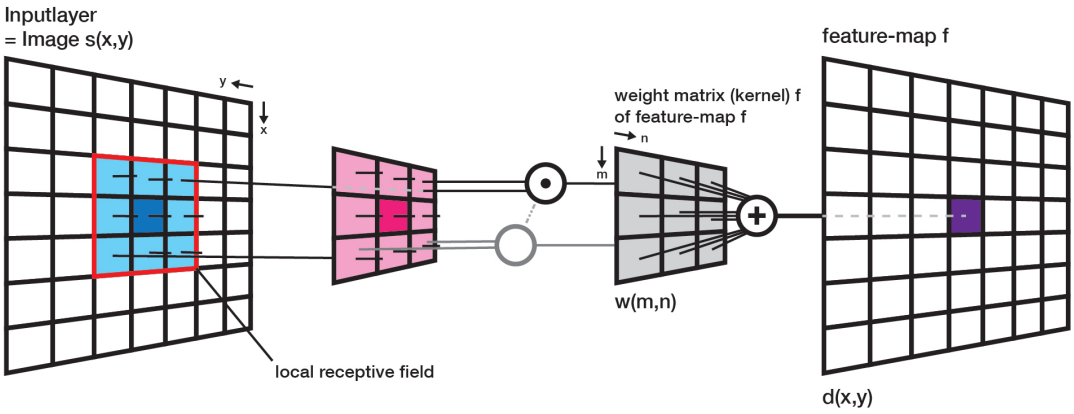


Abbildung 2.6.: Eine Feature-Map f wird durch Faltung des Input-Layers mit der Gewichtsmatrix f berechnet

Bei den Convolutional Neuronal Networks (CNN) (dt. Faltungsnetzwerke) handelt es sich um eine Sonderform des Multilayer Perceptron mit einer speziellen Architektur. Die CNNs entstanden aus Untersuchungen des visuellen Cortex des Gehirns und inspirierten das im Jahr 1980 vorgestellte Neocognitron, das nach und nach, zudem weiterentwickelt worden ist, was man heute als Convolutional Neural Networks definiert. Yann LeCun [LeCun u. a. (1989)] präsentierte die Convolutional Neuronal Networks zum ersten Mal 1989. Erst dank anwachsender Rechenkapazitäten, der Menge an verfügbaren Trainingsdaten sowie weiterer zusätzlicher Techniken zur weiteren Steigerung der Erkennungsrate, konnten die ersten Erfolge verzeichnet werden. Ein wichtiger Meilenstein dabei war ein Artikel [LeCun u. a. (1998)] aus dem Jahr 1998, indem Yann LeCun die berühmte *LeNet-5-Architektur* vorstellte. Seit 2012 hat sich die DL-Architektur vollständig etabliert und gehört seitdem zu den „State-of-the-Art"-Methoden in Anwendungsbereichen wie Bild- und Spracherkennung, Objektklassifizierung sowie zur automatischen Videoanalyse und -klassifikation [Krizhevsky u. a. (2012)]. Gegenüber anderen Bildklassifizierungsverfahren unterscheidet sie sich besonders durch ihre Unempfindlichkeit gegen jegliche Objekttransformation, wie Skalierung, Rotation und Translation [Lecun u. a. (1998)]. Ermöglicht wird das durch die Faltungskerne, die während des Trainings angelernt werden. Folglich führt das auch zu einer Reduktion der Trainingsmenge und Trainingszeit. Die Faltungskerne dienen dazu, bestimmte Merkmale von zu klassifizierenden Objekten zu erkennen, z. B. Linien, Bögen, Rechtecke oder Kreise.
Soll eine Objektklassifizierung durchgeführt werden, bekommt das Faltungsnetz als Eingabewert bspw. ein zweidimensionales Bild. Die Neuronen in der ersten Faltungsschicht sind dabei nicht mit jedem Pixel im Eingabebild verbunden, sondern nur mit den Pixeln in einem sog. lokalen Wahrnehmungsfeld (*eng.* local receptive field). Daraus folgt, dass jedes Neuron im zweiten CNN Layer ausschließlich mit Neuronen innerhalb eines kleinen Bereiches mit der ersten Schicht verbunden ist. Der Architektur ist es dadurch möglich, in der ersten verborgenen Schicht sich auf kleinteilige Merkmale zu fokussieren, welche sich wiederum in der nächsten verborgenen Schicht zu übergeordneten Merkmalen zusammensetzen. Dieser Vorgang setzt sich von Schicht zu Schicht fort. Dadurch entwickelt sich nach und nach eine Merkmalshierarchie. In jedem Layer findet eine weitere Filterung statt, mit steigender Anzahl von Schichten steigt auch die Komplexität im Layer. Die Gewichte eines Neurons lassen sich als kleines Bild in der Größe des *local receptive field* darstellen (siehe Abbildung 2.6). Die Gewichte werden auch Filter, Faltungsmaske, Faltungsmatrix oder Kernel genannt, welche in Abbildung 2.6 als w gekennzeichnet sind. Die Abbildung 2.6 soll dabei die Anwendung des Filters demonstrieren. Sie zeigt, wie ein Filter auf ein Wahrnehmungsfeld oder einen Bildbereich angewandt und durch die Berechnung von korrespondierenden Punkten im Anschluss ein neuer Pixel erzeugt

wird. So bildet eine Schicht Neuronen mit dem gleichen Filter eine *Feature-Map* (dt. Abbildung von Merkmalen). Jeder Filter dient dazu, Merkmale von Objekten aus den entsprechenden Wahrnehmungsfeldern zu erkennen, z. B. Linien, Bögen, Rechtecke oder Kanten. Die Merkmale können in weiteren Schritten auch kombiniert und verfeinert werden, wodurch aus dieser Menge von Feature-Maps wieder weitere Feature-Maps gebildet werden. Die Faltung lässt sich dabei wie folgt beschreiben:

$$d(x,y) = b_f + \sum_{m=-b}^{a} \sum_{n=-b}^{b} w(m,n) \cdot s(x-m, y-n), \tag{2.37}$$

mit b_f als der Bias-Term der Feature-Map f. Während des Trainings werden die Filter trainiert und entsprechend immer komplexer und lernen auch neue komplexe Features bzw. Muster zu identifizieren oder kombinieren. Jeder Faltungskern dient dazu, ein bestimmtes Merkmal von den zu klassifizierenden Objekten innerhalb eines Bildes zu erkennen.
Um in einem CNN die Komplexität, die Rechenlast und die Anzahl der trainierbaren Parameter weiter zu reduzieren (und nebenbei auch das Risiko für Overfitting zu senken), ist der Einsatz eines sog. Pooling Layer möglich. Typischerweise folgt ein Pooling Layer nach einem oder mehreren Convolutional Layern. Der Pooling Layer dient dazu, überflüssige Informationen zu verwerfen und parallel damit die Bildauflösung zu reduzieren. Eines der bekanntesten Faltungsnetzarchitekturen ist das „LeNet-5“ in Abbildung 2.7. Das Netz besteht aus mehreren Merkmalsextraktions-Phasen (Feature-Extraction-Phases). Dazu zählen C1 mit S2 und C3 mit S4. Mit „Subsampling“ S ist hier lediglich ein Pooling Layer gemeint. Zum Abschluss folgt ein Klassifikator, üblicherweise ein klassisches MLP, welches sich hier aus drei Schichten zusammensetzt (C5, C6, OUTPUT).

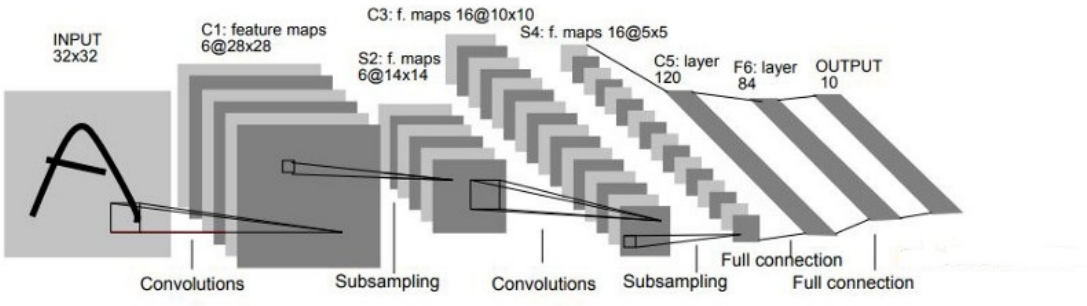


Abbildung 2.7.: Die Architektur des LeNet-5, ein Convolutional Neural Networks zur Klassifizierung von handschriftlichen Ziffern [Lecun u. a. (1998)]

2.3.2. Recurrent Neural Networks

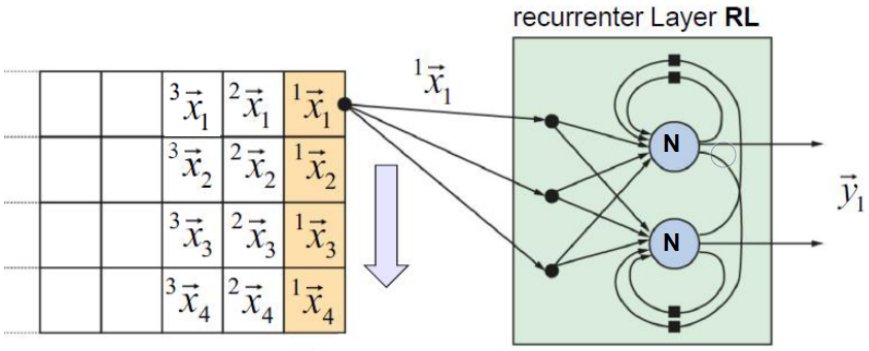


Abbildung 2.8.: Ein Recurrent Layer (RL) mit einem TBPTT-Zeitenster mit 4 Inputvektoren (Timesteps) [Meisel (2017)]

Neuronale Netze (NN) benötigen als Eingabesequenz eine im Vorhinein festgelegte Länge. Damit eigenen sie sich nicht unbedingt zum Verarbeiten von Sequenzen mit beliebiger Länge von Vektoren. Hier kommen die Recurrent Neural Networks (dt. rekurrenten neuronalen Netze) ins Spiel. Sie eignen sich nicht nur zum Verarbeiten von einzelnen Eingaben, sondern auch von einer Abfolge von Eingaben, deren Länge vorher nicht festgelegt worden ist. Durch ihre rückwärts gerichteten Verbindungen sind sie in der Lage sich selbst zu beeinflussen, wodurch sie ein leistungsfähiges Werkzeug zur Modellierung von Sequenzen darstellen. Die Abbildung 2.8 soll beispielhaft den Aufbau einer Schicht rekurrenter Neuronen (RL-Recurrent Layer) verdeutlichen. Sie soll zeigen, dass ein RNN nicht nur aus einem einzelnen rekurrenten Neuron besteht, sondern auch eine ganze Schicht mit rekurrenten Neuronen erstellt werden kann. Die nachfolgende Abbildung 2.9 geht noch einen Schritt weiter. Hier wurde das RNN entlang der Zeitachse dargestellt, was auch häufig als das *„Netz entlang der Zeitachse aufrollen“* bezeichnet wird. Jeder RL kann dementsprechend auch als Zeitschritt (Timestep) interpretiert werden.Zu jedem *Ausführungszeitpunkt* t erhält der RL die Eingabe $\boldsymbol{x}_t$ sowie seine eigene Ausgabe aus dem vorhergegangen Zeitschritt $\boldsymbol{y}_{t-1}$. Um ein RNN zu trainieren, wird auf das *backpropagation through time* (BPTT) zurückgegriffen. Das Besondere bei diesem Verfahren liegt darin, die RNNs entlang der Zeitachse aufzurollen und im Anschluss das gewöhnliche Backpropagation-Verfahren anzuwenden. Ein bereits bekanntes Problem ist, dass es keine Möglichkeit gibt Informationen *unmittelbar* weiterzugeben, denn alle Parameter spielen bei jedem Zeitschritt die gleiche Rolle, d. h. in jedem Berechnungsschritt werden die gleichen Parameter verwendet. Daraus folgt, dass der Einfluss früherer Ereignisse auf spätere Berechnungsschritte verloren geht. Um dem Problem entgegenzuwirken, führt man sog. *gates* ein, die

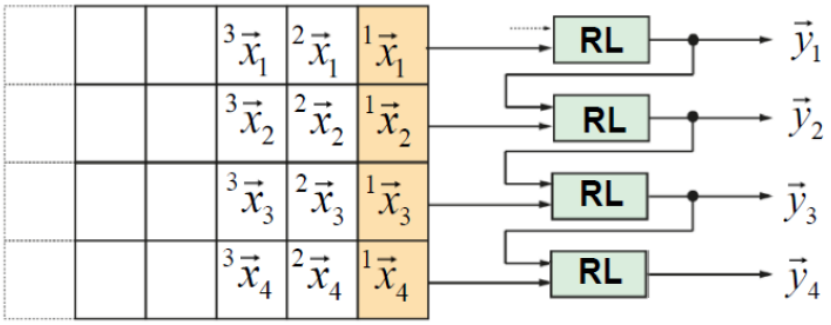


Abbildung 2.9.: Zeitlich aufgerollter Recurrent Layer (RL) in 4 aufeinanderfolgenden Zeitschritten [Meisel (2017)].

bestimmte Informationen unmittelbar (d. h. unverändert) über beliebig viele Zwischenschritte weiterleiten können.

LSTM - *Long-Short-Term Memory*

Die *Long-Short-Term-Memory*-Zelle wurde im Jahr 1997 von Sepp Hochreiter und Jürgen Schmidhuber [Hochreiter und Schmidhuber (1997)] vorgeschlagen und im Laufe der Jahre von mehreren Wissenschaftlern wie Alex Graves, Hasim Sak [Sak u. a. (2014)](2014:338-342) und vielen anderen weiterentwickelt. Man könnte das Ganze auch automatentheoretisch auffassen und sich das RNN als Automaten vorstellen: Es gibt eine Folge von Eingaben $\vec{x}_1, ..., \vec{x}_n$ und nach jeder Eingabe wird ein neuer Zustand $\vec{s}_i$ erreicht. Daraus resultiert eine Zustandsfolge $\vec{s}_0 \overset{\vec{x}_1}{\Rightarrow} \vec{s}_1 \overset{\vec{x}_2}{\Rightarrow} ... \overset{\vec{x}_n}{\Rightarrow} \vec{s}_n$. Das Problem bei dieser Folge ist, dass an jedem Zeitpunkt t_i das Netz keinen Zugriff auf den unmittelbaren vorhergehenden Zustand, sowie die neue Eingabe hat. Damit bleiben Muster weiterhin unsichtbar, insbesondere voneinander entfernte weitreichende Abhängigkeiten zwischen Eingaben.

Die Architektur einer LSTM-Zelle ist in Abbildung 2.10 dargestellt. Da es sich bei Sensordaten, wie sie auch in dieser Arbeit verwendet werden, um Zeitreihen mit weitreichenden temporalen Abhängigkeiten handelt, wo der Einfluss früherer Ereignisse auf aktuelle Berechnungen mit berücksichtigt werden soll, wird das RNN zusätzlich um eine Langzeitgedächtniszelle oder auch LSTM-Zelle erweitert. Die Idee der Architektur ist, den Zustandsvektor $\vec{s}$ in zwei einzelne Vektoren aufzuteilen: $\boldsymbol{h}_{(t)}$ und $\boldsymbol{c}_{(t)}$ (»c«für cell und »h«häufig für hidden). Dabei kann $\boldsymbol{h}_{(t)}$ man sich als Kurzzeitgedächtnis und $\boldsymbol{c}_{(t)}$ als Langzeitgedächtnis vorstellen. Der Grundgedanke ist, dass das Netz lernen kann, was im Langzeitgedächtnis gespeichert werden soll, was vergessen werden kann und wie das Gedächtnis die entsprechenden Daten interpretieren könnte.

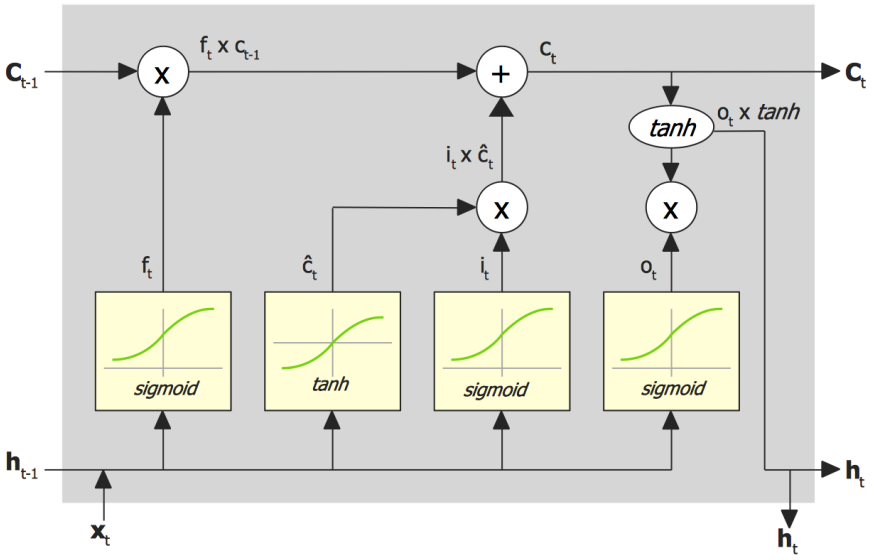


Abbildung 2.10.: Einfache LSTM-Zelle [Géron (2017)].

Während der Langzeitzustand $\boldsymbol{c}_{(t-1)}$ Informationen *fast* unverändert weiterreichen kann, passiert er die entsprechenden *gates*, die die Weitergabe kontrollieren und optimieren. Der Zustand $\boldsymbol{h}_{(t-1)}$ wird an vier einzelne, vollständig verbundene Schichten übergeben, komplett bearbeitet und durchläuft dabei verschiedene Rechenschritte. Näheres dazu kann in der Arbeit *„Long-short-term memory recurrent neural network architectures for large scale acoustic modeling“* [Sak u. a. (2014)](2014:338-342) nachgelesen werden. Die *gates* aus der Abbildung 2.10 können wie folgt beschrieben werden:

- ***Input Gate*:**
  Das Input Gate kontrolliert, welche Teile von $\tilde{\boldsymbol{c}}_t$ in das Langzeitgedächtnis einfließen (wird von $\boldsymbol{i}_{(t)}$ gesteuert).
- ***Forget Gate*:**
  Das Forget Gate kontrolliert, welche Teile vom Langzeitgedächtnis behalten werden sollen (wird von $\boldsymbol{f}_{(t)}$ gesteuert).
- ***Output Gate*:**
  Das Output Gate kontrolliert, welche Teile des Langzeitgedächtnisses ausgelesen und ausgegeben werden sollen (ausgegeben an $\boldsymbol{h}_{(t)}$ und $\boldsymbol{y}_{(t)}$; von $\boldsymbol{o}_{(t)}$ gesteuert); gleichzeitig handelt es sich dabei auch um den Output des Zeitschrittes.

Die nachfolgenden Gleichungen 2.38-2.43 sollen zusammengefasst noch einmal nachvollziehbar beschreiben, wie sich der Zustand des Langzeitgedächtnisses, des Kurzzeitgedächtnisses und die Ausgabe eines Zeitschrittes für die einzelnen Datenpunkte berechnen lassen.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.38}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.39}$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.40}$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.41}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.42}$$
$$h_t = o_t \cdot \tanh(C_t) \tag{2.43}$$

- $\boldsymbol{W_f, W_i, W_{\tilde{c}}, W_o}$ sind Gewichtsmatrizen des Eingabevektors $\boldsymbol{x_t}$ zu den vier Schichten, sowie Gewichtsmatrizen der Verbindungen des Kurzzeitgedächtnisses $\boldsymbol{h_{t-1}}$ zu den vier Schichten.
- $\boldsymbol{b_f, b_i, b_c, b_o}$ entsprechen den Bias-Termen der vier Schichten.

Der *cell state* und *hidden state* eines LSTM-Netzes wird auch als interner Zustand oder interne Zustände (engl. *internal state*) bezeichnet. Zur Steuerung der internen Zustände wird zwischen einem zustandslosen (engl. *stateless*) und einem zustandsbehafteten (engl. *stateful*) LSTM unterschieden. Bei einem stateful LSTM wird der letzte, für den aktuellen Batch, berechnete interne Zustand des LSTMs als Initialisierungszustand (engl. *initial-state*) für die Trainingsbeispiele des nachfolgenden Batches verwendet. Die Batch-Size beschreibt die Menge an Trainingsbeispielen, mit der das Netz trainiert wird, bevor die Gewichte aktualisiert werden. Bei einem stateless LSTM erfährt der interne Zustand stattdessen einen Reset nach jeder Batch-Sequenz, der folgende Batch erhält als initial-state einen sog. zero-state.

Es wurde gezeigt, warum LSTM-Architekturen so ausgesprochen erfolgreich beim Erkennen von weitreichenden und temporalen Abhängigkeiten sind, vor allem solche, die das Modellieren von Sequenzen erfordern. Sie liefern derzeit „State-of-the-Art“ Ergebnisse für eine große Anzahl von Aufgabengebieten.

3. Vergleichbare Arbeiten und Techniken

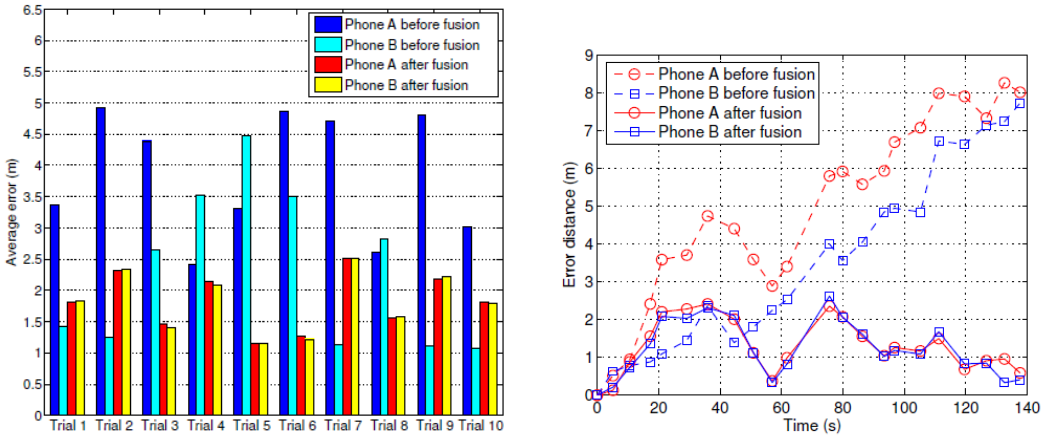
Die Entwicklung von INS hat sich in den letzten Jahren von kardanisch gelagerten Systemen auf Strapdown-Systeme verlagert. Die Gründe liegen in den enormen Weiterentwicklungen der Gyros wie RLG, FOGs und vibrierende Gyros. Dennoch sind Strapdown-Systeme gegenüber MEMS-Inertialsensoren immer noch vergleichsweise groß, schwer und in der Anschaffung relativ teuer. Wegen ihrer Eigenschaften werden sie daher in den professionellen Bereichen wie z. B. beim Militär in Lenkflugkörpern, in Flugzeugen, Schiffen und U-Booten eingesetzt. Dieses Kapitel stützt sich daher auf verwandte Arbeiten, deren Grundlage für INS ebenfalls Low-cost-Sensoren gewesen sind und die nicht auf eine Kombination mit satellitengestützten Verfahren zurückgegriffen haben. Um in verschiedenen Umgebungen eine zuverlässige und möglichst exakte Navigation zu garantieren, bietet die Literatur ein breites Spektrum an unterschiedlichen Ansätzen und Technologien an. Für eine relative Ortung kommen häufig Funkwellen wie Bluetooth [Li u. a. (2017), Bahillo u. a. (2015)] und WiFi [Zhang u. a. (2017)] zum Einsatz. Für eine relative Positionierung wird häufig auf eine Kombination zwischen Inertialsensoren [Park und Suh (2010b)] und Magnetometern [Wang u. a. (2016)] zzgl. Korrektursystem gesetzt. Außerdem sehr beliebt und weit verbreitet sind die unterschiedlichen Varianten der visuell gestützten Systeme zum Erkennen von visuellen Features in der Umgebung mittels Kamera [Ruotsalainen u. a. (2010), Yan u. a. (2018) Fu u. a. (2018)]. Um eine weitere Steigerung der Genauigkeiten zu erhalten, werden häufig auch Schallsensoren und Barometer zur Unterstützung herangezogen. Hybrid basierte Ansätze (hybride oder fusionierte Ansätze) erhält man durch eine Vereinigung bzw. Verknüpfung mehrerer Techniken miteinander [Davidson und Piché (2017)].
Hybride Systeme sind ein bevorzugtes Mittel, da die spezifischen Vorteile eines Systems genutzt werden können, um die Nachteile des anderen Systems zu kompensieren. Zusätzlich kann so auch eine gewisse Redundanz geschaffen werden. Da kostengünstige Sensoren mittlerweile dafür bekannt sind mit der Zeit fehlerbehaftete Messwerte zu liefern, fokussierten sich die Mehrzahl der Ansätze auf Kombinationen verschiedener Techniken und Methoden. Um die Fehler dabei möglichst klein zuhalten, wird auf Reduzierung und Kompensation gesetzt. Die dafür am häufigsten verwendeten und geeignetsten Algorithmen sind die sog. *Pedestrian-Dead-Reckoning-(PDR)*-Algorithmen. Der Stand der Technik im Bereich der relativen Positi-

onsbestimmung insbesondere für Fußgänger hat sich in den letzten Jahren weitgehend auf Lösungen für die Montage des Sensors am Fuß konzentriert. Alternative Bewegungsformen, wo stattdessen die Person das Smartphone in einer Handtasche, am Hosengürtel, auf einem Trolley oder gar in der Hosentasche mit sich trägt, stellen die PDR-Verfahren immer noch vor große Herausforderungen.

3.1. PDR-basierte Low-cost-Ansätze

In der Arbeit von [Jin u. a. (2011)] wurde bspw. ein traditioneller PDR-Algorithmus verwendet, jedoch mit dem Unterschied, statt eine Sensoreinheit gleich zwei identische Sensoreinheiten zu verwenden. Die Intention der Arbeit bestand darin, durch eine kooperative Sensordatenfusion eine zuverlässige Bestimmung der eigenen Position zu erreichen. Bei dem vorgeschlagenen Fusionsansatz handelt es sich um einen *Maximum-a-posteriori*-Algorithmus. Diese Methode ist auch als spezieller Bayes-Schätzer bekannt, ein Schätzverfahren aus der mathematischen Statistik. Bei den Sensoren handelt es sich um drei Smartphones – ein HTC Magic (*Worker 1*), HTC Hero (*Manager*) und ein Google NexusOne (*Worker 2*). Das HTC Hero wurde als Manager bezeichnet und während der Durchführung der Experimente in der Hand gehalten. Die anderen beiden Smartphones wurden als *Worker* bezeichnet und abwechselnd in der Hosentasche getragen. Die Verknüpfung der Messdaten fand dabei immer nur zwischen einem *Worker* und dem *Manager* statt. Bei dem Versuch wurden Beschleunigungssensoren und Magnetometer verwendet. Dabei ist es nicht erforderlich, fortlaufend zwei Smartphones mit sich zu tragen. Ein Tablet, Notebook, PDA oder entsprechende eingebettete Sensoren am oder im Schlüsselbund sind ebenfalls dafür geeignet. Bevor die Sensorfusion angewendet werden kann, müssen die Sensoreinheiten die Phasen des PDR-Algorithmus durchlaufen haben, erst im Anschluss kommt es zu einer Sensorfusion der berechneten Positionen. Bei der Schrittdetektion wurde die vertikale Achse des Beschleunigungssensors dazu verwendet, um mittels Peak-Detektion die regelmäßige Abfolge der Bewegungen zu registrieren. Der Autor wies in seiner Arbeit besonders daraufhin, dass das Rauschen des Smartphones unabsichtlich detektierte Peaks verursachte, weswegen ein zusätzlicher Threshold definiert worden ist. Ein neuer Schritt wird somit erst erkannt, wenn die Werte zwischen einem definierten lokalen Minimum und einem lokalen Maximum vorliegen und einen Abstand von mindestens 150 ms aufweisen. Die zweite Phase des PDR-Verfahrens beinhaltet die Schrittlängenberechnung. Zur Bestimmung der Schrittlänge ρ_k wurde die folgende Gleichung verwendet:

$$\rho_k = K \cdot \sqrt[4]{a_k^{v-\max} - a_k^{v-\min)}}, \tag{3.1}$$



(a) Durchschnittlicher Fehler aller 10 Experimente

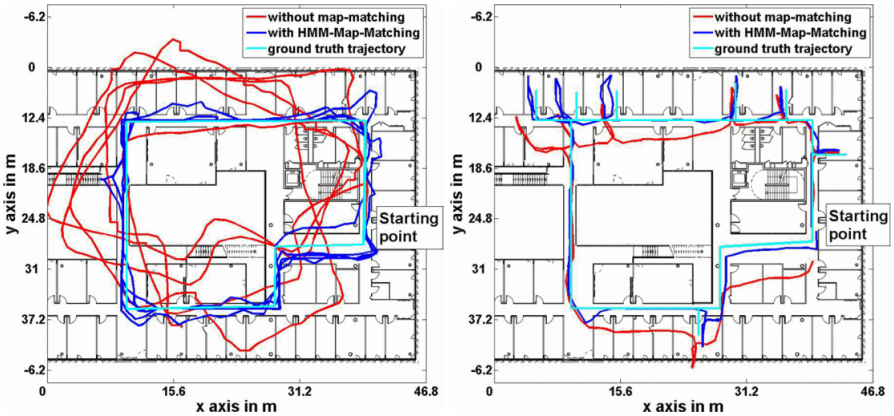
(b) Experiment 6 im Detail

Abbildung 3.1.: Die Resultate der 10 durchgeführten Experimente *maximum-a-posteriori* [Jin u. a. (2011)].

$a_k^{v-\max}$ und $a_k^{v-\min}$ entspricht der Vertikalbeschleunigung des Beschleunigungssensors, während jedem k^{th}-Schritt. K ist eine personenspezifische Kalibrierkonstante [Inderst u. a. (2015), Abe u. a. (2016)]. Zum Schluss wurde die Orientierung mithilfe des Magnetometers bestimmt. Mit den Smartphones wurden insgesamt zehn unterschiedliche Datensätze gesammelt und ausgewertet (siehe Abb. 3.1a). Dabei konnte lediglich bei einem Datensatz (Experiment 6 in Abb. 3.1a und in Abb. 3.1b im Detail) eine Reduzierung des Fehlers von bis zu 73.7%, im Vergleich zu herkömmlichen PDR-Verfahren erreicht werden. Verwandte Arbeiten, die denselben Ansatz verfolgen, sind [Lei Fang u. a. (2005)] und [Beauregard (2006)]. In der ersten wurde der Sensor am Gürtel der Person montiert und in [Beauregard (2006)] wurde der Sensor während des Reitens auf dem Helm getragen (bspw. wäre diese Montage für Feuerwehrleute und Polizeibeamte geeignet). Beide Ergebnisse wurden als zufriedenstellend beschrieben. Vor allem in [Beauregard (2006)] konnten die Schwächen der Montage am Helm schnell identifiziert werden, denn sobald die Person nur die Blickrichtung änderte, kam es bereits zu Abweichungen in der Trajektoriebestimmung. In einer anderen Arbeit [Lamy-Perbal u. a. (2015)] wurde das Ziel verfolgt, eine möglichst genaue 2D-Trajektorie zu erzeugen, um eine Lokalisierung in einer Umgebung unabhängig von der Infrastruktur zu erlauben. Ein Hidden Markov Model (HMM) sollte dabei die Wahrscheinlichkeit der nächsten möglichen Position anhand der bereits durchlaufenen Positionen auf einer Karte angeben. Sollte es aufgrund fehlerhafter Sensordaten zu einer falschen oder unklaren Positionsberechnung kommen, wodurch die Position „verloren “ gehen kann, registriert das HMM diese Abweichung und korrigiert die aktuelle Position entspre-

chend. In der Arbeit wird das Verfahren als *Map-Matching-(dt. Kartenanpassung)*-Algorithmus bezeichnet. Die großen Vorteile dieses Verfahrens sind zum einen die Unabhängigkeit von der Infrastruktur und zum anderen die Kostenersparnis, da auf eine zusätzliche Hardware oder Aufrüstung der Infrastruktur verzichtet werden kann. Der große Nachteil ist der Aufwand, den das Verfahren mit sich bringt. Ein Teil dieser Lösung liegt darin, dass der Raum vorher als Gitter (die Zustände) modelliert werden muss, außerdem muss den Gittern eine Wahrscheinlichkeitsdichte zugewiesen werden. Das Verfahren des *map-matching* wurde dabei ursprünglich für die Robotik entwickelt und gilt als ein Prinzip der Positionsbestimmung beim SLAM[1] Verfahren (*Simultaneous Localization and Mapping*) [Harle (2013)]. Die Resultate der Arbeit (siehe Abb. 3.3, 3.2a, 3.2b) machen deutlich, dass auch aus der Robotik entstandene Verfahren in der Fußgängernavigation Anwendung finden können. Beide Routen sind von der eigentlichen Ground-Truth nur schwer zu unterscheiden. Die Tabelle 3.3 zeigt bei der ersten Route einen Fehler $< 3\ m$ bei 96.2% der Zeit. Die zweite Route erreichte sogar einen Wert von rund 99.3%. Ein entsprechender Ansatz mit einem Partikelfilter statt eines HMM ist in [Yanghuan u. a. (2014)] zu finden. Dort wurde eine durchschnittliche Fehlerabweichung von $<\ 2.5\ m$ erreicht. Ein anderer Ansatz, um die Abweichungen, die ein konventioneller PDR-Algorithmus mit sich bringt, besteht in der Verknüpfung mit der Fingerprint-Methode aus dem Bereich der Indoor-Positionierung. Dieser Ansatz wurde in [Hilsenbeck u. a. (2014)] verfolgt und implementiert. Die Fingerprinting-Methode arbeitet auf Grundlage typischer Signalstärkeverteilungen (z. B. WLAN, Magnetometer) der Umgebung. Die Fingerprints können dabei auch als sog. Signaturen bezeichnet werden. Dabei wird eine Reihe von Punkten an bekannten Positionen gemessen, die möglichst gleichmäßig über den Bereich verteilt sind. Bei den Messungen handelt es sich bspw. um die empfangenen Werte der Signalstärke mehrerer Basisstationen, welche im Anschluss in einer Datenbank abgespeichert werden. Die Experimente wurden auf einem $5000m^2$ großen Bürokomplex durchgeführt. Die Teststrecke betrug $300\ m$ und umfasste eine ungefähre Dauer von 5 Minuten. Das Ergebnis zeigte von insgesamt 700 Versuchen eine Genauigkeit $< 2.19m$ nach 50% der Zeit, nach 90% der Zeit wuchs der Fehler bereits auf $7.16m$ an. Zusammenfassend lässt sich sagen, dass dieser Ansatz sich als Lösung für Flughäfen, Museen, Universitäten sowie große Bürogebäude zwar grundsätzlich

[1] Neben der Personen- und Fahrzeugnavigation, existiert noch das aus der Robotik stammende Prinzip der Positionsbestimmung, welches als *SLAM-Simultaneous Localization and Mapping* bekannt ist. Es wird als ein Problem der Robotik bezeichnet, weil der mobile Roboter eine Karte seiner Umgebung erstellen und parallel seine Pose innerhalb dieser Karte schätzen muss. Um dem Roboter die Wegfindung zu ermöglichen und sich entsprechend in seiner Umgebung bzw. Karte zu orientieren ist eine Art Positionsbestimmung notwendig. Ohne SLAM müsste vor dem Einsatz eine Karte erstellt werden, was den Einsatz verzögern und verteuern würde. Daher ist es wichtig, dass ein Roboter in der Lage ist, völlig autonom eine neue Umgebung zu erkunden und eine Karte zu erstellen, die er dann später zur Navigation nutzen kann. SLAM gilt als ein aktives Forschungsgebiet der Robotik, welches weltweit von zahlreichen Forschergruppen bearbeitet wird.



(a) Route 1 (b) Route 2

Abbildung 3.2.: Trajektorien beider gelaufener Routen im Vergleich mit der Ground-Truth [Lamy-Perbal u. a. (2015)]

Path 1	max(error) (m)	mean(error) (m)	% time $\leq$ 1m	% time $\leq$ 3m
without HMM	26.4	9.75	6	16
with HMM	4.8	1.3	42	96.2
Path 2	max(error) (m)	mean(error) (m)	% time $\leq$ 1m	% time $\leq$ 3m
without HMM	5.77	2.57	24	54
with HMM	3.05	1.05	59	99.3

Abbildung 3.3.: Tabellarische Übersicht der Ergebnisse beider Routen und Darstellung des Fehlers zwischen der geschätzten Trajektorie vom HMM und Ground-Truth [Lamy-Perbal u. a. (2015)].

anbieten würde, dass aber alternativen gefunden werden müssen, sobald die Umgebungen einen erhöhten Anteil an Komplexität aufweisen.

3.2. Deep-Learning-basierte Ansätze

Deep-Learning-basierte Ansätze haben ihr enormes Potenzial bereits mehrfach unter Beweis gestellt. In der Sprachverarbeitung [Keneshloo u. a. (2018)], Spracherkennung [Lakkhanawannakun und Noyunsan (2019)] sowie in der Bildverarbeitung und Bildanalyse [Latif u. a. (2019)] sind diese Ansätze in ihren Gebieten bereits Vorreiter. Im Bereich der visuellen Odometrie (VIO) hat man sich dieses Potenzial zunutze gemacht und integrierte daraufhin diese Ansätze in die Arbeit [Clark u. a. (2017a)], um mittels VIO eine Posenbestimmung[2] durchzuführen. Später wurde dieser Ansatz von derselben Forschungsgruppe [Clark u. a. (2017b)] durch Inertialsensoren ergänzt. Beide Lösungsvorschläge setzen auf eine Kombination von CNN und RNN. Vor allem visuelle Verfahren haben den großen Vorteil ohne jegliche künstliche Infrastruktur zu funktionieren. Anhand visueller markanter Punkte der Umgebung können Verfahren der visuellen Odometrie die Eigenbewegung der Kamera bestimmen, um daraus schließlich die eigene Position abzuleiten. In den Arbeiten wurden die Convolutional Networks mit sechs Layer und die Recurrent Networks mit zwei Layer modelliert. Als Datensatz wurde der KITTI[3] mit der Sequenz *Seq-00* [Geiger u. a. (2012)] verwendet. Die Arbeit demonstrierte den großen Vorteil hinsichtlich der Fehlervermeidung bei der Kalibrierung und Synchronisierung der Kamera mit den Sensoren. Des Weiteren zeigten die abschließende Trajektorie und der dazugehörige Box-Whisker-Plot eine starke Fehlerausbreitung der $500\ m$ langen Teststrecke. Die Autoren zeigten sich dennoch optimistisch und wollten bereits in Kürze mit weiteren Experimenten fortfahren. Eine aufbauende Arbeit [Liu u. a. (2019)] befasste sich mit einem ähnlichen Ansatz und verfolgte das Ziel einer Lokalisierung und Kartierung für eine mobile Roboternavigation. Der vorgeschlagene DL-Ansatz soll dabei später das komplexe SLAM-Verfahren ersetzen.
Da die inertiale Navigation im Wesentlichen von der Qualität der Sensordaten abhängt, schlugen [Chen u. a. (2018)] einen DL-Ansatz mit CNN (Convolutional Neural Networks) vor, um das Problem der inhärenten Fehler der MEMS-Inertialsensoren zu beseitigen. Zur Bewältigung der Aufgabe sollten insgesamt fünf Convolutional Layer mit Pooling Layer zum Einsatz kommen. Bei der Durchführung der Experimente wurde nach unterschiedlichen Batch-Sizes (150, 200) und Trainingsepochen (20, 25) trainiert. Mit einem Beschleunigungssensor (1000 Hz) wurden

[2] Die Pose einer Kamera kann rein visuell bestimmt werden und besteht aus der Orientierung und Position des bewegten Kamerakoordinatensystems in Bezug auf ein festes Weltkoordinatensystem.
[3] Der KITTI-Trainingsdatensatz besteht aus 21 Trainingssequenzen, aus Stadtfahrten, Autobahnfahrten, Landstraßenfahrten und Fußgängerzonenfahrten.

insgesamt 100 Datensätze gesammelt. Jeder Datensatz enthielt Beschleunigungsdaten über eine ungefähre Dauer von 10 Sekunden. Es konnte eine maximale Genauigkeit von 79.48% mit einer Batch-Size von 150 und 25 Epochen erreicht werden.

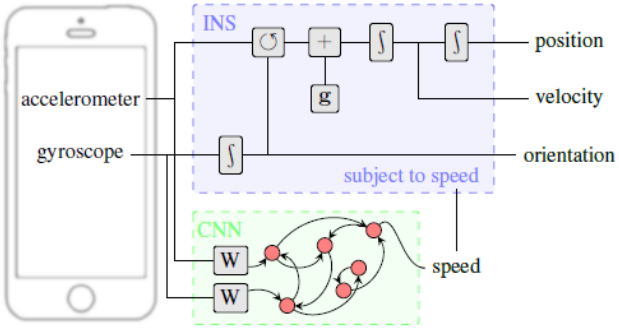
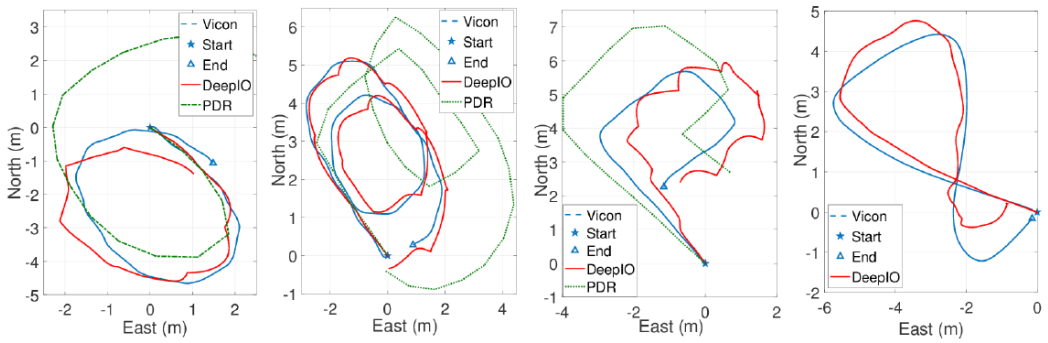


Abbildung 3.4.: Ein Faltungsnetzwerk wird verwendet, um eine Geschwindigkeitsschätzung durchzuführen [Cortés u. a. (2018)].

Daraufhin erweiterten Cortés, Solin und Kannala noch im selben Jahr eine vorhergegangene Arbeit [Solin u. a. (2017)] mit diesem Ansatz und implementierten daraufhin die Faltungsnetze zur Geschwindigkeitsschätzung und Klassifikation unterschiedlicher Fortbewegungsarten (Static, Walking, Stairs, Elevator, Escalator) [Cortés u. a. (2018)].In der vorhergegangen Arbeit [Solin u. a. (2017)] ging es um eine relative Positionsbestimmung bei einer im Vorhinein definierten Geschwindigkeit. Damit entging man dem Problem durch das Aufsummieren fehlerhafter Messwerte. In dem neuen Ansatz soll das CNN die Geschwindigkeitsschätzung übernehmen und fortlaufend aktualisieren (Abbildung 3.4). Die Klassifikationen geben dabei die zu lernenden Geschwindigkeiten an. Dazu soll immer ein Zeitfenster von 2 Sekunden betrachtet und in das CNN eingespeist werden. Im Vergleich zur vorhergegangenen Arbeit konnte durch das CNN eine Verbesserung von 28% erzielt werden. Die durchschnittliche Abweichung betrug noch 0.62 m. In [Yan u. a. (2017)] wurde die Geschwindigkeitsschätzung mithilfe einer Support Vector Regression (SVR) durchgeführt, wohingegen sie in [Chen u. a. (2018b)] mit Recurrent Neural Networks realisiert wurde. Im Gegensatz zu [Cortés u. a. (2018)] wurden in [Chen u. a. (2018a)] nicht nur einzelne Komponenten eines inertialen Navigationssystems durch DL-Methoden ersetzt, sondern es wurde das vollständige INS in ein DL-Modell auf Basis von RNNs überführt. In [Chen u. a. (2018a), Chen u. a. (2018b)] wurde dafür ein kompletter Raum mit einem *Optical Motion Capturing System (Vicon)* ausgestattet (siehe Abb. 3.6). Mit dem Erfassungssystem konnten exakte Ground-Truths erzeugt und Trainingsdaten unterschiedlichster Bewegungsformen gesammelt werden. Die erzielten Resultate können



(a) Handheld (b) In Pocket (c) In Handbag (d) On Trolley

Abbildung 3.5.: Bestimmung der Trajektorien unterschiedlicher Bewegungsformen (a) in der Hand (b), in der Hosentasche (c), in der Handtasche (d), auf einem Trolley mit der Vicon (blau) als Ground-Truth [Chen u. a. (2018b)].

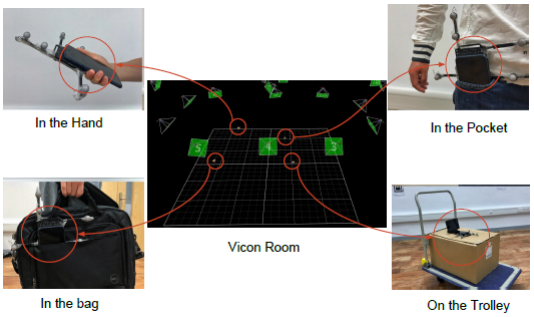


Abbildung 3.6.: Aufzeichnung von Trainingsdaten mithilfe des Vicon-Erfassungssystems [Chen u. a. (2018b)]

der Abbildung 3.5 entnommen werden. Die blaue Trajektorie in Abb. 3.5 entspricht die des Vivon-Erfassungssystems und stellt die Ground-Truth des Experiments dar.

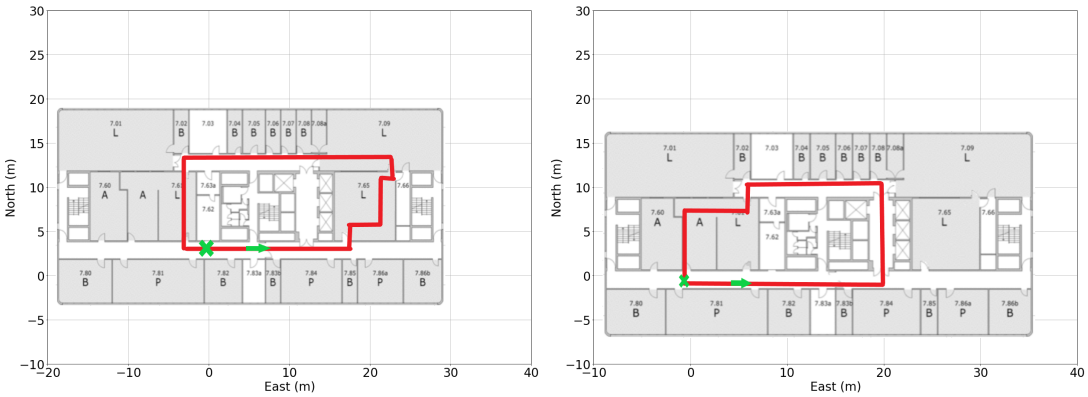
4. Datensatz

In Anbetracht der verfügbaren Datensätze zur Durchführung einer inertialen Navigation für IMU-basierte Aufgaben mittels Low-cost-Sensoren, wird schnell deutlich, dass auf dem Markt noch ein gewisser Mangel herrscht. Die meisten vorhandenen Datensätze in diesen Bereichen haben das autonome Fahren, die visuelle Entfernungsmessung, 3D-Objekterkennung oder 3D-Tracking zum Schwerpunkt. Zum Einsatz kommen dabei immer kostspielige IMUs mit einer entsprechend hohen Genauigkeit sowie mit GPS oder einem Laserscanner kombiniert, wie z. B. der KITTI- [Geiger u. a. (2013)] oder EuRoC-MAV [Burri u. a. (2016)] Datensatz. In [Chen u. a. (2018b)] wurden stattdessen Daten mit einem optischen Bewegungserfassungssystem (Vicon) in einem extra dafür angefertigten $(5m \times 8m)$ Vicon-Raum aufgezeichnet (siehe Abb. 3.5). Andere Datensätze wiederum, wie es z. B. beim Tum-VI [Schubert u. a. (2018)] und ADVIO [Reina u. a. (2018)] der Fall ist, haben eine Lokalisierung der Position mittels visueller-inertiale Odometrie zum Ziel. Obwohl bei diesen Datensätzen inertiale Navigationsdaten verwendet werden, ist ihr Anwendungspotenzial stark begrenzt. Denn die inertiale Messeinheit (IMU) muss an einer festen Position angebracht und darf im Nachhinein nicht mehr bewegt werden. Zudem müssen die Kameras immer in Laufrichtung gehalten werden.

Um den Anforderungen und Zielen gerecht zu werden, wurde im Rahmen der Arbeit und für die Durchführung eigener Experimente und Untersuchungen ein eigener Datensatz erstellt. Allerdings ist eine Menge an Daten fürs Training, Validieren und Testen erforderlich. Darüber hinaus ist ein sehr präzises Labeln bzw. Beschriften der Trainingsdaten notwendig, weshalb die nachfolgenden Abschnitte sich mit diesen Themen näher befassen.

4.1. Beschaffung von Trainingsdaten

Die eben beschriebenen verwandten Datensätze sind für Personen weniger flexibel und benötigen oftmals zusätzliche Komponenten zur Umsetzung. Personen sind bestrebt ihr Smartphone beim Gehen entweder in der Hand oder in der Hosentasche zu tragen, aber auch das Tragen von Sensoren in Schuhen oder Einlegesohlen zur Analyse der Lauftechnik ist bereits weit verbreitet.



(a) Route 1 (b) Route 2

Abbildung 4.1.: Die geplanten Laufstrecken beider Routen.

Wie in Kapitel 1.2 erläutert, werden für die Erstellung eines eigenen Datensatzes Daten zweier gängiger Bewegungsformen auf zwei unterschiedlichen Strecken aufgezeichnet.
Bei der ersten Bewegungsform handelt es sich um eine typische Schwingbewegung, bei der das Smartphone in der Hand getragen wird (engl. handheld). Bei der zweiten Bewegungsform wird der Sensor stattdessen am Fußrücken montiert (engl. foot-mounted) getragen. Um die Umsetzbarkeit des entwickelten inertialen Navigationssystems auch auf seine Praxistauglichkeit hin zu untersuchen, wurden Daten von insgesamt drei Inertialsensoren erfasst. Zudem nahmen drei Teilnehmer unterschiedlichster Größe ebenfalls an den Experimenten teil. Die Daten wurden auf dem Campus-Gelände der HAW Hamburg in den Laboren des Departments Informatik gesammelt. Eine vollständige Auflistung der gesammelten Daten, gemäß den Bewegungsformen unterteilt, ist in Tabelle 4.1 aufgeführt. Der Datensatz umfasst (eine Menge von) 162 Sequenzen. Eine Sequenz entspricht dabei einer gelaufenen Strecke. Die Dauer einer Strecke misst eine Zeit zwischen $80\,\text{sec} - 85\,\text{sec}$, wobei im Durchschnitt die Dauer der Route 1 bei $80\,\text{sec}$ und die Route 2 bei $85\,\text{sec}$ lag. Für die Laufstrecken wurden Sensordaten mit drei unterschiedlichen Geräten aufgezeichnet. Die verwendeten Geräte können der Tabelle 4.1 entnommen werden. Alle drei Geräte beinhalten einen 3-achsigen Low-cost-Inertialsensor (9DoF), bestehend aus Beschleunigungssensor, Gyroskop und einem zusätzlichen Magnetometer (siehe Tabelle 6.1). Mit diesen Sensoren wurden die notwendigen Messungen der einzelnen Sequenzen aufgenommen.
Dabei wurde bei der Beschaffung der Daten das Ziel verfolgt, mit dem IMU-9DoF-Sensor einen Datensatz an der Seite zu haben, der eine hohe Datenqualität aufweisen kann. Zum einen, um überhaupt eine Ground-Truth erzeugen zu können, zum anderen, um sicherzustellen und

demonstrieren zu können, dass neuronale Netze für diese Aufgabe generell geeignet sind, und außerdem, um eine gute Mischung an Daten (möglichst effektiver Datensatz) für den Lernalgorithmus bereitstellen zu können. Dies geschieht auf dem Hintergrund, dass die eingebauten Sensoren in Smartphones keine Kalibrierung in Vorhinein erfahren haben und somit davon auszugehen ist, dass sie grundsätzlich keine qualitativen Daten erzeugen, stattdessen aber fehlerbehaftete Sensorwerte liefern. Das DL-Modell wurden zudem für diesen Zweck als Ensemble entworfen und implementiert. Eine inertiale Navigation muss auch weiterhin bei fehlerbehafteten Messwerten garantiert werden. Des Weiteren führt das Einbringen von Rauschen in ein neuronales Netz zur Anreicherung der Daten [Sietsma und Dow (1991)].
Um das Drift- und Rauschverhalten des IMU-9DoF-Sensors während der Datenaufzeichnung weitestgehend gering zu halten, wurde nach jeder aufgenommenen Datensequenz (Strecke) eine Pause von mindestens 60 und 120 Sekunden eingehalten. Es ist bekannt, dass das Rauschen des Sensors prinzipiell von der Wärmeentwicklung abhängig ist: Das Ziel, diese Fehlerquelle (temperaturabhängige Fehleranteile) weitestgehend auszuschließen oder möglichst gering zu halten ist der Grund, weshalb diese Maßnahmen ergriffen wurden. Insgesamt umfasst der Datensatz eine Gehzeit von ungefähr $13.155\,\mathrm{sec}$ ($219\,\mathrm{min}$). Eine Raumübersicht des Labors beider Strecken stellen die Abbildungen 4.1a und 4.1b dar. Die Abbildungen unterscheiden sich dabei jeweils nur durch die in rot gekennzeichneten Linien, was die geplanten Laufstrecken der Routen widerspiegeln soll. Wichtig dabei zu erwähnen ist, dass es sich lediglich um die geplanten Laufstrecken und nicht um die eigentlichen Ground-Truths handelt. Die Karten richten sich nicht nach einen Maßstab, sind also nicht Maßstabsgetreu und mussten daher anhand der aufgezeichneten Daten geschätzt werden. Die Strecke 4.1a beinhaltet die Schwierigkeit enger Kurven, bei der Strecke 4.1b hingegen müssen zwei schwere Feuerschutztüren überwunden werden. Für die Erzeugung und Berechnung der Ground-Truths (GT) war ausschließlich der IMU 9DoF (Degree of Freedom) geeignet (siehe Kap. 6.1.2, Abb. 6.1). Die GT-Erstellung wird im nachfolgenden Kapitel 4.3 näher erläutert.

4.2. Vorverarbeitung der Trainingsdaten

Für Machine-Learning-Algorithmen sind die Qualität der Daten sowie der Umfang der darin enthaltenen Informationen entscheidende Faktoren, die festlegen, wie gut ein Lernalgorithmus Daten verallgemeinern kann. Die Art der Vorverarbeitung für den Lernalgorithmus in dieser Arbeit bestand darin, jede Sequenz des Datensatzes mittels PDR-Algorithmus zu visualisieren, vor der Analyse zu vereinheitlichen und auf ihre Datenqualität hin zu untersuchen. Da es sich um ein Supervised-Learning-(dt. Überwachtes Lernen) Problem handelt, müssen die

Tabelle 4.1.: Aufgezeichneter Sensordatensatz.

	Type	Sequenzen FM	Sequenzen HH	Time (s)
Bewegungsformen	Handheld (HH)		**83**	
	Foot (FM)	**73**		
Devices	IMU (9DoF)	24	26	
	iPhone 5	19	24	
	iPhone 6	21	24	
Users (IMU (9DoF))	User 1 ($1.83\ m$)	3	3	
	User 2 ($1.75\ m$)	3	3	
	User 3 ($1.91\ m$)	3	3	
Routen	Route 1	31	41	≈ 5760
	Route 2	45	42	≈ 7395
Gesamt		**162**		**13.155**

Sensordaten einer Sequenz einem entsprechenden Label bzw. korrespondierenden Daten, in diesem Fall der Ground-Truth, zugeordnet werden. Dafür war es erforderlich, die Dauer jeder Sequenz des Datensatzes der Größe der Ground-Truth anzupassen. Parallel dazu musste kontinuierlich mittels PDR-Algorithmus überprüft werden, ob sich nach den Anpassungen noch eine möglichst fehlerfreie Trajektorie erzeugen lässt. Dieser Schritt war notwendig, weil die Generalisierungsfähigkeit des DL-Algorithmus maßgeblich von der Qualität der Trainings- und Testdaten abhängt. Dadurch sollte eine hohe Qualität der Trainingsdaten sichergestellt werden.
Zusätzlich mussten stark verrauschte und von Störgrößen beeinflusste Sensordaten zur Rauschreduzierung mit einem Median-Filter vorverarbeitet werden. Nachdem die Signale von sämtlichen Störeinflüssen bereinigt worden waren, wurden die Daten im Nachhinein noch wie folgt standardisiert:

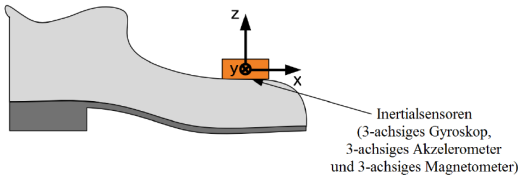
$$\begin{aligned} z &= \frac{x-\mu}{\sigma},\ mit \\ \mu &= \frac{1}{N}\sum_{i=1}^{N}(x_i)\ und \\ \sigma &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2} \end{aligned} \tag{4.1}$$

Bei der Standardisierung z in Gleichung 4.1 wird der berechnete Mittelwert μ von den Daten x subtrahiert und im Anschluss durch die Standardabweichung σ geteilt. Damit besitzen die standardisierten Werte nun einen Mittelwert von Null und eine entstehende Verteilung der Varianz von 1. Die Standardisierung kann dazu beitragen, beim Trainieren das Gradientenabstiegsverfahren zu verbessern, weil zum Auffinden einer guten oder optimalen Lösung (das globale Minimum) weniger Schritte erforderlich sind.
Ein weiterer Vorverarbeitungsschritt ist die Transformation der Daten mithilfe der *Sliding-Window-* (dt. gleitendes Fenster) Technik. Dazu wird ein *Sliding-Window* auf den kompletten Datensatz angewandt, die Fenstergröße (entspr. der Größe der Timesteps) bestimmt dabei den zu beobachtenden Zeitraum. Die Schrittweite, um die verschoben wird, ist auf 1 Sekunde festgelegt. Der Datensatz mit den Abmessungen von $[n\_sample,\ n\_input]$ wird dazu in die Form $[n\_sample,\ n\_steps,\ n\_input]$ umstrukturiert. Dabei ist mit n_sample die Größe des Datensatzes, n_input die Eingabesequenz und n_steps die zu beobachtende Sequenz (Timesteps), in der temporale Abhängigkeiten erkannt werden sollen.

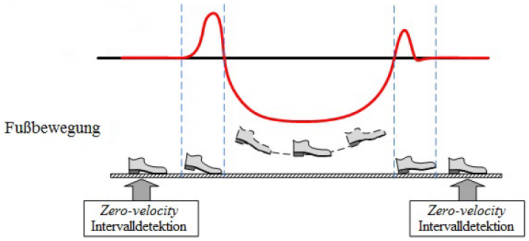
4.3. MEMS-basierte Positionsbestimmung - (Ground-Truth)

Dieses Kapitel befasst sich mit der Erzeugung der Ground-Truth und stellt dazu den implementierten PDR-Algorithmus vor. Wie in Kapitel 1.2 beschrieben, müssen sowohl für die Bewegungsform *foot-mounted* als auch *handheld* Ground-Truths erstellt werden. Bei der *foot-mounted* Navigation besteht die Ground-Truth aus den exakten Positionsdaten, woraus sich problemlos eine Trajektorie erzeugen lässt. Bei der Bewegungsform *handheld* entspricht die Ground-Truth dem Heading (Yaw-Winkel), dabei handelt es sich um die Orientierung. Für die Ground-Truth-Erstellung kam ein kostengünstiger IMU 9DoF (Degree of Freedom) zum Einsatz (siehe Kapitel 6.1.2). Mit diesem IMU-Sensor wurden zum einen Trainingsdaten gesammelt und zum anderen bildete er die Grundlage für die Berechnungen einer möglichst fehlerfreien Ground-Truth-Trajektorie.

Bei der relativen Bestimmung der Position mit inertialen Sensoren kann prinzipiell zwischen einem schritt-basierten oder einem inertial-basierten Algorithmus unterschieden werden. Wie in Kapitel 3 beschrieben, sind auch PDR-Algorithmen keine fehlerfreien Systeme und benötigen ebenfalls ein geeignetes Fehlerkorrektursystem. Sie unterliegen ebenfalls dem Problem, dass sich Fehler mit der Zeit aufsummieren, weshalb in diesem Ansatz eine Methode verwendet werden soll, mit der dieser Fehler möglichst klein gehalten werden kann, um so eine möglichst fehlerfreie Ground-Truth-Erzeugung zu garantieren. Zur GT-Erstellung wurde der schritt-



(a) Montierung des Sensors am Fußrücken nach [Park und Suh (2010b)]



(b) Bewegung des Fußes während einer Schrittdetektion mit simuliertem Sensorsignalverlauf [Park und Suh (2010b)]

Abbildung 4.2.: Darstellung der Bewegungsform mit der Montierung des Sensors am Fußrücken

basierte PDR-Algorithmus gewählt und implementiert. Dieser Ansatz kann in in drei Phasen unterteilt werden:

- Schrittdetektion
- Schrittlängenschätzung
- Richtungsschätzung.

Dabei existieren zahlreiche Möglichkeiten einen *Pedestrian Dead Reckoning (PDR)* zu implementieren. Die Wahl der Algorithmen der einzelnen Phasen hat sich an Ansätzen, die dem aktuellen Stand der Forschung entsprechen, orientiert. Bei der Beschreibung der Ansätze zur Positionsbestimmung war es nicht möglich die verwendeten Methoden auf beide Bewegungsformen (*foot-mounted* und *handheld*) gleichermaßen anzuwenden. Dies wurde bei der Implementierung berücksichtigt, weshalb auf vergleichbare Alternativen zurückgegriffen wurde. Für die Schrittdetektion wurde ein *Zero-Velocity-Interval*-Algorithmus (ZVI-Algorithmus) nach [Park und Suh (2010b)] implementiert. Diese Methode eignet sich dabei lediglich nur bei einer Schrittdetektion mit der Montierung des IMUs am Fußrücken, da man sich hier die Bewegung des Nutzers zu eigen macht. Für das *Handheld*-Szenario musste stattdessen ein weiterer Algorithmus betrachtet werden. Dafür kam ein *Exponentially-Weighted-Moving-Average*-Algorithmus (EWMA-Algorithmus), kombiniert mit einer anschließenden Peak-Detektion, zum Einsatz. Im nachfolgenden Kapitel werden die Methoden näher erläutert.

4.3.1. Schrittdetektion

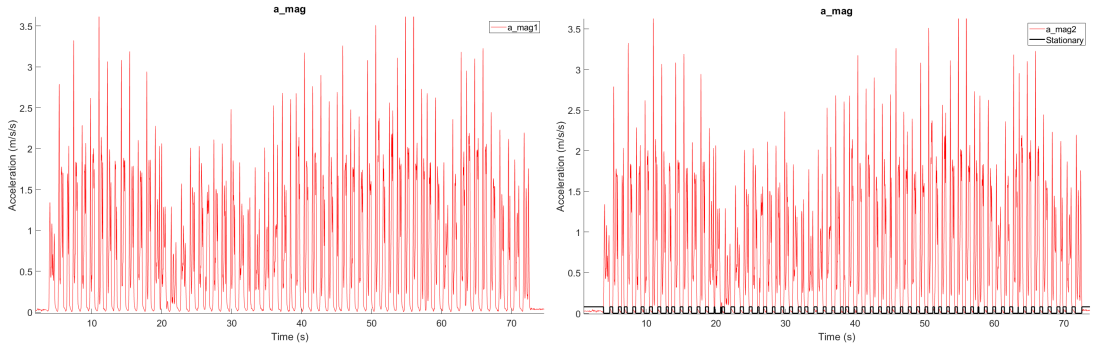
Bei der Schrittdetektion kann zwischen einem Doppelschritt und einem einzelnen Schritt unterschieden werden. Dabei unterscheiden sie sich prinzipiell nur in der Länge. Zudem existieren auch zahlreiche Möglichkeiten einen Schrittzähler zu implementieren. Schritte können

dabei mithilfe eines Gyroskops oder mit einem Beschleunigungssensor detektiert werden. In dieser Arbeit übernimmt der Beschleunigungssensor die Aufgabe der Schrittdetektion eines Doppelschrittes. Für den ZVI-Algorithmus muss der Sensor wie in Abb. 4.2a am Fuß befestigt werden. Dadurch erhält man während der Bewegung des Fußes einen periodischen Verlauf von Sensorsignalen (siehe Abb. 4.2b). Nach [Park und Suh (2010b)] können Schritte sowohl anhand der z-Achse des Beschleunigungsmessers als auch anhand der y-Achse des Gyroskops abgeleitet werden. Um den Algorithmus unabhängig von dem Bewegungsmodus und der Drehung eines Gerätes zu implementieren, wird stattdessen die Magnitude des Beschleunigungssensors verwendet. Dadurch gewinnt der Algorithmus zusätzlich an Robustheit und Zuverlässigkeit.

$$\alpha_{mag} = \sqrt{a_x^2 + a_y^2 + a_z^2}\,, \tag{4.2}$$

wobei a_x, a_y und a_z die Messungen vom 3-achsigen Beschleunigungsmesser sind. Zusätzlich kam zur Rauschreduzierung ein Tiefpassfilter zur Anwendung. Das Resultat der Anwendung des Tiefpassfilters auf α_{mag} wird nun als α_{mag_zvi} bezeichnet. In der Abbildung 4.3a ist der periodische Verlauf der Schritte nach dem Anwenden des Tiefpassfilters deutlich erkennbar. Der nächste Schritt des ZVI-Algorithmus sieht eine Einteilung der detektierten Schritte in Intervalle vor. Dabei sollen exakt die Zeiträume in α_{mag_zvi} erfasst werden, wo der Fuß den Boden berührt und eine Nullgeschwindigkeit erzeugt. Dieser Zeitraum wird üblicherweise als *Zero-Velocity Interval* bezeichnet. Zur Erfassung der ZVI-Zeiträume wird ein Threshold-Wert α_{thres} mit 0.4 definiert, welcher als Schranke dienen soll. Solange $\mid \alpha_{mag\_zvi} \mid$ sich innerhalb des definierten Schrankenwerts α_{thres} befindet, kann eine Nullgeschwindigkeit detektiert werden ($\mid \alpha_{mag\_zvi} \mid \leq \alpha_{thres}$). Die Abbildung 4.3b veranschaulicht die identifizierten *Zero-Velocity*-Intervalle als schwarze Flanken, was die detektierten Schritte darstellen soll. Da der ZVI-Algorithmus sich nur für am Fuß montierte Sensoren eignet, wurde für das *Handheld*-Szenario ein weiterer Algorithmus betrachtet und implementiert – der *Exponentially-Weighted-Moving-Average*-Algorithmus (EWMA-Algorithmus). Der EWMA-Algorithmus ermöglicht es Schritte unabhängig von der Position zu detektieren und registriert zudem auch ein "Stehenbleiben" der Person. Wie bereits beim ZVI-Algorithmus wird auch bei dem EWMA-Algorithmus der Beschleunigungsvektor aus Gleichung 4.2 berechnet. Die Verwendung von α_{mag} ermöglicht es auch dem *Handheld*-Bewegungsmodus Schritte unabhängig von der Lage des IMUs zu detektieren. Im Anschluss erfolgt eine Tiefpassfilterung mittels gleitenden Mittelwertes:

$$Y_t = \frac{1}{n} \sum_{i=0}^{n-1} \alpha_{t-i} \tag{4.3}$$



(a) Gefilterten Magnitude des Beschleunigungssensors

(b) Einteilung der gefilterten Magnitude in Intervalle

Abbildung 4.3.: Darstellung der detektierten Schritte durch die Einteilung der gefilterten Magnitude in Intervalle (ZVI-Algorithmus).

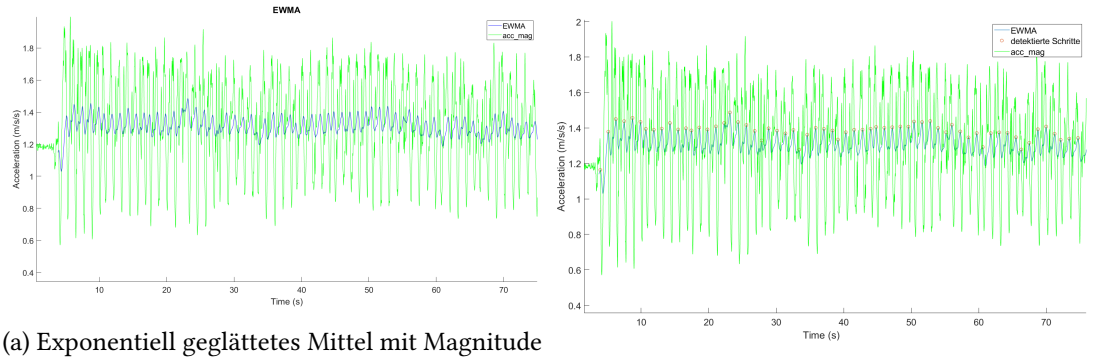
aus Y_t wird nun ein exponentiell geglättetes Mittel (*Exponentially-Weighted-Moving-Average*) berechnet. Die Berechnung des EWMA entspricht einer *Infinite-Impulse-Response*-Filterung mit einer exponentiellen Gewichtung α. Der Wert α liegt dabei in einem Bereich zwischen 0 und 1.

$$E_t = \alpha \cdot Y_t + (1 - \alpha) \cdot E_{t-1} \tag{4.4}$$

Aus der Gleichung geht hervor, dass α den Einfluss der gefilterten Y_t in die E_t-Berechnung bestimmt. Die Abbildung 4.4a zeigt den Beschleunigungsvektor zusammen mit dem daraus resultierenden berechneten exponentiell geglätteten Mittel. Zum Schluss wurde noch eine *Peak-Detektion* angewendet, um die Schritte zu identifizieren (siehe Abbildung 4.4b). Als α-Wert wurde hier für den EWMA 0.01 gewählt. Eine andere Möglichkeit wäre auch gewesen, einen kleineren α-Wert zu wählen, wodurch sich der EWMA einer geraden Linie annähern würde. Danach wäre wiederum eine Einteilung in Intervalle möglich, um Schritte entsprechend zu detektieren.

4.3.2. Schrittlängenschätzung

Nachdem die erste Phase des PDR-Algorithmus abgeschlossen ist, folgt die zweite Phase mit der Schrittlängenschätzung. Für den Bewegungsmodus *foot-mounted* ergibt sich die Schrittlänge



(a) Exponentiell geglättetes Mittel mit Magnitude des Beschleunigungssensors

(b) Peak Detektion

Abbildung 4.4.: Detektierte Schritte mithilfe von EWMA

schließlich aus der Position durch Integration der Beschleunigung und Geschwindigkeit der *none-zero-velocity*-Intervalle.

$$\vec{v}(t) = \vec{v}(t-1) + \int_0^t \vec{a}(t)dt \tag{4.5}$$

$$\vec{p}(t) = \vec{p}(t-1) + \int_0^t \vec{v}(t)dt \tag{4.6}$$

Nun wird ersichtlich, welchen großen Vorteil der ZVI-Algorithmus für die Positionsbestimmung hat. Eine Integration der Beschleunigung wird dabei nur bei einer tatsächlichen Bewegung vorgenommen, was den *None-Zero-Velocity*-Intervall-Phasen entspricht. Dadurch können schließlich die Ursachen für Navigationsfehler, welche unter anderem durch numerische Fehler als auch Initialisierungs- oder Messfehler der Sensoren hervorgerufen werden, auf ein Minimum reduziert werden. Denn während der *Zero-Velocity*-Phasen wirkt keine Beschleunigung und der bereits aufsummierte Fehler kann gelöscht bzw. mit Null initialisiert werden.

Für die Schrittlängenschätzung einer *Handheld*-Bewegungsform existieren eine Reihe verschiedener Ansätze. In dieser Arbeit wurde ein Algorithmus nach [Alvarez u. a. (2006)] - „Schrittlänge durch vertikale Verschiebung“ - gewählt und implementiert. Die Grundlage des Ansatzes basiert auf der Idee der Gleichsetzung des menschlichen Ganges mit einem invertierten Pendelmodell (siehe Abb. 4.5). Aus der Abbildung wird ersichtlich, dass die Schrittlänge von zwei Faktoren abhängig ist, einmal der Beinlänge l und der Höhe h. Die maximale Höhe tritt auf, sobald sich der Fuß vollständig auf dem Boden befindet. Dabei entspricht die Höhe

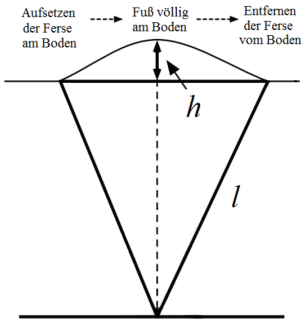


Abbildung 4.5.: Der menschliche Gang als invertiertes Pendelmodell nach [Alvarez u. a. (2006)]

der vertikalen Verschiebung des Körperzentrums bei der Durchführung eines Schrittes. Die Schrittlänge SL lässt sich mittels Höhe h und der Beinlänge l wie folgt berechnen:

$$SL = K2\sqrt{2lh - h^2}\,. \tag{4.7}$$

Bei der Variablen K handelt es sich um eine Kalibrierkonstante, welche von Person zu Person unterschiedlich ist. Sie muss zuvor durch Testmessungen kalibriert oder geschätzt werden. Die vertikale Beschleunigung erhält man im Anschluss durch eine doppelte Integration. Dabei erfolgt die Integration für jeden Schritt einzeln.

4.3.3. Richtungsschätzung

Die Richtungsbestimmung wird häufig auch als Heading bezeichnet und entspricht dem Winkel zwischen der Fortbewegungsrichtung und der Nordrichtung. Für die Bestimmung der Richtung kommen die inertialen Sensoren zum Einsatz. Dazu zählen sowohl der Beschleunigungssensor, das Gyroskop als auch das Magnetometer. Bei der Richtungsbestimmung muss berücksichtigt werden, dass es sich hierbei um eine Indoor-Navigation handelt. Indoor-Umgebungen sind im Gegensatz zu Outdoor-Umgebungen für Sensoren und vor allem MEMS-Inertialsensoren sehr viel störanfälliger, was sich schließlich auf die Berechnungen auswirken kann. Für die Richtungsbestimmung in solchen Fällen wird typischerweise direkt auf einen Kalmanfilter zurückgegriffen. Neben dem Kalmanfilter existieren mittlerweile auch anderer Algorithmen, mit denen identische Resultate erzielt werden können. Der in dieser Arbeit verwendetet Algorithmus zur Bestimmung der Orientierung nennt sich *Madgwick-Filter* und wurde 2011 von Sebastian Madgwick im Rahmen seiner Doktorarbeit entwickelt (vgl.[Madgwick (2010)]).
Bei diesem Algorithmus handelt es sich um einen Orientierungsalgorithmus, welcher sich vor allem für Geräte eignet, die in ihren Ressourcen stark begrenzt sind. Der Madgwick-Algorithmus

basiert auf einem Gradientenabstiegsverfahren, verwendet eine Quaternionendarstellung und unterliegt damit nicht der Problematik der Singularitäten. Zudem bietet er zwei Implementierungsarten an, den IMU-basierten und den MARG-basierten Ansatz. Bei dem IMU-basierten Ansatz findet eine Fusion zwischen dem Beschleunigungssensor und dem Gyroskop statt. Bei dem MARG-basierten Ansatz werden Beschleunigungssensor und Gyroskop zusätzlich durch das Magnetometer ergänzt. Außerdem kompensiert der Madgwick-Algorithmus durch das Gradientenverfahren magnetische Störungen und Verzerrungen sowie Sensordrifts. In [Madgwick u. a. (2011)] wurde die Leistung des Madgwick-Filters mit dem des Kalmanfilters verglichen. Die Ergebnisse ergaben nur minimale Abweichungen und zeigten, dass der Madgwick-Filter auf demselben Leistungsniveau wie der des Kalmanfilters arbeitet und eine gute Alternative darstellt. Weitere Experimente sind in [Ludwig und Burnham (2018)] und [Mahony u. a. (2008)] zu finden. In beiden Arbeiten konnte der Madgwick-Filter zuverlässige Resultate mit einer hohen Genauigkeiten erreichen. Folgende Vorteile zeichneten ihn gegenüber dem Kalmanfilter aus:

- eine vergleichsweise einfachere Implementierung mit geringerem Aufwand gegenüber der anspruchsvollen Implementierung des Kalmanfilters
- konstante und zuverlässige Resultate mit unterschiedlichen Abtastraten
- unterstützt sowohl eine IMU-basierte wie auch eine MARG-basierte Implementierung
- wesentlich geringerer Rechenaufwand, dadurch für ressourcen-begrenzte Geräte geeignet

Für beide Bewegungsformen wurde die MARG-basierte Implementierung gewählt. Dadurch erhielt man auf Grundlage von Beschleunigungs- und Drehratensensor sowie Magnetometer die Orientierung als Quaternionen. Im Anschluss fand eine Überführung vom b-frame- in das n-frame- Koordinatensystem statt, wie im Kapitel 2.1.3 beschrieben. Die Überführung des einen Koordinatensystems in das andere lässt sich mit dem Quaternion $\mathbf{q}_b^n$ beschreiben und kann parallel auch direkt zur Transformation eines Vektors in ein anderes Koordinatensystem genutzt werden. Eine Positionsbestimmung der *Foot-mounted*-Bewegungsform kann dabei wie folgt berechnet werden:

$$\vec{v}(t) = \vec{v}(t-1) + ((\mathbf{q}_b^n(t-1) \cdot \vec{a}(t) - g_n))dt \tag{4.8}$$

$$\vec{p}(t) = \vec{p}(t-1) + \vec{v}(t-1)dt\ . \tag{4.9}$$

Die Beschleunigung $\vec{a}$ wird im b-frame gemessen und muss mittels $\mathbf{q}_b^n$ erst in das entsprechende Koordinatensystem transformiert werden. Im Nachhinein kann mittels weiterer Integration die Position berechnet bzw. fortwährend aktualisiert werden.

Für den *Handheld*-Bewegungsmodus kommt, wie oben bereits erwähnt die Integration nicht in Frage. Hier wird stattdessen auf die Rho-Theta-Technik nach [Hofmann-Wellenhof u. a. (2003)] zurückgegriffen. Rho-Theta ist in der Navigation eine Bezeichnung und steht für Navigations- und Ortungsmethoden, die auf Messungen von Polarkoordinaten beruht. Dazu werden die Richtungsmessung (Theta, griech. θ für Winkel) und Distanzmessung (Rho, griech. ρ für range) kombiniert (siehe Gl. 4.10-4.11). Im Vorhinein muss jedoch aus den Quaternionen, wie in Kapitel 2.1.3 beschrieben das Heading (Yaw-Winkel (θ)) bestimmt werden.

$$x_{k+1} = x_k + SL_k \cos\theta_k \tag{4.10}$$

$$y_{k+1} = y_k + SL_k \sin\theta_k \,, \tag{4.11}$$

wobei θ dem Heading, also dem Yaw-Winkel, entspricht, k ein Index für die jeweiligen Schritte ist und $x(k)$ und $y(k)$ der ermittelten Position in Richtungen Nord (x) und Ost (y) entsprechend. Zum Schluss war es möglich aus den Phasen des PDR-Algorithmus für beide Bewegungsformen eine Trajektorie zu bestimmen. Die Abbildungen 4.6 und 4.7 zeigen die mit dem Sensor erzeugten Ground-Truths der *Foot-mounted*-Bewegungsform, für das *Handheld*-Szenario wurden identische Ergebnisse erzielt. Die Abbildung 4.6 entspricht der GT-Trajektorie der Route 1 und 4.7 der GT-Trajektorie für Route 2.

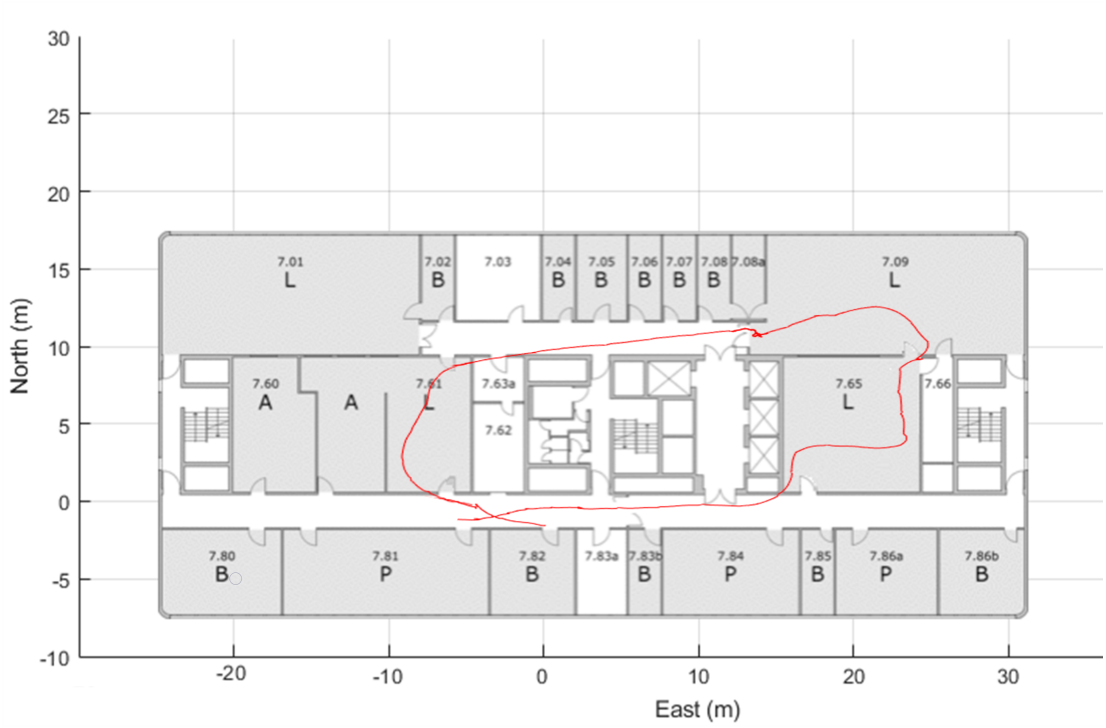


Abbildung 4.6.: Ground-Truth für Route 1

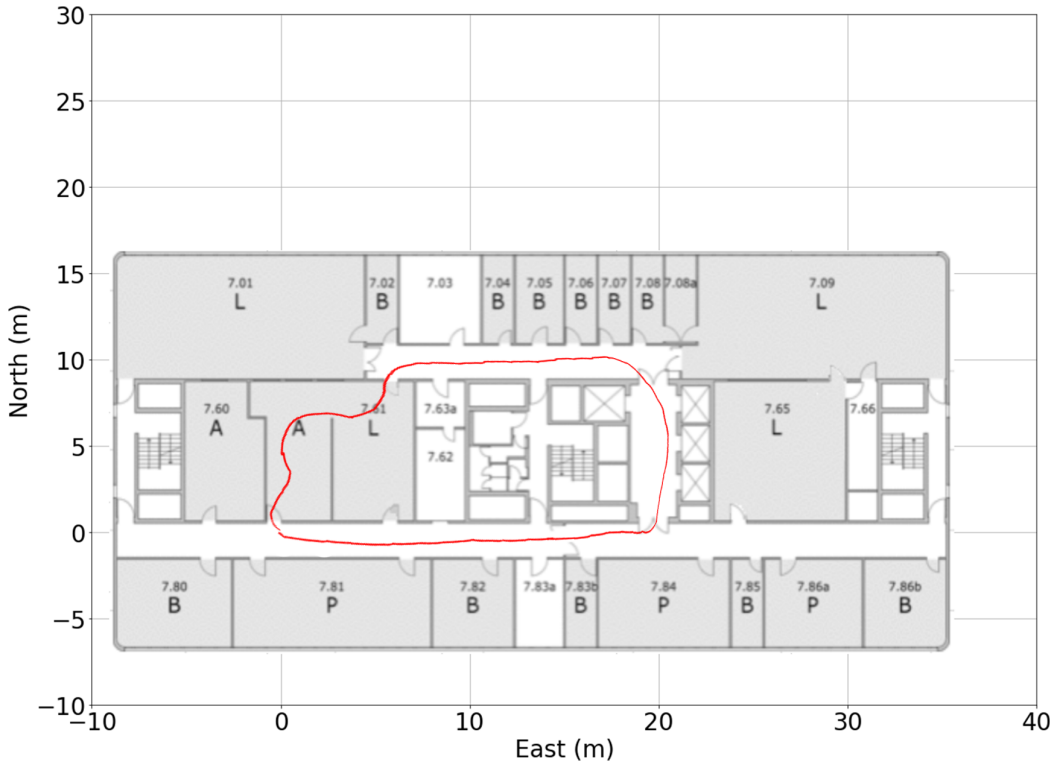


Abbildung 4.7.: Ground-Truth für Route 2

5. System Design

Eine zuverlässige Navigation hängt größtenteils von der Qualität der verfügbaren Sensordaten ab, weshalb inertiale Navigationssysteme nicht ohne geeignete Referenzsysteme auskommen. Gleichwohl existieren auch Systeme, die keine Korrekturmaßnahmen benötigen. Diese Lösungen erfahren dabei häufig eine individuelle Kalibrierung der Sensoren und werden den Systemen entsprechend angepasst, weshalb sie auch mit einem hohen technischen und finanziellen Aufwand verbunden sind. Das in dieser Arbeit entwickelte DL-Modell soll dazu beitragen auch kostengünstigere Geräte, die mit einem Low-cost-Inertialsensor ausgestattet sind, eine genaue und zuverlässige Navigation mit einem möglichst geringen Aufwand und unabhängig von der Umgebung, zu ermöglichen. Das vorliegende Kapitel befasst sich mit dem Systemmodell, präsentiert die einzelnen Komponenten, erläutert in diesem Zusammenhang ihre Aufgaben und verdeutlicht den Nutzen ihrer strategischen Kombination.

5.1. Ensemble-Deep-Learning-Modell

Das Ensemble Learning hat zum Ziel, mehrere Systeme bzw. Modelle zu einem Meta-Modell zu kombinieren, welches über eine bessere Verallgemeinerungsfähigkeit verfügt als jedes einzelne Modell. Bei der Systemidentifikation mittels DL-Techniken sind zeitveränderliche Daten, die ein dynamischen Verhalten aufweisen, nach wie vor eine große Herausforderung. Aus diesem Grund können Ensembles dementsprechend sehr nützlich für die Analyse komplexer Datensätze sein und zu einer verbesserten Performance führen.

Daher setzt sich das nachfolgend beschriebene hybride DL-Modell im Kern aus einem Ensemble aus den Convolutional Neural Networks und den Recurrent Neural Networks zusammen. Dass sich die Faltungsnetze nicht als eigenständiges Modell für multidimensionale Daten eignen, konnte bereits in der Arbeit [Meyer (2018a)] nachgewiesen werden. Dabei wurde festgestellt, dass sie große Schwierigkeiten aufweisen in den Daten die zeitliche Dynamik zu erfassen und Abhängigkeiten zu erkennen. Vielmehr haben sie ihre Stärken in der Merkmalextraktion und

können parallel auch zur Dimensionsreduktion verwendet werden. Des Weiteren zeigten die Arbeiten von [Chen u. a. (2018)] und [Cortés u. a. (2018)], wie auch in Kapitel 3 beschrieben, dass man das Problem der inhärenten Fehler, welches durch die MEMS-Inertialsensoren verursacht wird, mittels Convolutional Neural Networks nahezu vollständig beseitigen kann. Somit vermeidet man die Probleme mit dem Aufsummieren fehlerhafter Messwerte.
Zudem erwiesen sich die Faltungsnetzwerke bereits mehrmals als eine sehr effektive Methode in der Bildverarbeitung und sollen daraus resultierend in dieser Kombination zusätzlich die Aufgabe der Extraktion von Merkmalen und übergeordneten Merkmalen übernehmen.
Nach der Verarbeitung durch die Convolutional Networks findet im Anschluss eine Überführung der Daten in das Recurrent Network statt. Die wiederkehrenden Schichten ermöglichen die Erfassung der zeitlichen Dynamik aus den Inertialdaten. Um auch weitreichende temporale Abhängigkeiten in den zeitveränderlichen Daten zu erkennen ist es notwendig das RNN zusätzlich mit einer LSTM-Gedächtniszelle auszustatten. Das vollständige Modell kann der Abbildung 5.2 entnommen werden. Die jeweiligen Komponenten werden nachfolgend noch einmal näher erläutert.

5.2. Convolutional Neural Network

Für das Erlernen von Merkmalen und übergeordneten Merkmalen in den Inertialdaten sollen die Faltungsnetzwerke als Feature-Extraktoren dienen und abstrakte Darstellungen aus den Daten in Feature-Maps abbilden. Zudem sollen sie verwendet werden, um sowohl deterministische als auch zufällige Fehler in den Sensordaten größtenteils zu entfernen. Den Faltungsnetzen soll es dadurch gelingen die Fehlereigenschaften der Sensoren zu beseitigen bzw. diese zu kompensieren und die Daten auf diese Weise zu bereinigen. Die zuverlässige Überführung bereinigter Daten in das LSTM ist deshalb so relevant, weil sicherstellt werden muss, dass das Netz nicht aus fehlerbehafteten Messwerten lernt.
Um die Rohdaten des IMU-Sensors in das CNN einzuspeisen (mit den deterministischen und zufälligen Fehlern), müssen sie vorher wie in Kapitel 4 vorverarbeitet werden. Das CNN erhält als Eingabe einen Tensor mit den Abmessungen $[none,\ n\_steps,\ n\_input]$, wobei die erste Dimension der Größe des Batches entspricht, n_steps der Anzahl der Zeitschritte und n_inputs die Sensorkanäle beschreibt. Für die Erzeugung der Feature-Maps werden 2D-Filter (Kernel) auf die Sensordaten angewandt. Mit jeder weiteren Faltungsschicht werden weitere Filter der Sequenz unter Verwendung der **ELU** (engl. exponential linear unit) als Aktivie-

rungsfunktion gelernt. Ein anschließendes Pooling erfolgt dabei nicht. Als Grund für die Verwendung der ELU statt der ReLU als Aktivierungsfunktion kann mit „das Problem der toten ReLUs “ angeführt werden. Bei einem Input von negativen Werten liefern die ReLUs stets 0 - die Folge sind schließlich tote Neuronen im Layer und es kann kein Lernen mehr stattfinden.

Die Faltungsnetzwerke können zudem auch als zusätzlicher Vorverarbeitungsschritt gesehen werden. Da sie eben einen Großteil der Vorverarbeitung übernehmen, wie die Auswahl der Merkmale, die für das Modell maßgeblich sind. Außerdem sind sie, für die Integrität der Daten verantwortlich.

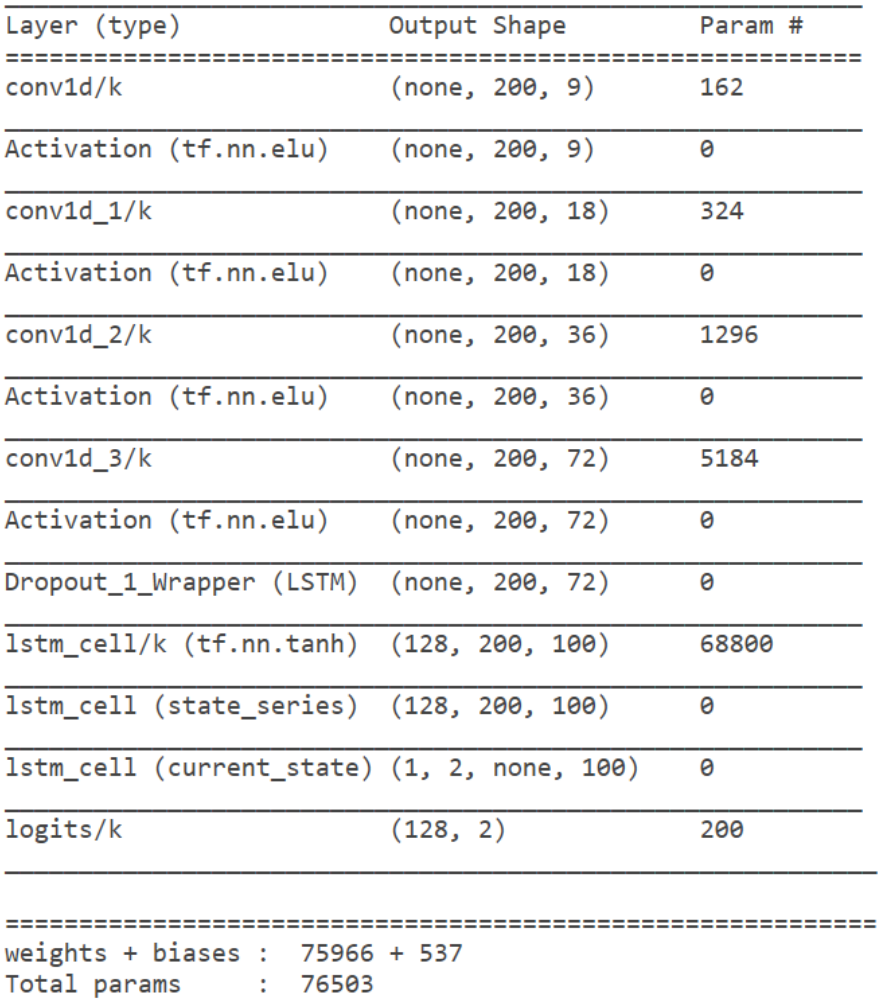
5.3. Recurrent Neural Network (LSTM)

Die Long-Short-Term-Memory-(LSTM) Netzwerke zeichnen sich insbesondere dadurch aus, dass sie auch weitreichende zeitliche Abhängigkeiten in Sequenzen erkennen können, ohne dabei den Einfluss von früheren Faktoren (auf spätere Berechnungen) zu verlieren. Aus diesem Grund liegt der Schwerpunkt dieser Komponente auf der Erkennung von temporalen Abhängigkeiten und Zusammenhängen in den Sequenzen bzw. zwischen den Merkmalen. Das LSTM-Netz ist insofern auch primär für die Berechnung der Positionen, aus denen die Trajektorie gewonnen wird, zuständig. Durch das Lernen stellt das LSTM die Abhängigkeiten zwischen den Sensorrohdaten und Koordinaten her und verfestigt die vorhandenen temporalen Strukturen dieser Sequenzen in den Gewichten.

Als Eingabewert erhält das LSTM die Matrix der Form $[none,\ n\_steps,\ n\_input]$ aus dem Faltungsnetzwerk. Nach der Verarbeitung der Daten durch das LSTM-Netz erhält man im letzten Zeitschritt (Timestep) am Ausgang einen Vektor der Größe mit der Anzahl der Recurrent-Neuronen im Recurrent-Layer. Die Ausgabe einer ganzen Recurrent-Schicht für einen Batch entspräche den Abmessungen $[none,\ n\_steps,\ n\_neuron]$, mit $none$ als Größe des Batches, n_steps entspricht dabei wieder der Anzahl der Zeitschritte und n_neuron der Anzahl der Neuronen in einem RN-Layer. Die Anzahl der Neuronen im RL ist auf 100 festgelegt, da sich diese Anzahl in vorhergegangenen Experimenten mit Sensordaten bewährt hat [Meyer (2018a), Meyer (2018b)]. Bevor neue *Positionsdaten/Orientierungsdaten* von dem Deep Neural Network (DNN) bestimmt werden können, erfolgt abschließend noch eine Überführung der produzierten Ausgabesequenzen der LSTM Berechnungen in die Softmax-Regressionsschicht.

Die Abbildung 5.1 veranschaulicht das Modell in Form einer textuellen Visualisierung und erlaubt einen detaillierten Einblick in die einzelnen Ebenen des Modells. Diese Art der Betrachtungsweise gibt Informationen über die Reihenfolge, die Ausgabeform sowie die Anzahl der trainierbaren Parameter in jeder Ebene und die Gesamtanzahl der verwendeten Parameter im Modell. Aus der Modellrepräsentation der Abb. 5.1 kann abgelesen werden, dass es sich um ein DNN mit 4 Convolutional Layer, einem LSTM-Layer mit 200 Timesteps und jeweils 100 Neuronen im RL handelt. Die folgenden Details sind dabei noch ergänzend zu erwähnen:

- **lstm cell/k**:
  Der RNN-Layer selber mit der LSTM-Zelle mit dem $\tanh$ als Aktivierungsfunktion (*k* für *Kernel*)
- **lstm cell/state series**:
  Zugriff auf die einzelnen Zeitschritte - entlang der Zeitachse aufgerollt
- **lstm cell/current state**:
  Zugriff auf den letzten Zeitschritt - repräsentiert den Endzustand der Zelle.
  Unterscheidet sich von den anderen durch die Form der Ausgabe. Dabei spiegelt sie lediglich die LSTM-Zelle wie in Kapitel 2.3.2 beschrieben wider. Die Abmessungen der Ausgabeform lassen sich wie folgt darstellen und beschreiben:
  $(1, 2, none, 100) \mapsto (n\_rnn\_layer, cell\ state/\ hidden\ state, batch\_size, n\_neuron)$



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d/k                     (none, 200, 9)            162
_________________________________________________________________
Activation (tf.nn.elu)       (none, 200, 9)            0
_________________________________________________________________
conv1d_1/k                   (none, 200, 18)           324
_________________________________________________________________
Activation (tf.nn.elu)       (none, 200, 18)           0
_________________________________________________________________
conv1d_2/k                   (none, 200, 36)           1296
_________________________________________________________________
Activation (tf.nn.elu)       (none, 200, 36)           0
_________________________________________________________________
conv1d_3/k                   (none, 200, 72)           5184
_________________________________________________________________
Activation (tf.nn.elu)       (none, 200, 72)           0
_________________________________________________________________
Dropout_1_Wrapper (LSTM)     (none, 200, 72)           0
_________________________________________________________________
lstm_cell/k (tf.nn.tanh)     (128, 200, 100)           68800
_________________________________________________________________
lstm_cell (state_series)     (128, 200, 100)           0
_________________________________________________________________
lstm_cell (current_state)    (1, 2, none, 100)         0
_________________________________________________________________
logits/k                     (128, 2)                  200
_________________________________________________________________

=================================================================
weights + biases :  75966 + 537
Total params     :  76503
```

Abbildung 5.1.: Visualisierung des Deep-Learning-Modells auf Parameter-Ebene mit 4 Convolutional Layer, 100 rekurrenten Neuronen, die entlang 200 Zeitschritten aufgerollt werden

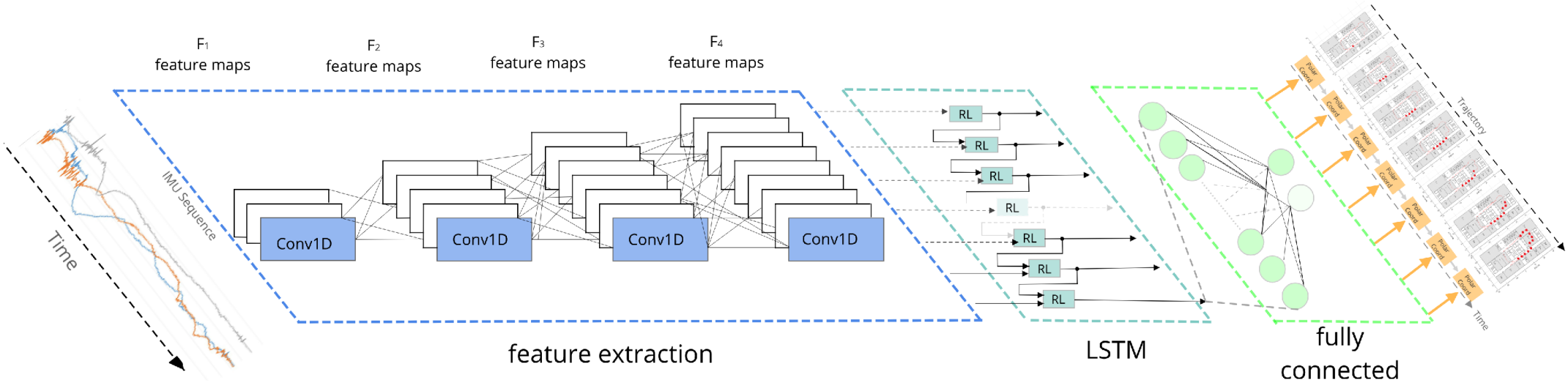


Abbildung 5.2.: Das Deep Learning Architekturmodell

6. Durchführung der Experimente

Das folgende Kapitel befasst sich mit dem Aufbau der Versuche und der Durchführung der Experimente. Dafür wird im nächsten Abschnitt zunächst die Simulationsumgebung mit der verwendeten Hard- und Software näher erläutert. Daraufhin folgen eine Beschreibung der Versuche und die Begründung für die Wahl der entsprechenden Trainingsparameter. Zudem wird auf die Vorgehensweise bei den Experimenten eingegangen. Abschließend findet eine Auswertung der jeweiligen Ergebnisse statt.

6.1. Systemkonfiguration

6.1.1. Software

Durchgeführt und implementiert wurden die Experimente mit TensorFlow als Deep Learning Framework und Python als Programmiersprache. Für die Durchführung von Datentransformationen und Vorverarbeitungen kamen die Frameworks Pandas und NumPy zum Einsatz. Für 2D- und 3D-Datenvisualisierung wurde sowohl auf Matlab als auch auf TensorBoard zurückgegriffen, ebenfalls wurde scikit-learn verwendet, weil es eine gute Unterstützung für viele weitere nützliche Funktionen für das Machine Learning bereitstellt. Der PDR-Algorithmus zur Ground-Truth-Erzeugung sowie Teile zur Aufbereitung der Daten, wie die Rauschreduzierung, wurden in Matlab umgesetzt.

6.1.2. Hardware

Die Grundlage der Arbeit basiert auf 3x leistungsstarken NVIDIA Tesla P100 mit 16 GB Arbeitsspeicher. Die Hardware sollte genug Performance aufbringen, um auch mit einer großen Anzahl und rechenintensiven Berechnungen die entsprechenden benötigten Ressourcen bereitstellen zu können und den Anforderungen zu entsprechen.
Für die Datenerfassung kamen die in Tabelle 6.1 aufgelisteten Sensoren zum Einsatz. Der IMU (9DoF) von SparkFun (siehe Abb. 6.1) wurde für die Ground-Truth-Erstellung eingesetzt, weshalb er zusätzlich einer Kalibrierung unterzogen werden musste. Bei 3-achsigen Sensoren ist

grundsätzlich eine Kombination aus Beschleunigungs-, Drehraten- und Magnetfeldsensoren gemeint, jeweils mit 3 Freiheitsgraden. Die 6-achsigen Sensoren werden als sogenannte *eCompass* angeboten und bestehen aus einer Kombination von 3-achsigen Beschleunigungssensoren und einem 3-achsigen Magnetfeldsensor. Eine weitere mögliche Ausführung ist ein Inertial Measurement Unit (IMU). Diese bestehen wiederum aus einem 3-achsigen Beschleunigungssensor und einem 3-achsigen Drehratensensor. Die 9-achsigen Sensoren (9DoF - Degree of Freedom) stellen die höchste Integration dar. Sie verfügen jeweils über einen 3-achsigen Beschleunigungssensor, einen 3-achsigen Drehratensensor und einen 3-achsigen Magnetfeldsensor.

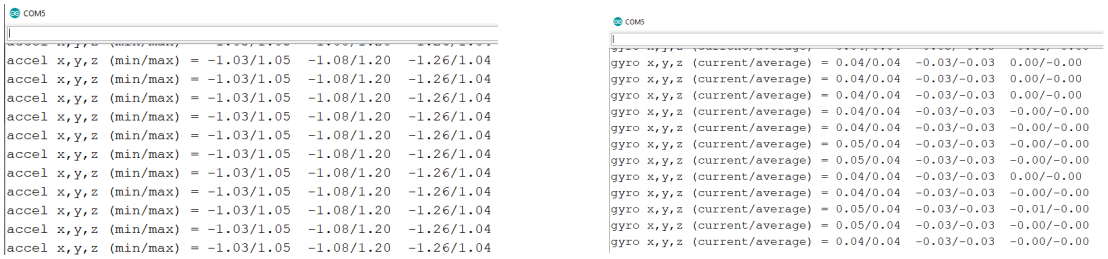
Bei dem IMU (9DoF) von SparkFun in Abb. 6.1 handelt es sich um einen 9-achsigen MEMS-Sensor (Abtastrate 100 Hz) von der Firma InvenSense inc. Der Sensor verfügt über einen MPU-9250 mit einem *System in Package (SiP)*. Das MPU-9250 ist ein Multi-Chip-Modul (MCM oder MCP für Multi Chip Package) und vereint ein 3-achsiges Gyroskop mit einem 3-achsigen Beschleunigungssensor und wird auch als *MotionTracking* Sensor bezeichnet. Das SiP kombiniert einen MPU-9250 mit einem 3-Achsen-Magnetometer (AK8963). Das MPU-9250 verwendet einen 16-Bit-Analog-Digital-Wandler (ADCs) um alle neun Achsen zu digitalisieren. Die *Sensitivity Scale Factor Tolerance* des Gyroskops liegt bei $25°C$ $\pm 3\%$, der *Noise*-Parameter beträgt $1 mdps/\sqrt{Hz}$. Beim Beschleunigungssensor beträgt der *Noise*-Parameter $300 \mu g/\sqrt{Hz}$. Das AK8963-Magnetometer weist einen (Wertebereich von –) *Full-Scale Range* $\pm 4800 \mu T$ und einen *Sensity Scale Factor* von $0.6 \mu T/LSB$ auf. Wie bereits erwähnt liefert die Verwendung eines zusätzlichen Magnetometers in der Navigation einen hohen Mehrwert. Die Kombination aus Beschleunigungssensor und Magnetometer kann zusätzliche Stützinformationen über die Lage liefern und daher zur Verbesserung der Genauigkeit beitragen. In Kapitel 2.2.4 wurden bereits

Abbildung 6.1.: Der 9DoF Razor IMU M0 ist eine Kombination, bestehend aus einem SAMD21-Mikroprozessor und einem 3-achsigen Low-cost-IMU-Sensor (MPU-9250 9DoF) [Electronics (2019)].

die möglichen Fehlereinflüsse von MEMS-Sensoren besprochen. Um die Sensorfehler daher weitestgehend klein zu halten und so auch eine möglichst fehlerfreie Ground-Truth-Erzeugung zu gewährleisten, wurden die Sensoren einer Kalibrierung unterzogen. Dafür wurde bei dem Beschleunigungssensor und dem Gyroskop die Bias-Instabilität geschätzt und entsprechend

Tabelle 6.1.: Verwendete Sensoren

Mobiles Gerät	IMU	Magnetometer
SparkFun IMU (9DoF)	InvenSense MPU9250	AK8963
iPhone 6	InvenSense MP67B	AKM8975
iPhone 5	-	-



(a) Beschleunigungssensors. (b) Gyroskop

Abbildung 6.2.: Ermittlung der Werte für die Kalibrierung.

kompensiert. Magnetometer wiederum weisen zwar ähnliche Fehler wie ein Beschleunigungssensor und Gyroskop auf, jedoch kann eine magnetische Umgebung des Sensors dazu führen, dass die wahren Messwerte überlagert werden. Aus diesem Grund kann die Kalibrierung des Magnetometers mit Umständen verbunden sein, denen eine besondere Bedeutung zugesprochen werden muss. Durch die Präsenz von lokal vorkommenden magnetischen Störfeldern (wie sie häufig bei einer Indoor-Navigation vorkommen) wird das Erdmagnetfeld überlagert, was die Messgenauigkeiten maßgeblich beeinflusst. Daher ist es wichtig die magnetische Umgebung des Sensors zu kontrollieren und ihn in der Umgebung zu kalibrieren, wo er eingesetzt werden soll. Die häufigsten Störfelder werden durch Hard- und Soft-Iron-Effekte verursacht. Sie entstehen durch die Überlagerungen unterschiedlicher magnetischer Felder. IMU-Kalibrierungen finden typischerweise durch Referenzmessungen statt. Dabei weisen diese Referenzmessungen oft eine weitaus höhere Genauigkeit auf, welche als wahre Messungen herangezogen werden, um die Sensoren danach auszurichten. Hersteller von Sensoren verwenden bspw. dafür sog. Kalibriertische. In [Titterton u. a. (2004)] wird ein solcher Vorgang mit einem Kalibriertisch beschrieben.

Tabelle 6.2.: Berechnete Werte für Bias und Skalierungsfaktor zur Kalibrierung der Achsen.

Achse	Bias	Skalierungsfaktor
x	0.235	1.265
y	0.03	1.06
z	-0.025	1.395

Kalibrieren des Beschleunigungssensors

Die Kalibrierung des Beschleunigungssensors erfolgt durch eine Rotationskalibrierung [Krohn u. a. (2005)]. Es lassen sich dadurch Nullpunktverschiebungen (Bias) und Steigungsfehler jeder Achse bestimmen. Dabei wird jede Achse einmal (*vorsichtig, langsam und immer um* $180°$) in Richtung der Erdanziehungskraft rotiert, sodass jede Achse mindestens einmal einer Beschleunigung von 1 g / -1 g (Erdbeschleunigung) ausgesetzt gewesen ist. Während der langsamen Rotation werden die Maximal- und Minimalwerte jeder Achse aufgezeichnet. Der Referenzwert für einen Maximalwert liegt bei 1 g, der Minimalwert entsprechend bei -1 g und Null wird erreicht, wenn sich die Achsen in der Horizontalen befinden. Die Abbildung 6.2a zeigt einen Ausschnitt des Kalibriervorgangs. Dabei werden die Minimalwerte und Maximalwerte aufgezeichnet. Die nachfolgende Gleichung erlaubt eine Kalibrierung des Beschleunigungssensors am Beispiel der x-Achse und bestimmt den Wert zur Korrektur für den Bias o_x und den Skalierungsfaktor s_x.

$$o_x = \frac{U_{max,x} + U_{min,x}}{2} \tag{6.1}$$

$$\tag{6.2}$$

$$s_x = \frac{U_{max,x} - U_{min,x}}{2} \tag{6.3}$$

Der Skalierungsfaktor korrigiert den Steigungsfehler der Kennlinie der jeweiligen Achsen, die Gl. 6.4 zeigt dies anhand der x-Achse. Die berechneten Werte sind in Tabelle 6.2 dargestellt.

$$accx_{scale} = \frac{accx_{aktuellerWert} - o_x}{s_x} \tag{6.4}$$

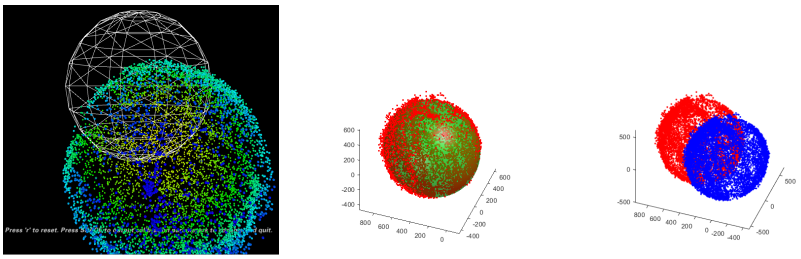
Tabelle 6.3.: Berechneter Offset jeder Achse in 0 $\frac{rad}{s}$.

Achse	Gyroskop
x	0.04
y	-0.03
z	0.00

Kalibrieren des Gyroskops

Wie der Beschleunigungssensor sind auch Drehratensensoren nicht frei von Messfehlern. Um die Abweichung des Gyroskops zu bestimmen, wird davon ausgegangen, dass es eine Abweichung von 0 $\frac{rad}{s}$ im Ruhezustand aufweist. Um spätere Drifts zu verringern, wird mit dem Gyroskop eine bestimmte Anzahl an Messungen durchgeführt, aus denen im Nachhinein der Mittelwert gezogen wird. Der berechnete Mittelwert wird als Bias abgespeichert und bei jeder Messung abgezogen. Der Sensor wird dafür lediglich für $30sec$ in den Ruhestand versetzt. Während dieses Zeitraums werden die Messwerte aufgezeichnet und der Mittelwert berechnet (siehe Abb. 6.2b). Die Tabelle 6.3 zeigt die so berechneten Bias-Werte.

Kalibrieren des Magnetometers



(a) (b) (c)

Abbildung 6.3.: Kalibriervorgang eines MEMS-Magnetometers zur Kompensation von Hard- und Soft-Iron-Effekte

Um für Anwendungen in der Navigation die Orientierung und Lage bestimmen zu können, kann es von großem Vorteil sein eine Kalibrierung der kostengünstigen MEMS-Magnetometern durchzuführen. Dabei geht es sowohl um die Kompensation von Störfeldern aus der Umgebung als auch die vom Hersteller selbst verursachten Fehler beim Herstellungsverfahren.
Wie im Kapitel 2 beschrieben, entstehen Hard-Iron-Effekte durch eine permanent magnetisierte Komponente der Gerätestruktur selbst. Sie induzieren dabei zusätzlich ihr eigenes magnetisches

Feld, was folglich zu Überlagerungen führt. Da Magnetometer häufig zusammen mit Mikrocontrollern und anderen Sensoren verbaut werden, entsteht eine unveränderliche Gerätestruktur, wodurch die Hard-Iron-Effekte ein zum Feld der Erde additives Magnetfeld erzeugen. Der Hard-Iron-Effekt zeigt sich somit als kontinuierlich wirkender Offset. Soft-Iron-Effekte entstehen hingegen durch Objekte in der Nähe des Sensors, die das umgebende Magnetfeld stören. Diese bewirken, dass die Kugel als ideales Maß gedehnt und gekippt wird. Die resultierenden Messungen liegen auf einem Ellipsoid. Die Durchführung der Kalibrierung sollte in der Umgebung umgesetzt werden, wo der jeweilige Sensor eingesetzt werden soll. Damit kann sichergestellt werden, dass die Soft-Iron-Fehler erkannt und entsprechend behoben werden können. Die Kalibrierung des Magnetometers zur Beseitigung von Hard- und Soft-Iron-Effekte erfolgte durch eine *Tilt-Compensated eCompas* oder auch Neigungskompensation [Ozyagcilar (2015)]. Das Institut für Softwaretechnik und theoretische Informatik: Quality and Usability Lab der TU Berlin bietet ein Tool zur Unterstützung einer Kalibrierung von Hard- und Soft-Iron-Effekten an [Berlin (2015)]. Dafür muss lediglich die Firmware des Sensors so angepasst werden, dass sie dem Tool eine Schnittstelle bereitstellt, wodurch das Tool die Magnetometerdaten in der Einheit mG empfangen kann.
Die Hard- und Soft-Iron-Effekte lassen sich im Anschluss bestimmen, indem man den Sensor so rotiert, dass die Erdkugel in Abb. 6.3a möglichst gleichmäßig mit den Markierungen abgedeckt wird. Unter idealen Umständen würden die Magnetometerdaten sich zu einer Erdkugel formen. Durch Störeinflüsse kommt es stattdessen zu Verschiebungen und Deformierungen der Kugel. Das Tool bestimmt aus diesen Daten die Parameter sowohl für die Hard-Iron- als auch die Soft-Iron-Kompensation. Die berechneten Parameter für die Kalibrierung können im Anschluss mit Matlab betrachtet und analysiert werden. Die Ergebnisse der Abbildungen 6.3 präsentieren die Hard- und Soft-Iron-Kompensation. Aus der Abbildung 6.3a geht bereits eine starke Verschiebung hervor, woraus ein Hard-Iron-Fehler abgeleitet werden kann. Die Abbildung 6.3b präsentiert die Originaldaten als in rot aufgetragene Markierungen. Dabei ist ersichtlich, dass sich die Markierungen zu einer Kugel geformt haben. Wären Soft-Iron-Effekte vorhanden, würde sich der Kreis zu einer Ellipse verzerren. Da aus der Abbildung 6.3b kein Ellipsoid erkennbar ist, geht stattdessen aus den Abbildungen 6.3a und 6.3c hervor, dass ein Hard-Iron-Effekt vorhanden ist, welchen es zu kompensieren gilt. Die berechnete Korrektur ist in Abbildung 6.3c dargestellt. Die in rot geformte Kugel spiegelt die originalen Magnetometer Daten wieder, die blauen Markierungen entsprechen dagegen dem kompensierten Wert.

6.2. Versuchsaufbau

Nachdem die Bereitstellung der Hardware sowie die Implementierung des DL-Modells abgeschlossen waren, wurde eine Reihe von Trainingsversuchen aufgestellt, mit dem Ziel, das Verhalten und die Auswirkungen unterschiedlicher Parameter auf das DL-Modell mit dem IMU-Datensatz genauer zu untersuchen. Außerdem soll das DL-Modell auf seine Eigenschaften und Anwendbarkeit in der Praxis evaluiert werden. Durch die Ermittlung im Parameterraum des Modells soll die optimale Kombination an Trainingsparametern festgestellt werden. Neuronale Netze bieten einen großen Spielraum, wenn es darum geht die optimale Kombination von Hyperparametern für die zu lösende Aufgabe zu finden und entsprechend einzustellen. Diese Flexibilität an einzustellenden Hyperparametern kann somit auch zu einem ihrer Hauptnachteile werden. Dennoch ist die Feineinstellung der Hyperparameter ein wichtiger Teil bei der Entwicklung eines Deep-Learning-Systems.
Die jeweiligen aufgestellten und zu untersuchenden Experimente mit der Vorstellung des im Vorhinein festgelegten Suchraums der Parameter für die entsprechenden Bewegungsformen repräsentiert die Tabelle 6.4 für *foot-mounted* und 6.5 für das *Handheld*-Szenario. Nachfolgend wird kurz auf die jeweiligen Trainingsversuche der Tabellen 6.4 und 6.5 eingegangen. Der Versuchsaufbau sieht dabei vor, dass die Parameterkombination mit der höchsten Genauigkeit bzw. in dieser Arbeit mit der kleinsten Fehlerabweichung in den nächsten Versuch übernommen wird. Dadurch entwickelt sich eine schrittweise Anpassung der Hyperparameter, bis sich zum Schluss die optimale Kombination aus Parametern herauskristallisiert.

- **Versuch 1**: Das Modell wird mit unterschiedlichen Mini-Batch Größen trainiert. Im Anschluss findet die erste Evaluation in Form einer ersten Feinabstimmung der Hyperparameter statt. Die ersten Erkenntnisse geben bereits Auskunft über das Verhalten des Modells und lassen daher eine erste Optimierung bzgl. der Generalisierungsfähigkeit zu. In Abhängigkeit von der Mini-Batch Größe wird die Lernrate gewählt. Im nachfolgenden Kapitel 6.3 wird näher darauf eingegangen.

- **Versuch 2**: Dieser Versuch befasst sich mit der Untersuchung der Timesteps in einem Zeitfenster zwischen $0.5 - 4$ Sekunden. Es soll die Anzahl der benötigten Zeitschritte bestimmt werden, die eine Verfestigung der temporalen Strukturen der Sequenz zulassen, wodurch eine Positionsbestimmung der entsprechenden Bewegungsform erlaubt wird.

- **Versuch 3**: Eine Untersuchung auf die bezüglich der benötigten Anzahl an Convolutional Layern soll zum einen Aufschluss über nicht deterministische Anteile und die Qualität der Sensordaten geben und zum anderen darüber, wie viele Layer mindestens notwendig

sind, um die erforderliche Menge an Merkmalen für eine zuverlässige Navigation aus den Daten zu extrahieren. Zusätzlich wird bei der Bewegungsform *handheld* ein weiterer RNN-Layer hinzugenommen. Das Szenario *foot-mounted* erreichte bei einem weiteren RNN-Layer die Grenzen der zur Verfügung stehenden Ressourcen.

- **Versuch 4**: Die Ermittlung und Überführung der bis hier erfolgversprechendsten Kombination an Metaparametern wird in einem finalen Training auf weitere Low-cost-Inertialsensoren angewendet und mit den dazugehörigen Datensätzen getestet und im Anschluss evaluiert. Die Tests wurden mit mehreren Teilnehmern und Geräten auf 2 Routen durchgeführt.

- **Versuch 5**: Dieser Versuch soll überprüfen, inwieweit sich die Ergebnisse bzw. der trainierten DL-Modelle auf andere Strecken übertragen lassen. Dafür wurde das Szenario der Bewegungsform *foot-mounted* herangezogen.

Aus den Tabellen 6.4 und 6.5 geht ebenfalls hervor, dass die Versuche 1 bis 3 nur an der Route 1 durchgeführt worden sind. Lediglich bei dem finalen Training in Versuch 4 und bei der Überprüfung der trainierten Modelle auf die Eignung der Anwendbarkeit auf andere mögliche Routen in Versuch 5 wurden beide Strecken eingesetzt.
Des Weiteren wurde in Versuch 1 darauf geachtet, dass trotz der unterschiedlichen Mini-Batch-Größen die Anzahl der Iterationen übereinstimmt. Die Batch-Size definiert die Anzahl an Trainingsbeispielen die pro Update trainiert werden. So erhält man ein gutes Maß, um die verschiedenen Batch-Größen direkt miteinander vergleichen zu können. Der vollständige Durchlauf eines Trainingsdatensatzes wird als *Epoche* bezeichnet und wurde ebenfalls an die Batch-Größe angepasst. Das Szenario *Handheld* benötigte dabei eine wesentlich größere Anzahl an Iterationen, um überhaupt ein Ergebnis produzieren zu können.

Aus der Tabelle 6.4, Spalte „Device/Größe Datensatz“ geht hervor, dass für die Versuche 1 bis 3 lediglich der IMU-Sensor zum Einsatz kam. Der Grund ist, dass eine Anwendung auf den vollständigen Datensatz (die Datensätze der anderen Sensoren hinzu gerechnet), die Simulations- bzw. Trainingszeit erheblich ansteigen lassen würde. Darum wurde für die Feinabstimmung der Hyperparameter nur der Datensatz des IMU-Sensors eingesetzt. Zudem kann aus dieser Spalte auch die Größe des verwendeten Datensatzes für den entsprechenden Versuch abgelesen werden. Sie ist angegeben als eine Matrix der Größe $(n\_samples, n\_channel)$, wobei $n\_samples$ der Anzahl der Trainingsbeispiele pro $n\_channel$ entspricht und $n\_channel$ der Anzahl der verwendeten Sensorkanäle. Damit enthält der Datensatz eine Größe $(n\_samples \times n\_channel)$. Für die Route 1 mit den Abmessungen von $(8000, 9)$ bedeutet das eine Anzahl von 10 Sequen-

zen/Strecken mit jeweils 8000 Sensordaten und 9 Sensorkanälen. Daraus kann eine Gesamtgröße von 80000×9 schlussgefolgert werden.
Für die Route 2 bedeuten die Abmessungen $(76500, 9)$ eine Gesamtgröße von 76500×9 mit 9 Sequenzen/Strecken, die jeweils 8500 Sensordaten beinhalten und ebenfalls 9 Sensorkanäle besitzen. Bezüglich der Route 2 in dem *Foot-mounted*-Szenario ist auffällig, dass nur 9 Sequenzen verwendet worden sind. Das liegt vor allem daran, dass für diese Route nur maximal 9 Sequenzen für das iphone 5 zur Verfügung standen. Eine ungleiche Verteilung bei der Verwendung der Trainingsdatensätze läuft Gefahr, dass die Sensordaten nicht optimal und fair erlernt und getestet werden können. Das hat zur Folge, dass sich während des Trainings die Merkmale und Eigenschaften der anderen Sensoren stärker in den Gewichten verfestigen können, was folglich fehlerhafte Ergebnisse verursacht.

Tabelle 6.4.: Trainingsszenario **foot-mounted**: Versuchsaufbau und Parameterwahl.

Route 1					
Versuch	Ziel / Untersuchung	Device / Größe Datensatz	Batch-Size/ Lernrate	Epochen	Iterationen
1	Batch-Sizes	Sensor/$(80000, 9)$	128/1e-04 600/1e-03 8000/1e-03	120 450 6000	60000 60000 60000
2	Timesteps (50,100,200,300,400)	Sensor/$(80000, 9)$	8000/1e-03	6000	60000
3	CNN Layer (0,1,2,3,4)	Sensor/$(80000, 9)$	8000/1e-03	6000	60000
4 5	Finales Training[1]	Sensor/$(80000, 9)$ iphone5/$(80000, 9)$ iphone6/$(80000, 9)$ Sensor iphone5 iphone6	8000/1e-03	6000	180000
Route 2					
4 5	Finales Training[1] Überprüfung auf die Validität anderer Strecken.	Sensor/$(76500, 9)$ iphone5/$(76500, 9)$ iphone6/$(76500, 9)$ Sensor iphone5 iphone6	8000/1e-03	6000	174000

6.3. Auswahl der Metaparameter und Modelltraining

Dieses Kapitel widmet sich der Beschreibung und Auswahl der spezifischen Metaparameter bzw. Hyperparameter sowie dem Modelltraining. Die zu untersuchenden und einstellbaren Parameter sind nach ihrem Relevanz und Vorkommen in den Versuchen sortiert.

Unterschiedliche Batch-Größen

Moderne Deep-Learning-Architekturen werden immer häufiger mit dem *Mini-Batch-Gradientenverfahren* anstatt dem *Stochastischen Gradientenverfahren* trainiert. Das mag zum einen an dem großen

[1] Validieren und Testen durch Überführung der bisherig gelieferten Ergebnisse auf weitere Sensoren.

Tabelle 6.5.: Trainingsszenario **handheld**: Versuchsaufbau und Parameterwahl.

Route 1					
Versuch	Ziel/ Untersuchungen	Device/ Größe Datensatz	Batch-Size/ Lernrate	Epochen	Iterationen
1	Batch-Sizes	Sensor/$(85000, 9)$	128/1e-04 600/1e-03 8500/1e-03	200 900 12500	125000 125000 125000
2	Timesteps (50,100,200,300,400)	Sensor/$(85000, 9)$	128/1e-04	200	125000
3	CNN Layer (0,1,2,3,4) RNN Layer (2)	Sensor/$(85000, 9)$	128/1e-04	200	125000
4	Finales Training[1]	Sensor/$(85000, 9)$ iphone5/$(85000, 9)$ iphone6/$(85000, 9)$	128/1e-04	300	597600
Route 2					
4	Finales Training[1]	Sensor/$(85000, 9)$ iphone5/$(85000, 9)$ iphone6/$(85000, 9)$	128/1e-04	300	597600

Vorteil des Parallelisierens von Operationen auf dafür optimierter Hardware zu tun haben. Denn dies führt dazu, dass der Speicherbedarf erheblich reduziert und gleichzeitig eine Steigerung der Performance erreicht werden kann. Für eine zuverlässige und gute Verallgemeinerungsfähigkeit hat sich dazu eine Mini-Batch-Größe zwischen 2 und 32 als optimal erwiesen [LeCun u. a. (1998), Keskar u. a. (2016)]. Ein weiterer wichtiger Faktor neben der Größe der Batch-Size die Lernrate. In [Masters und Luschi (2018)] wird empfohlen, bei der Wahl eines vergleichbaren großen Mini-Batches auch eine entsprechend große Lernrate zu wählen. Möchte man stattdessen sicherstellen, dass die best-mögliche Leistung mit dem Modell erzielt werden soll, sollte die Verwendung einer kleinen Lernrate mit einer kleinen Mini-Batch-Größe gewählt werden. Der Grund für eine kleine Lernrate bei einer kleinen Batch-Size hängt mit der Varianz der Gradientenberechnung zusammen. Aufgrund dieser Varianz in der Gradientenberechnung ist keine Stabilität mehr gegeben, was folglich den Trainingsprozess erheblich in die Länge ziehen würde – zum einen wegen der Lernrate und zum anderen wegen der Menge an Iterationen die zusätzlich notwendig sind, um die Trainingsdaten entsprechend anzupassen.

Eine zu große Lernrate kann dazu führen, dass das lokale Minimum nicht gefunden wird, wohingegen eine zu kleine Lernrate möglicherweise bewirken kann, dass die Fehlerfunktion des Trainingsalgorithmus das lokales Minimum nicht mehr verlassen kann. Daher wird immer öfter auf Varianten mit verschiedenen Lernraten oder dynamisch implementierten Lernraten zurückgegriffen. In [Smith u. a. (2017)] konnte dagegen jedoch erfolgreich gezeigt werden, dass der Erfolg des Trainingsalgorithmus sowie die Dauer der Trainingszeit nicht nur durch die Lernrate gesteuert werden kann, sondern auch von der Größe des Mini-Batches abhängt. Bei einer unveränderten Lernrate wurde daher während das Trainings die Größe des Mini-Batches erhöht. Dabei gab es keine nennenswerten Abweichungen auf Kosten der Genauigkeit. Stattdessen konnte eine Verringerung der Trainingszeit von über 50% verzeichnet werden. Begründet wurde dieser Erfolg unter anderem damit, dass das Problem auch mit weniger Updates (Gewichtsaktualisierungen) hätte erreicht werden können.Die Wahl eines kleinen Mini-Batches kann außerdem zu einen Regulierungseffekt führen, da kleine Batch Größen dem Lernprozess möglicherweise ein sog. Rauschen hinzufügen [Wilson und Martinez (2003)]. Der geringste Generalisierungsfehler ergibt sich dabei bei einer Batch-Größe von 1.
Eine wichtige Erkenntnis aus diesen Arbeiten ist, dass neben der Lernrate und der Batch-Size auch die Anzahl der Parameterupdates eine große Rolle spielen kann. Wählt man die richtige Strategie bei einem entsprechenden Problem, kann eine wesentlich höhere Genauigkeit bei der Modellleistung, zugunsten eines deutlich kürzeren Trainingsprozesses erzielt werden.

Da die Komplexität in dieser Arbeit aufgrund mangelnder verwandter Arbeiten nur schwer einzuschätzen ist, findet eine Untersuchung auf verschiedenen Größen von Mini-Batches statt. Dadurch können nicht nur neue wertvolle Erkenntnisse gesammelt werden, auch die Wahrscheinlichkeit guter Resultate ist höher.

Anzahl der Timesteps

Ein weiterer wichtiger und zu untersuchender Parameter liegt in der Anzahl der Timesteps des LSTM-Netzes. Die Anzahl der Timesteps wird durch das zu lösende Problem festgelegt. Durch ihre Größe bestimmen sie den Zeitraum einer Sequenz, in der temporale Abhängigkeiten erkannt werden. Dabei ist es wichtig, die Timesteps entsprechend groß zu wählen, sodass sich die Bewegungsformen bzw. das zu erlernende Ziel in den Timesteps widerspiegeln kann. Die in dieser Arbeit zu untersuchenden Größen sind dabei $50, 100, 200, 300, 400$ Zeitschritte. Das entspricht einem Zeitfenster zwischen $0.5 - 4$ Sekunden und ist aufgrund der verfügbaren Ressourcen auch auf diese beschränkt. Dennoch sollte die Größe der Timesteps ausreichen, um eine Bestimmung der Position in Abhängigkeit der Bewegungsformen zu ermöglichen, um so erste Erkenntnisse in der Positionsbestimmung mittels Deep-Learning zu gewinnen.

Anzahl Convolutional Layer

Über die Verwendung des Convolutional Networks wurde bereits in Kapitel 5 ausführlich gesprochen. In der Arbeit [Meyer (2018a)] wurde mit einem identischen System Design eine Aktivitätserkennung erfolgreich umgesetzt. Dabei konnte herausgefunden werden, dass eine höhere Anzahl an Convolutional Layern sich zugunsten der Erkennungsgenauigkeit im DL-Modell auswirken kann. Dabei konnten ebenfalls auf Basis von Sensordaten zuverlässige Ergebnisse $> 90\%$ mit einer maximalen Anzahl von 4 Convolutional Layer verzeichnet werden. In [Chen u. a. (2018)] wurden 5 Convolutional Layer verwendet, um in Sensordaten die nicht deterministischen Anteile, die als Störgrößen erfasst wurden, zu entfernen. In dieser Arbeit wird die maximale Anzahl auf bis zu 4 Convolutional Layer festgelegt.

Weitere Parameter wie die Anzahl der Neuronen im Recurrent Layer wurden auf 100 festgelegt. Bei der Wahl der Anzahl der verwendeten Neuronen wird grundsätzlich ein empirischer Ansatz offeriert. In [Géron (2017)] empfiehlt und beschreibt Aurélien Géron, dass bei der Wahl darauf zu achten ist, dass das neuronale Netz (NN) einen trichterförmigen Aufbau aufweisen sollte. Um prinzipiell die Wahl der Hyperparamter im NN klein zu halten, sollten die Neuronen daher im Vorhinein auf eine feste Anzahl begrenzt werden. Im Allgemeinen entwickelt das Erhöhen

der verborgenen Schichten mehr Durchschlagskraft als das permanente Erhöhen der Neuronen in den Schichten. Eine Erhöhung der Neuronen würde nur angestrebt werden, wenn das Netz Gefahr laufen sollte zu overfitten. Da die Convolutional Networks bereits die Aufgabe der Erfassung der komplexeren Merkmale übernehmen und sie in ihren Feature-Maps abbildet, werden die Neuronen im RL auf 100 festgelegt.

Als Aktivierungsfunktion wurden, wie bereits in Abschnitt 5 erläutert, für die Faltungsnetze die **ELU**- und für die LSTM-Netze die **tanh**-Funktion verwendet. Obwohl bei der naheliegendsten und momentan populärsten Lösung bei der Wahl der Aktivierungsfunktion schnell auf die **ReLU** zurückgegriffen wird, weil sie effektiv zur Vermeidung des exponentiellen Fehlerschwunds beiträgt, spielt sie in dieser Arbeit keine Rolle (vgl. [Géron (2017)]). Ihr nachteiliger Effekt ist ihr konstanter 0-Wert im negativen Definitionsbereich. Das Problem ist auch unter dem Namen „*sterbende ReLUs*“ bekannt. Dabei sterben einige Neuronen während des Trainings praktisch ab, d. h., sie geben nichts anderes als 0 aus. Das kann dahin führen, dass die Hälfte der Neuronen im Netzwerk tot ist und es ist unwahrscheinlich, dass die entsprechende Neuronen im Nachgang wieder zum Leben erweckt werden, weil der Gradient für negative Eingaben eben 0 beträgt. Da in dieser Arbeit Sensorwerte als Trainingsdatensatz verwendet werden, deren Definitionsbereich weit unter 0 liegt, wird der ReLU in dieser Arbeit keine Bedeutung zugesprochen. Zur Beurteilung der Güte des Modells wird die Wurzel der mittleren quadratischen Abweichung oder auch der Root Mean Square Error (RMSE) angewendet. Sie entspricht der Größe des Fehlers, den das System im Mittel bei der Positions- und Orientierungsbestimmung macht. Im Bezug auf die Positionsbestimmung wird der Abstand zwischen zwei Positionen bestimmt. Da die Wurzel einer quadratischen Summe gleichzeitig auch dem *Euklidischen Abstand* entspricht, erhält man bei der Position den direkten Abstand in Meter m bzw. die Differenz zwischen der vom Modell geschätzten Position und der Ground-Truth. Im Fall der Bewegungsform *handheld* erhält man die Differenz der vom Modell erzeugten Orientierung und der Ground-Truth. Die Heading Daten, welche dem Yaw-Winkel entsprechen, müssen erst mithilfe der Rho-Theta Technik aus Gl. 4.10-4.11 in Kapitel 4.3 in eine Trajektorie umgerechnet werden, bevor diese im Anschluss mit der Gleichung für den Euklidischen Abstand berechnet und mit anderen Trajektorien verglichen werden können. Dadurch ist ein ordentlicher Standard für ein Qualitätsmaß geschaffen.
Die Optimierung der Parameter wird durch die Minimierung der Verlustfunktion (engl. loss function) mit Hilfe des Mini-Batch-Gradientenverfahrens und der Adam-Update-Regel [Kingma und Ba (2014)] durchgeführt. Um das neuronale Netz vor Overfitting zu schützen, wurden die LSTM-Layer zusätzlich mit einer Dropout-Schicht ausgestattet (mit einer Dropout-

Wahrscheinlichkeit von 25%). Zudem wird beim Training auf die weit verbreitete Technik der Kreuzvalidierung (engl. cross-validation) zurückgegriffen, um sicherzustellen, dass das Modell die Daten gut verallgemeinert. Bei der cross-validation wird der Testdatensatz so lange zurückgehalten, bis die Qualität des Modells zuversichtlich eingeschätzt werden kann.

Zum Schluss wurde in Abhängigkeit von den Batch-Größen die internen Zustände des LSTM-Netzes gesteuert. Ein Reset der internen Zustände von *cell state* und *hidden state* fand jeweils nach einer Sequenz statt. Eine Sequenz entsprach eine vollständig gelaufenen Strecke und beinhaltete immer eine ungefähre Menge von 8000×9 Sensordaten, abhängig von der Strecke und der Bewegungsform. Mit dem Ziel eine temporale Abhängigkeit in einer Sequenz zu erkennen, sowohl zur Bestimmung einer Position als auch zur Bestimmung der Orientierung, wurde der interne Zustand des LSTM-Netzes als *stateless* oder *stateful* konfiguriert.

Für die Mini-Batch-Größen von 128 und 600 wurde eine *stateful* LSTM-Konfiguration gewählt. Der letzte berechnete interne Zustand wird als Initialisierungszustand für das nächste Trainingsbeispiel des des nachfolgenden Batches verwendet. Nach Empfehlungen von [Brownlee (2019)] findet ein Reset der Zustände nach jeder Epoche statt. Damit bleibt das Finden von Abhängigkeiten auf die jeweiligen Sequenzen begrenzt.
Für die Mini-Batch-Größe $8000/8500$ wurde *stateless* gewählt. Die internen Zustände der LSTM Zelle erfahren nach jedem Batch und nach jeder Epoche einen Reset. Der nachfolgende interne Zustand des neuen Trainingsbeispiels des Batches erhält als Initialisierungszustand einen zero-state. Damit bleiben ebenfalls die Abhängigkeiten bei einer Batch-Größe, die einer Strecke entsprechen, auch auf eine Strecke begrenzt.
Ein Durchmischen (engl. shuffel) der Trainingsdaten findet nicht statt, da die Reihenfolge, in der sich die Trainingsdaten befinden, wichtig für das Erkennen von Abhängigkeiten ist.

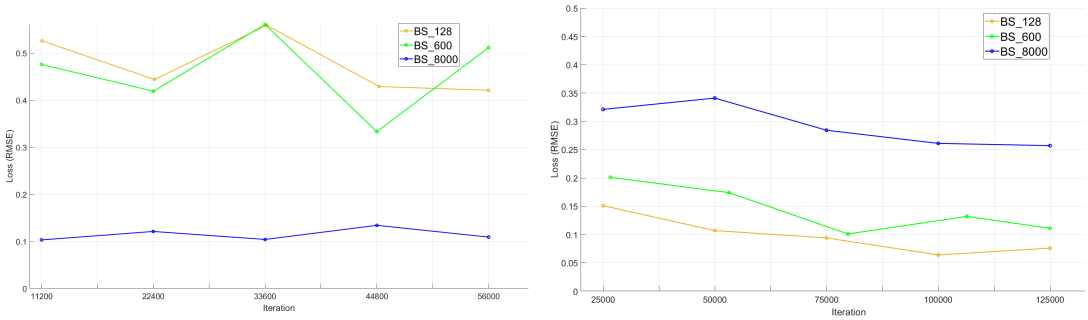
6.4. Ergebnisse

Dieses Kapitel befasst sich ausschließlich mit der Vorstellung und Beschreibung der Ergebnisse. Um die Güte des DL-Modells beurteilen zu können, wird die Lossfunktion verwendet. Sie misst den Fehler der berechneten Position und der tatsächlichen Position (Ground-Truth). Anhand dieser Verlustfunktion kann abgelesen werden, wie gut ein trainiertes Modell auf den Datensatz verallgemeinern konnte und mit welcher Parameterkombination die höchste Leistungsfähigkeit erzielt wurde. Wie im Kapitel 6.3 bereits erwähnt, entspricht die Verlustfunktion dem RMSE. Da die Wurzel einer quadratischen Summe gleichzeitig auch dem *Euklidischen Abstand* entspricht, erhält man die Differenz der geschätzten Position vom Modell und der Ground-Truth in m. In jedem Versuch werden die Ergebnisse beider Bewegungsformen gemeinsam dargestellt, umso einen adäquaten Vergleich herstellen zu können.
Für den Anfang wird mit einer Parameterkombination aus: 200 Timesteps (TS) und 4 Convolutional Layern begonnen, welche im Laufe der Experimente schrittweise angepasst werden.

6.4.1. Versuch 1

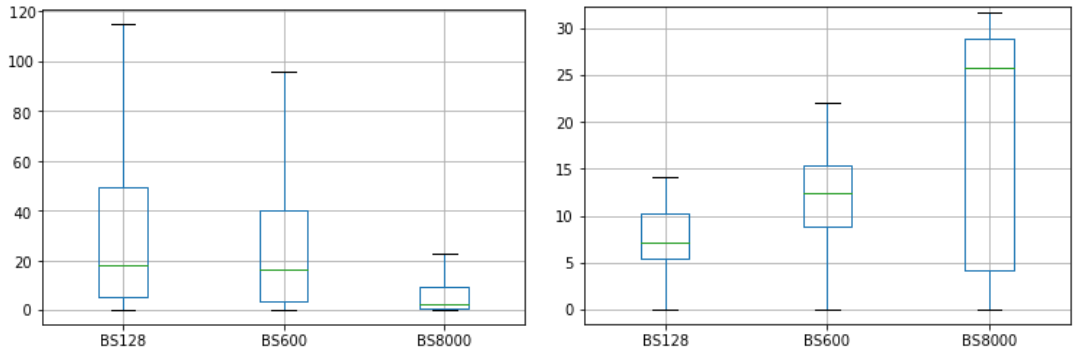
Der Versuch 1 sieht eine Untersuchung auf unterschiedliche Größen des Mini-Batches vor. Die Batch-Größen waren dabei **128, 600**, und **8000**. In beiden Bewegungsformen in Abb. 6.4 lassen sich klare Unterschiede ausmachen. So verallgemeinert das DL-Modell die Bewegungsform *foot-mounted (fm)* bei großen Mini-Batches sehr gut, wohingegen die Bewegungsform *handheld (hh)* bei kleinen Mini-Batches eine gute Verallgemeinerungsfähigkeit demonstriert. Die Abbildungen in 6.4 zeigt die Fehlerabweichungen beider Bewegungsformen der DL-Modelle während des Trainings über die Anzahl der Iterationen hinweg. Dabei erreicht das Szenario *foot-mounted* mit einer Batch-Größe von 8000 ein identisches Ergebnis wie das Szenario *handheld* mit einer Batch-Größe von 128. Die kleinste Fehlerabweichung für *foot-mounted* beträgt $10.9\ m$ bei Iterationsschritt 11200, die kleinen Batch-Größen (128, 600) erreichen mit einer durchschnittlichen Abweichungen von $45\ m$, was eine Differenz von 24.2% entspricht, einen Vergleichsweisen hohen Fehler. Der kleinste Fehler für *handheld* beträgt $7.4\ m$ bei BS 128 und Iterationsschritt 100000. Der kleinste Fehler für BS 8000 hingegen beträgt $29.5\ m$, was eine Differenz von ca. 25% entspricht. Außerdem ist bei *fm* auffällig, dass die beste Leistung bereits nach kurzer Zeit erreicht worden ist und auch nach weiteren Iterationen keine großen Veränderungen mehr erkennbar sind. Dabei ist zusätzlich auf den Unterschied in der Menge der benötigten Iterationen beider Bewegungsformen zu achten.
Der nachfolgende box- und whisker-plot in Abbildung 6.5 erlaubt nochmals einen andere Betrachtungsweise auf die Ergebnisse. So wird schnell deutlich, dass die Ergebnisse aus Abbildung

6.4 sich mit den Ergebnissen in Abbildung 6.5 decken. Die Streubreite der Fehlerabweichung der Bewegungsform *fm* für eine Batch-Größe von 8000 liegt in einem Bereich zwischen $0 - 22\ m$. Der Median ist kaum zu erkennen, da er sich sehr weit unten nahe der $2\ m$ ansiedelt. Das bedeutet, dass sich 50% der Fehler unter einem Wert von $2\ m$ befinden. Die Werte für die Batch-Größen 128 und 600 für *fm* liegen hingegen weitaus höher und lassen sich von der Batch-Size (BS) 8000 klar abgrenzen. Für die Bewegungsform *handheld* in Abb. 6.5b decken sich die Ergebnisse ebenfalls mit denen aus Abb. 6.4b. Die Fehlerabweichung für BS 128 bewegt sich in einem Bereich zwischen $0 - 15\ m$. Die Hälfte davon befindet sich unter $7\ m$. Die anderen Batch-Größen erreichen dagegen weitaus höhere Werte und lassen sich klar abgrenzen.



(a) Foot-Mounted (b) Handheld

Abbildung 6.4.: Versuch 1: Analyse unterschiedlicher Mini-Batch-Größen



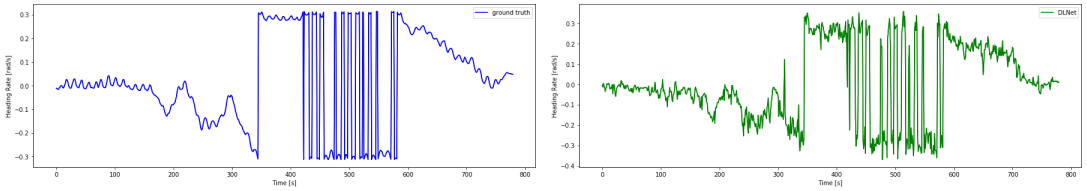
(a) Versuch 1: Box-und Whisker-plot für fm (b) Versuch 1: Box-und Whisker-plot für hh

Abbildung 6.5.: Versuch 1: Lage und Streubreite der Verteilung des Fehlers anhand Box-und Whisker-plot – Einheit in m

Die nachfolgenden Abbildungen beinhalten bereits die Parameterkombinationen die sich als die erfolgversprechendsten erwiesen haben:

- Foot-Mounted: Eine Batch-Größe von 8000, 200 Timesteps und 4 Convolutional Layer
- Handheld: Eine Batch-Größe von 128, 200 Timesteps und 4 Convolutional Layer.

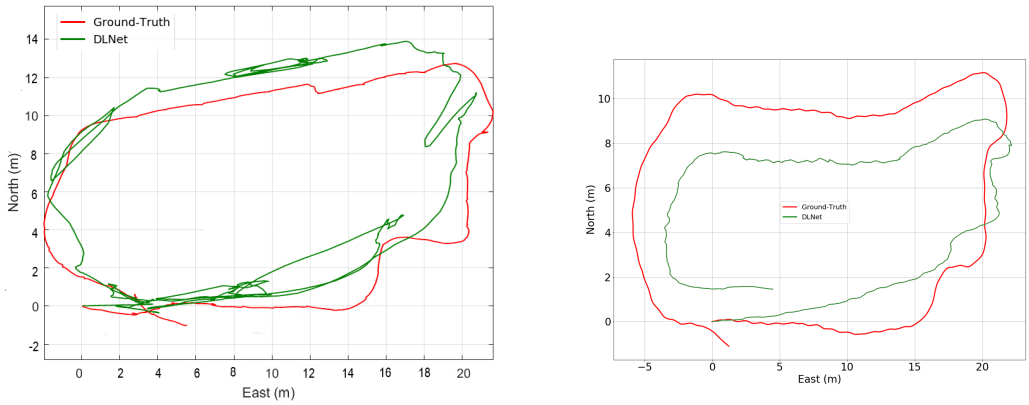
Die Darstellungen in 6.6 demonstrieren die Heading-Daten für das Szenario *handheld*, wodurch sich die Trajektorie erzeugen ließ. Die Abb. 6.6a stellt die Ground-Truth dar, während die Abb. 6.6b das erzeugte Resultat des DL-Modells demonstriert. Dabei kann die Abb. 6.6b auch als eine Abweichung des Yaw-Winkelfehlers interpretiert werden. Die Abbildungen in 6.7 zeigen die



(a) Versuch 1: Ground-Truth Heading für handheld (b) Versuch 1: DL Heading für handheld

Abbildung 6.6.: Versuch 1: Erzeugte Heading Daten als Resultat des DL-Modell für handheld

vom DL-Modell erzeugten Trajektorien jeweils mit der dazugehörigen Ground-Truth überlagert. Darstellung 6.7a veranschaulicht das Szenario *fm*, wohingegen Abb. 6.7b die erstellte Trajektorie aus dem Heading in Abb. 6.6b des *hh*-Szenarios widerspiegelt.



(a) Versuch 1: Trajektorie foot-mounted (b) Versuch 1: Trajektorie für handheld

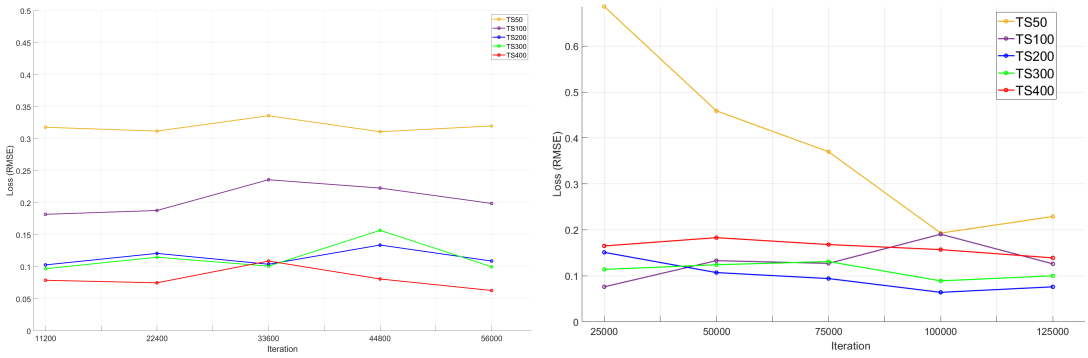
Abbildung 6.7.: Versuch 1: Erzeugte Trajektorie für fm und hh als Resultat des DL-Modells

6.4.2. Versuch 2

Im zweiten Versuch findet eine Untersuchung der verschiedenen Bewegungsformen auf unterschiedliche große Timesteps (TS) statt. Dabei ist vor allem interessant, wie viele Timesteps maximal benötigt werden, um Abhängigkeiten in der Bewegungsform zu erkennen und sie als temporale Struktur in den Gewichten zu verfestigen, um daraus eine Bestimmung der Position abzuleiten. Für diesen Versuch wurden die Parameter durch die neuen Erkenntnisse aus dem vorhergegangenen Versuchs entsprechend angepasst.

- Foot-Mounted: Eine Batch-Größe von 8000, 200 Timesteps und 4 Convolutional Layer
- Handheld: Eine Batch-Größe von 128, 200 Timesteps und 4 Convolutional Layer

Die Abbildung 6.8 demonstriert die Resultate beider Bewegungsformen. Dabei konnte die geringste Abweichung mit einem Wert von 6.3% für *fm* mit einer Timestep-Größe von 400 bei Iterationsschritt 56000 gemessen werden. Für *hh* hat sich die Timestep-Größe 200 als optimal herausgestellt. Die Abweichung für TS 200 beträgt bei Iterationsschritt 100000 ca. $7.4\ m$. So fällt bei *hh* weiterhin auf, dass ein zu klein gewähltes Zeitfenster genauso wenig die



(a) Foot-Mounted (b) Handheld

Abbildung 6.8.: Versuch 2: Auswirkung durch die Verwendung unterschiedlich großer Timesteps

richtige Wahl zu sein scheint, wie ein großes. Obwohl mit TS 200 der Fehler am niedrigsten und die Genauigkeit an dem Punkt am höchsten ist, erreicht man mit TS 300 an demselben Iterationsschritt einen sehr ähnlich niedrigen Wert von $8.9\ m$. Mit TS 400 stattdessen steigt der Fehler bereits um 40.8% wieder an.
Bei *fm* hingegen ist eine deutliche Abgrenzung ab Iterationsschritt 33600 erkennbar.

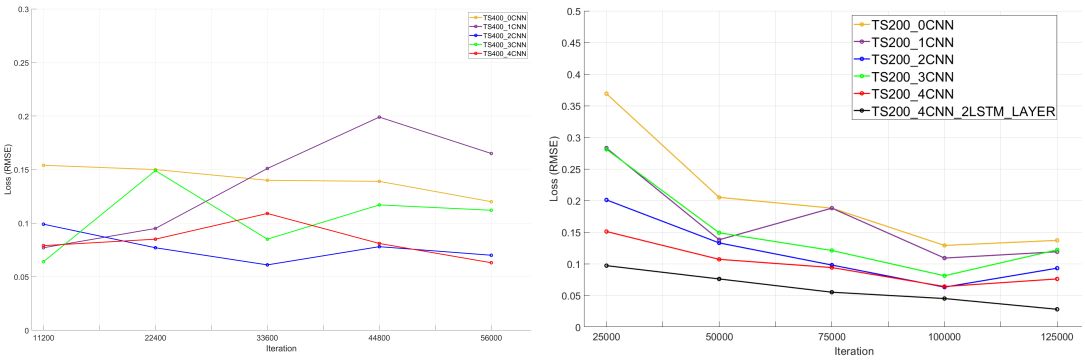
6.4.3. Versuch 3

Der Versuch 3 befasst sich mit der Untersuchung der Auswirkungen unterschiedlicher Anzahl an Convolutional Layern des DL-Modells auf die Positionsbestimmung. Dafür wurden zu Anfang die Trainingsparameter auf Basis der Resultate des vorhergegangen Versuchs wieder schrittweise angepasst.

- Foot-Mounted: Eine Batch-Größe von 8000, 400 Timesteps und 4 Convolutional Layer
- Handheld: Eine Batch-Größe von 128, 200 Timesteps und 4 Convolutional Layer.

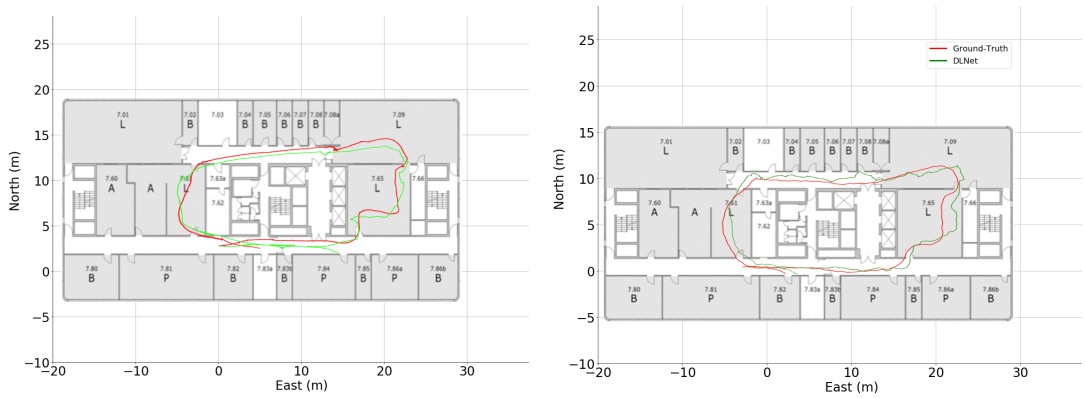
Die Ergebnisse des Versuchs in 6.9 zeigen, dass beide Bewegungsformen ihre Fehlerabweichungen nochmal optimieren konnten. So erreichte *fm* in Abb. 6.9a mit 2 Convolutional Layenr bei Iterationsschritt 33600 nur noch eine durchschnittliche Fehlerabweichung von $6.1\ m$. Für das *hh*-Szenario in Abb.6.9b konnte mit 4 Convolutional Layern und 2 RNN-Layern eine Reduzierung des Fehler von $36,8\%$ erreicht werden, das entspricht einem durchschnittlichen Fehler von $2.9\ m$.
Für *fm* konnten im Durchschnitt mit 2 Convolutional Layern ähnliche Resultate erzielt werden, wie bei der Verwendung von 4 Convolutional Layern. Zum Schluss erzielten beide letztendlich ein ziemlich identisches Resultat. Die Auswirkung auf eine unterschiedliche Anzahl an Convolutional Layern ist bei dem Szenario *hh* besonders auffällig, bei *fm* hingegen scheint der Convolutional Layer zwar ein wichtiges Fundament zu bilden, aber es sind dabei nicht viele Layer für eine optimale Leistung notwendig.

Die eben präsentierten Resultate werden in Abbildung 6.10 als Trajektorien überlagert auf den dazugehörigen Ground-Truths auf der Karte veranschaulicht. In beiden Darstellungen 6.10 kommen die von den DL-Modellen erzeugten Trajektorien den Ground-Truths sehr nahe.



(a) Foot-Mounted (b) Handheld

Abbildung 6.9.: Versuch 3: Verhalten bei der Verwendung unterschiedlicher Anzahl an Convolutional Layer



(a) Foot-Mounted (b) Handheld.

Abbildung 6.10.: Versuch 3: Verhalten bei der Verwendung unterschiedlicher Anzahl an Convolutional Layer

Tabelle 6.6.: Übersichts- und Referenztabelle der verwendeten Geräte während der Durchführung der Tests.

Bezeichnung	Gerät
Device 1	IMU (9Dof) SparkFun
Device 2	iphone 5
Device 3	iphone 6

6.4.4. Versuch 4

Der vierte Versuch beinhaltet ein finales Training mit einer Reihe von Tests mit verschiedenen Benutzern, Geräten und dem vollständigen Sensordatensatz (siehe Tabelle 4.1), welcher die aufgezeichneten Daten der Low-cost-Inertialsensoren vom iphone 5 und iphone 6 beinhalten. Die Trainingsparameter wurden den Resultaten aus den vorhergegangenen Versuchen entsprechend wie folgt angepasst:

- Foot-Mounted: Die Batch-Größe beträgt 8000, die Timesteps sind auf 400 festgelegt und die Anzahl an Convolutional Layern wurde auf 2 reduziert.
- Handheld: Die Batch-Größe beträgt 128, die Timesteps sind auf 200 festgelegt, die Anzahl an Convolutional Layern beträgt 4 und die RNN-Layern wurden auf 2 erhöht.

Für die Beurteilung der Leistung und die Darstellung der Ergebnisse der DL-Modelle kam die kumulative Verteilungsfunktion (CDF - Cumulative Distribution Function) zum Einsatz. Für die Leistungsbewertung der Modelle wurden, wie auch in den Versuchen zuvor, die Resultate der DL-Modelle gegen die Ground-Truths gemessen. Das Ergebnis der Messungen entspricht der Fehlerabweichung. In der Bewertung wurden zusätzlich die herkömmlichen PDR-Algorithmen als adäquater Vergleich miteinbezogen. Die Leistung der PDR-Algorithmen wurde ebenfalls bewertet und gegen die Ground-Truths gemessen. Um eine möglichst fehlerfreie Trajektorie mit den PDR-Algorithmen zu erzielen, musste jeder individuell dem Benutzer und dem Gerät angepasst werden. Die nachfolgenden Tabellen 6.6 und 6.7 dienen als Referenz für die verwendeten Geräte und Teilnehmer während der Durchführung der Tests. Die Datensätze der Teilnehmer und der Geräte sind nicht Teil des Trainingsdatensatzes gewesen.

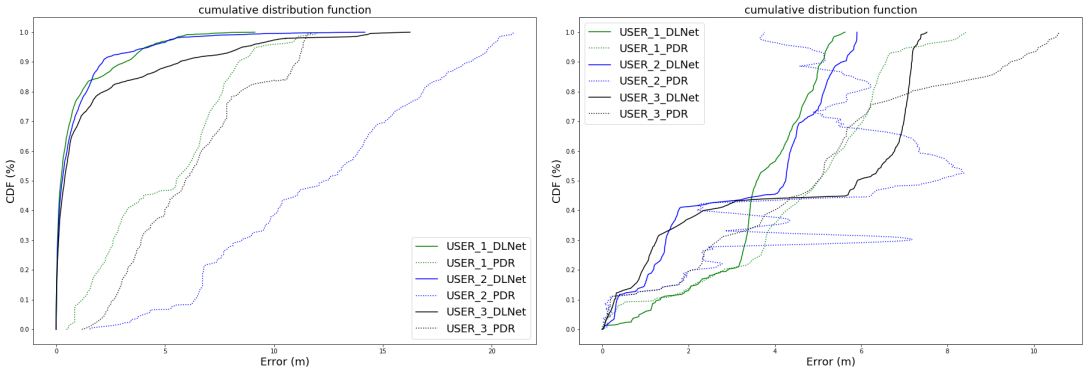
Tabelle 6.7.: Übersichts- und Referenztabelle der verwendeten Geräte während der Durchführung der Tests.

Bezeichnung	Teilnehmer
User 1	1.83 m
User 2	1.75 m
User 3	1.91 m

Route 1: Users – Foot-Mounted/Handheld

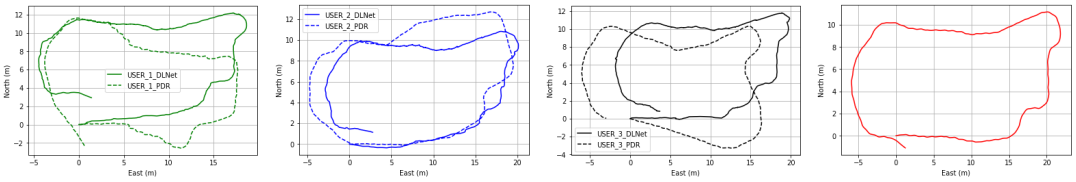
Die Abbildung 6.11 präsentiert die Fehlerabweichungen der einzelnen DL-Modelle (DLNet) mit den entsprechenden PDR-Algorithmen (PDR) als kumulative Verteilungsfunktion (CDF). Für beide Bewegungsformen wurden unterschiedliche Resultate erzielt. So konnte in Abb. 6.11a der maximale Fehler für das DLNet der Bewegungsform *foot-mounted* für 80% der Testzeit unter $2.5\ m$ gehalten werden. Den konkurrierenden PDR-Algorithmen war es hingegen nur möglich für ungefähr 20% der Testzeit den Fehler unter $5\ m$ zu halten.

Das *Handheld*-Szenario in Abb. 6.11b entzieht sich aufgrund mangelnder erreichter Genauigkeit dem direkten Vergleich zu *foot-mounted*. Obwohl die Resultate des DLNet die des PDRs noch übersteigen, kreuzen sie sich bereits bei 50% der Testzeit bei einem Fehler von ca. $4\ m$ und steigen fortwährend an.
Die dazugehörigen erstellten Positionsdaten für *hh* sind den Abbildungen 6.12 zu entnehmen. Die Positionsdaten der Bewegungsform *fm* sind in den Abbildungen 6.13 veranschaulicht. Die dabei erzeugten Trajektorien der *Foot-mounted*-Bewegungsform machen besonders mit einem unruhigen Regelverhalten auf sich aufmerksam.



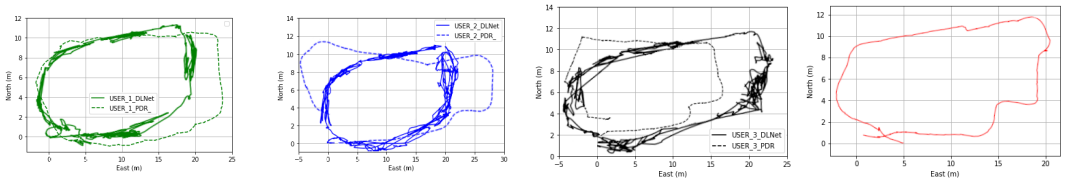
(a) Foot-Mounted (b) Handheld

Abbildung 6.11.: Versuch 4 - Route 1: Bewertung der Leistung durch unterschiedlicher **User** mittels CDF



(a) User 1 (b) User 2 (c) User 3 (d) Ground-Truth

Abbildung 6.12.: Versuch 4 - Route 1: Die Trajektorien der jeweiligen **User** der Bewegungsform **handheld** überlagert durch das DLNet mit dem PDR-Algorithmus



(a) User 1 (b) User 2 (c) User 3 (d) Ground-Truth

Abbildung 6.13.: Versuch 4 - Route 1: Die Trajektorien der jeweiligen **User** der Bewegungsform **foot-mounted** überlagert durch das DLNet mit dem PDR-Algorithmus

Route 1: Devices – Foot-Mounted/Handheld

Die Resultate der Experimente für unterschiedliche Geräte sind der Abbildung 6.14 zu entnehmen. Insgesamt zeigen die Ergebnisse beider Bewegungsformen bei unterschiedlichen Geräten deutlich bessere Resultate als im vorherigen Experiment durch verschiedene Teilnehmer.
Für das DLNet der *Foot-mounted*-Bewegungsform in Abb. 6.14a wurde ein Fehler für 90% der Testzeit unter $1.5\ m$ verzeichnet. Während bei den PDR-Verfahren bereits bei 40% der Zeit ein Drift größer $5\ m$ festgehalten werden kann.

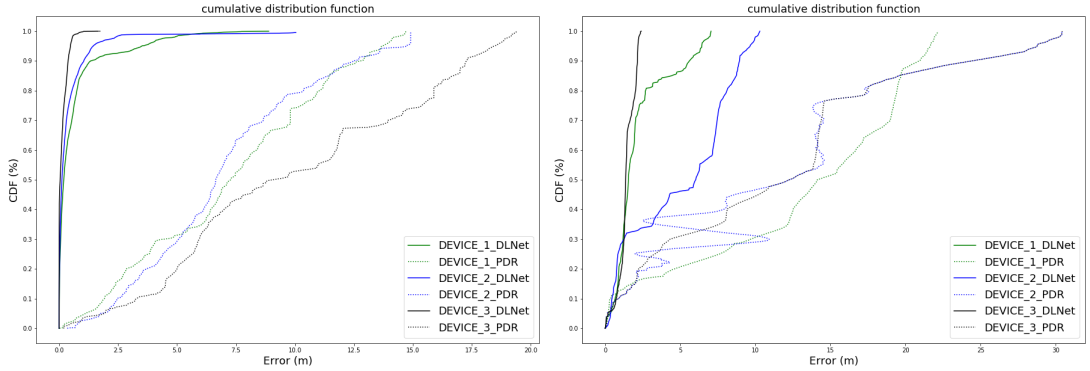
Für das DL-Modell der *Handheld*-Bewegungsform in Abb. 6.14b konnte für den IMU-Sensor (Device 1) und dem iphone 6 (Device 3) ein Fehler kleiner $2\ m$ für 80% der Zeit erfasst werden. Das iphone 5 erfährt stattdessen bei 30% eine zusätzliche Diskrepanz von ca. $3\ m$. Ein ausgeprägtes, aber dennoch ähnliches Verhalten ist bei dem entsprechenden PDR des iphone 5 zu erkennen. In einem Bereich zwischen $2.5\ m$ und $10\ m$ zeigt das Device 2 über 30% der Zeit ein oszillierendes Verhalten, bevor der Fehler einen weiteren Anstieg auf bis zu $30\ m$ zu verzeichnen hat.
Generell erreichten die Geräte der PDR-Verfahren nicht ansatzweise die Genauigkeiten der DL-Modelle. Dabei ist besonders der plötzliche Einbruch des Signalverlaufs vom iphone 5 bei ca. 30% des PDRs auffällig, da er beim DLNet wesentlich kleiner ausfällt.

Die Abbildungen in 6.15 demonstrieren die vom DL-Modell und PDR-Verfahren berechneten Positionen für das *Handheld*-Szenario. Die Positionsbestimmung der Abb. 6.15b für das iphone 5 macht dabei besonders auf sich aufmerksam. Während dem PDR-Algorithmus keine fehlerfreie Trajektorie gelingt, zeigt das DLNet eine gute Verallgemeinerungsfähigkeit.

Die Darstellungen aus 6.16 zeigen die berechneten Positionen für das *Foot-mounted*-Szenario. Es ist zu erkennen, dass die berechneten Positionen von Device 2 (iphone 5) vereinzelt Signalsprünge aufweisen, währenddessen das Device 3 (iphone 6) eine beinahe fehlerfreie Positionsbestimmung gelingt. Für die entsprechenden PDR-Verfahren ist es für kein Gerät möglich gewesen eine saubere Bestimmung der Positionen zu berechnen.

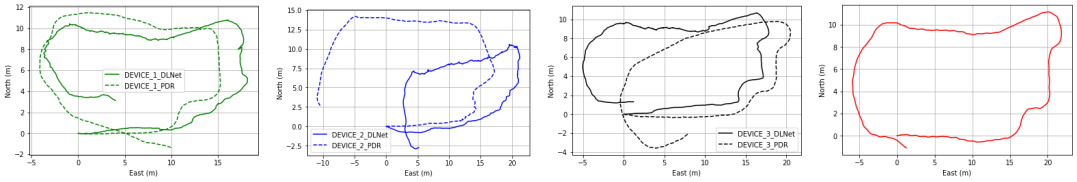
Route 2: Users – Foot-Mounted/Handheld

Die Abbildung 6.17 demonstriert die Ergebnisse beider Bewegungsformen für die zweite Route. Insgesamt wurden in Abb. 6.17a mit *fm* die deutlich besseren Resultate als mit *hh* erzielt. Bis auf dem User 2 (in Abb. 6.17a), wurde über 90% der Testzeit der Fehler kleiner $1.5\ m$ gehalten.



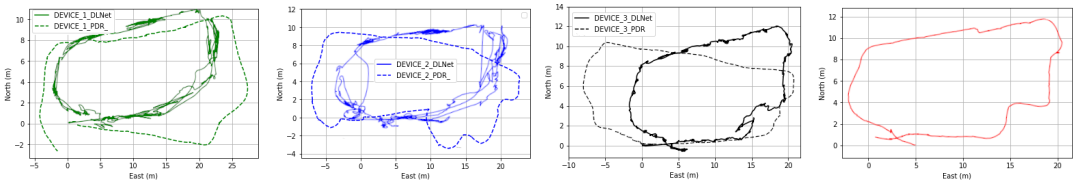
(a) Foot-Mounted (b) Handheld

Abbildung 6.14.: Versuch 4 - Route 1: Bewertung der Leistung durch unterschiedliche **Devices** mittels CDF



(a) Device 1 (b) Device 2 (c) Device 3 (d) Ground-Truth

Abbildung 6.15.: Versuch 4 - Route 1: Die Trajektorien der **Devices** für die Bewegungsform **handheld** überlagert durch das DLNet mit dem PDR-Algorithmus

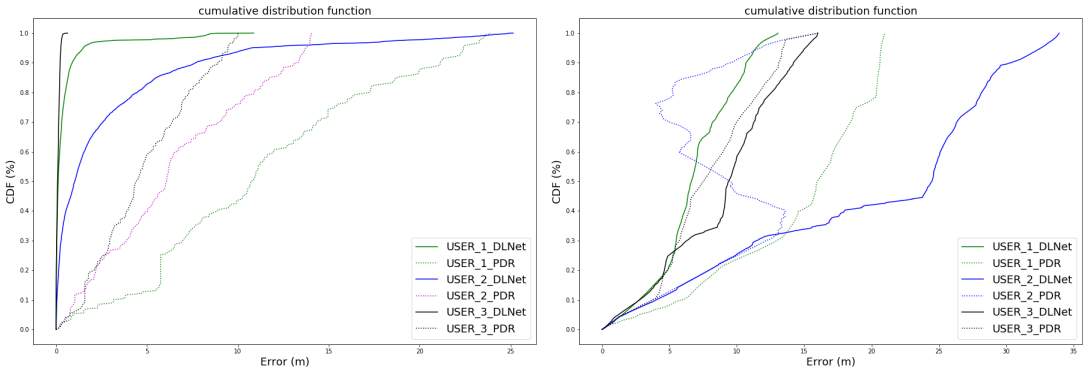


(a) Device 1 (b) Device 2 (c) Device 3 (d) Ground-Truth

Abbildung 6.16.: Versuch 4 - Route 1: Die Trajektorien der **Devices** für die Bewegungsform **foot-mounted** überlagert durch das DLNet mit dem PDR-Algorithmus

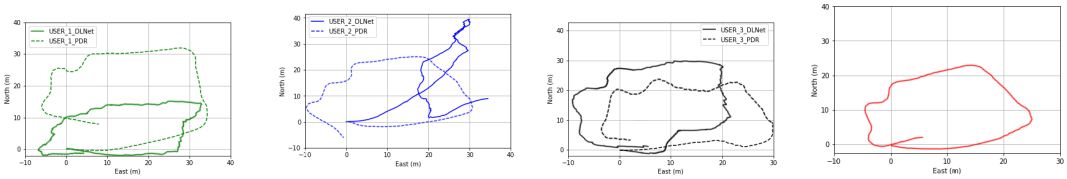
Die PDR-Algorithmen konnten den Fehler hingegen nur für ungefähr 15% der Testzeit unter 2 m halten. Dieses Verhalten spiegelt sich auch in Abbildungen 6.19 wieder. Besonders auffällig ist der Signalverlauf des Users 2 in Abb. 6.19b. Die berechneten Positionen zeigen starke Schwankungen im Signalverlauf, wohingegen die anderen Teilnehmer weitestgehend frei davon sind.

Für das *Handheld*-Szenario in Abb. 6.17b zeichnen sich im Großen und Ganzen keine großen Unterschiede zwischen dem DLNet und dem PDR-Verfahren ab. Lediglich der User 2 zeigt besondere Auffälligkeiten. Denn bei 30% ist ein Einbruch des DLNet zu erkennen, wo im Anschluss der Fehler weiter stark anwächst, während der PDR-Algorithmus an dieser Stelle den Fehler kompensiert und ihn dadurch stabilisiert. Die Abbildungen 6.18 demonstrieren die dazugehörigen Trajektorien.



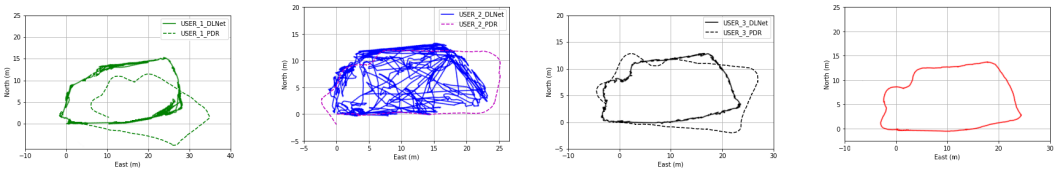
(a) Foot-Mounted (b) Handheld

Abbildung 6.17.: Versuch 4 - Route 2: Bewertung der Leistung durch unterschiedliche **User** mittels CDF



(a) User 1 (b) User 2 (c) User 3 (d) Ground-Truth

Abbildung 6.18.: Versuch 4 - Route 2: Die Trajektorien der *User* für die Bewegungsform **handheld** überlagert durch das DLNet mit dem PDR-Algorithmus



(a) User 1 (b) User 2 (c) User 3 (d) Ground-Truth

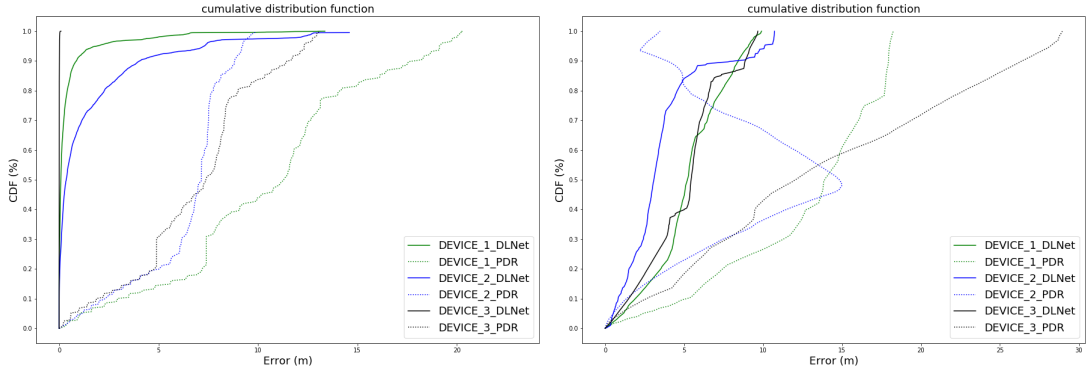
Abbildung 6.19.: Versuch 4 - Route 2: Die Trajektorien der **User** für die Bewegungsform **foot-mounted** überlagert durch das DLNet mit dem PDR-Algorithmus

Route 2: Devices – Foot-Mounted/Handheld

Die Experimente unterschiedlicher Devices in Abb. 6.20 lassen wieder eine Abgrenzung zwischen den DL-Modellen und den PDR-Verfahren zu. So verzeichnen die Resultate der *Foot-mounted*-Bewegungsform in Abb. 6.20a einen Fehler kleiner 3 m über eine Dauer von 80%. Für Device 1 und Device 3 wurde sogar ein Wert kleiner $1m$ für 90% der Testzeit erreicht. Die PDR-Verfahren verzeichnen hingegen bereits nach 30% der Testzeit einen Fehler größer 5 m. Die geschätzten Positionsdaten sind in Abb. 6.22 als Kartenplot veranschaulicht.

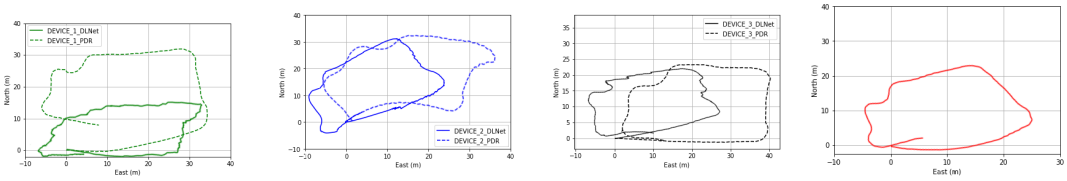
Bei dem *Handheld*-Szenario (Abb. 6.21) hingegen blieb der Fehler der DL-Modelle für 80% der Zeit unter einem Wert von 5 m. Die PDR-Verfahren erreichten die 5 m bereits nach 20%. Hier sei noch einmal hervorgehoben, dass Device 2 sich bei der Hälfte der Testzeit wieder stabilisieren und den Fehler durch Korrektur ausgleichen konnte. Die dazugehörigen erstellten Positionsdaten als Kartenplot sind den Abbildungen 6.21 zu entnehmen.

Abschließend kann festgehalten werden, dass bei den DL-Modellen beider Bewegungsformen im durchschnitt 82% der gelaufenen Strecke einen Fehler kleiner 3.5 m erfasst werden konnte. Nach 25% der gelaufener Strecke ist bei dem PDR-Verfahren beider Szenarien bereits eine Abweichung von mindestens 5 m zu verzeichnen.



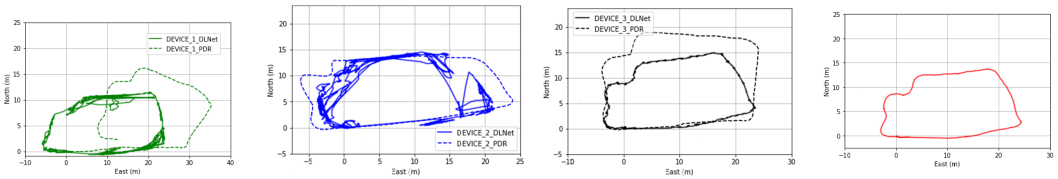
(a) Foot-Mounted (b) Handheld

Abbildung 6.20.: Versuch 4 - Route 2: Bewertung der Leistung durch unterschiedliche **Devices** mittels CDF



(a) Device 1 (b) Device 2 (c) Device 3 (d) Ground-Truth

Abbildung 6.21.: Versuch 4 - Route 2: Die Trajektorien der **Devices** für die Bewegungsform **handheld** überlagert durch das DLNet mit dem PDR-Algorithmus



(a) Device 1 (b) Device 2 (c) Device 3 (d) Ground-Truth

Abbildung 6.22.: Versuch 4 - Route 2: Die Trajektorien der **Devices** für die Bewegungsform **foot-mounted** überlagert durch das DLNet mit dem PDR-Algorithmus

6.4.5. Versuch 5

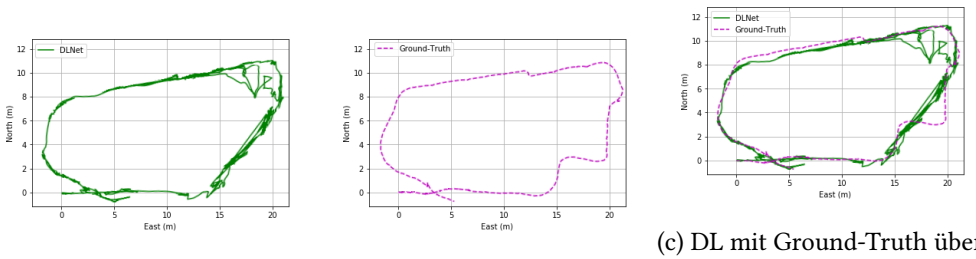
Zum Schluss soll ein letztes Experiment das Verhalten und die Generalisierungsfähigkeit des Modells für den Fall untersuchen, würde man Sensordaten einer Route in sie einspeisen, für die das DL-Modell nicht trainiert worden ist.

Für dieses Experiment wurde das *Foot-mounted*-Szenario herangezogen. Zum Testen und Validieren kamen die Testdatensätze der jeweiligen Routen zum Einsatz. Am Anfang jeden Experiments wurden für einen adäquaten Vergleich die DL-Modelle mit ihrem eigenen Testdatensatz validiert, bevor man ihnen den Testdatensatz der jeweils anderen Route eingespeiste.

Testdaten Route 2 in Route 1

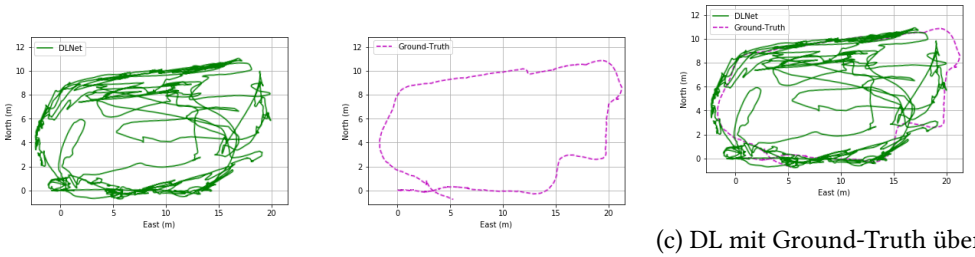
Die Darstellung 6.23 veranschaulicht nochmal die Ergebnisse des Testdatensatzes der Route 1 für das entsprechende trainierte DL-Modell der Route 1. Bis auf kleine Unregelmäßigkeiten in Abb. 6.23a, konnte eine nahezu fehlerfreie Bestimmung der Position erreicht werden. Der durchschnittlich erfasste Fehler beträgt $0.74\ m$.

Die Abbildung 6.24 demonstriert die Resultate des DL-Modells der Route 1 mit dem Testdatensatz der Route 2. Die Abbildung 6.24c zeigt starke Signalschwankungen in den berechneten Positionen. Zudem ähnelt die Trajektorie nicht der Route 2, sondern der Route 1. Der Fehler liegt ungefähr bei $10.1\ m$.



(a) DLNet (b) Ground-Truth (c) DL mit Ground-Truth überlagert

Abbildung 6.23.: Versuch 5: Ergebnis der Verallgemeinerungsfähigkeit des DL-Modell der Route 1 mit dem Testdatensatz für Route 1



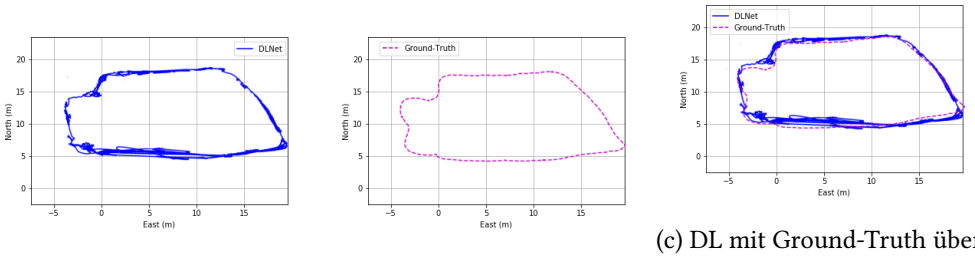
(a) DLNet (b) Ground-Truth (c) DL mit Ground-Truth überlagert

Abbildung 6.24.: Versuch 5: Ergebnis der Einspeisung des Testdatensatz der Route 2 in das DL-Modell der Route 1

Testdaten Route 1 in Route 2

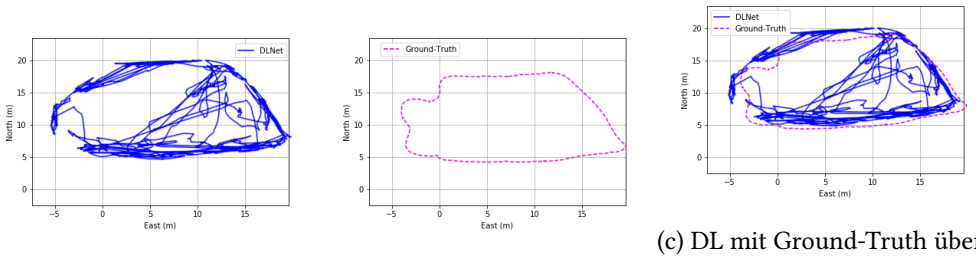
Das Vorgehen dieses Experiments ist dem Ablauf des vorherigen sehr ähnlich. Die Darstellungen in 6.25 repräsentieren die Ergebnisse des Testdatensatzes der Route 2 für das DL-Modell der Route 2. Die Resultate gleichen dem vorigen Experiment. Die Trajektorie der Darstellung 6.25a erlaubt eine fehlerlose Positionsbestimmung. Die Fehlerabweichung ist bei $0.62\ m$ einzuschätzen.

Die Abbildung 6.26 demonstriert die Resultate des DL-Modells der Route 2 mit dem Testdatensatz der Route 1. Die Ergebnisse sind mit dem vorhergegangenen Versuch vergleichbar, denn die berechneten Positionen spiegeln nicht die Route 1, vielmehr die Route 2 wieder. Die Signalsprünge in Abbildung 6.25c sind vor allem an der rechten Seite der Trajektorie stark ausgeprägt. Die Diskrepanz zwischen der geschätzten Position und der tatsächlichen beträgt $8.89\ m$.



(a) DLNet (b) Ground-Truth (c) DL mit Ground-Truth überlagert

Abbildung 6.25.: Versuch 5: Ergebnis der Verallgemeinerungsfähigkeit des DL-Modell der Route 2 mit dem Testdatensatz für Route 2



(a) DLNet (b) Ground-Truth (c) DL mit Ground-Truth überlagert

Abbildung 6.26.: Versuch 5: Ergebnis der Einspeisung des Testdatensatz der Route 1 in das DL-Modell der Route 2

7. Diskussion

In dieser Arbeit wurde ein Deep-Learning Modell zur Umsetzung einer inertialen Navigation vorgestellt, implementiert und durch umfangreiche Experimente ausgewertet. Die Durchführung der Tests ist mit unterschiedlichen Geräten und Personen umgesetzt werden. Zudem wurden zwei unterschiedliche Bewegungsformen in Relation zueinander gesetzt. Zum einen das *Foot-mounted*-Szenario, mit dem es möglich gewesen ist, direkt aus den Rohdaten eine Trajektorie zu erzeugen und zum anderen das *Handheld*-Szenario, welches zum Erstellen einer Trajektorie noch auf personenbezogenen Daten angewiesen war.
Die Experimente demonstrierten, dass DL-Verfahren das Potenzial aufweisen die traditionellen PDR-Verfahren vollständig zu ersetzen. Denn in jeden der Versuche konnten die DL-Modelle im Durchschnitt die herkömmlichen PDR-Algorithmen übertreffen.

Bereits zu Beginn der Experimente in Versuch 1 konnte gezeigt werden, welchen Einfluss die Batch-Größe auf das zu lösende Problem haben kann. Für das DL-Modell des *Handheld*-Szenarios war es daher eine größere Herausforderung aus den Sensordaten die Orientierung zu generalisieren als direkt die Positionsdaten zu berechnen. Was folglich aus den benötigten Iterationsschritten geschlussfolgert werden kann, die notwendig gewesen waren, um eine gültige Trajektorie zu erzeugen. Für die Orientierung spielte neben dem gewählten Zeitfenster auch die Anzahl der Updates (Gewichtsaktualisierungen) eine wichtige Bedeutung. Für das Erzielen einer guten Leistung musst eine Anpassung der Gewichte daher bereits immer nach kurzen Abständen erfolgen. Zu begründen lässt sich dies mit der Schwierigkeit der Verallgemeinerung der Orientierung der entsprechenden Bewegungsform. Ohne genauere Untersuchungen der Batch-Size auf das zu lösenden Problem hätte der Fehler um ca. 25.5% höher ausfallen können. Für das *Foot-mounted*-Szenario hätte die beste Leistung der kleinen Batch-Größen zu einer Reduzierung der Genauigkeit also Erhöhung des Fehlers zu 24.2% geführt. Des Weiteren fällt für das *Foot-mounted*-Szenario die Anzahl der notwendigen Iterationen erwartungsgemäß klein aus. Bereits das Minimum an durchgeführten Iterationen wäre ausreichend gewesen, um die best-mögliche Leistung mit dem DL-Modell für die zu lösenden Aufgabe zu erzielen. Im Gegensatz zu dem *Handheld*-Szenario, welches vermutlich mit einer höheren Anzahl an

Iterationen auch zu einer weiteren Steigerung der Leistung geführt hätte. Daraus lässt sich schließen, dass es sich in dieser Arbeit um zwei unterschiedlich zu lösende Probleme mit unterschiedlichen Anforderungen an Komplexität für den Lernalgorithmus gehandelt hat. Die Ergebnisse des Versuchs 1 decken sich zudem mit den Behauptungen in [Smith u. a. (2017)], woraufhin der Erfolg des Trainingsalgorithmus nicht nur von der Lernrate abhängig, sondern vielmehr von der Größe des Batches abhängig sein kann.

In Versuch 2 wurden unterschiedliche Zeitfenster (Timesteps) für die entsprechenden Bewegungsformen näher untersucht. Dabei müssen die Ergebnisse beider Bewegungsformen differenziert betrachtet werden. Für das *Handheld*-Szenario erwies sich ein Zeitfenster von 2 Sekunden als die richtige Wahl, während das Zeitfenster für *foot-mounted* weitaus größer wie 4 Sekunden hätte ausfallen können. Eine größere Anzahl an Timesteps wirkte sich für auf das *Handheld*-Szenario in Abb. 6.8b wiederum negativ auf die Genauigkeit aus. Eine mögliche Erklärung für diese Ergebnisse können möglicherweise Arbeiten aus dem Bereich der Ganganalyse zugrunde liegen. Diese Arbeiten [Vezocnik und Juric (2019), Alvarez u. a. (2006)] bestätigen, dass ein Schritt sich in einem Zeitfenster von 0.72 und 1.30 Sekunden befindet. Ein Doppelschritt ist abhängig von der Geschwindigkeit, Alter und Zustand eines Menschen und liegt in der Regel zwischen 1.2 und 2.5 Sekunden. Ausgehend davon, dass mit einem Doppelschritt eine Kursänderung einer Person eingeleitet wird, liegt in dieser Zeit die Timestep-Größe von 200 begründet. Des Weiteren lassen sich damit auch die negativen Werte bei einem Zeitfenster von 4 Sekunden ableiten. Zu begründen wäre das damit, dass in einem Zeitfenster größer 2 Sekunden bereits wieder eine neue Kursänderung stattgefunden haben könnte, wie man aus der Abb. 4.6 abzulesen kann (zwei Kursänderungen in kurzer Zeit), was das prognostizieren der Orientierung einer Person nicht möglich macht. Für das *Foot-mounted*-Szenario ist wiederum deutlich erkennbar, dass eine steigende Anzahl an Timesteps zu einer höheren Genauigkeit führt. Begründet werden lässt sich das mit der zu lernenden Aufgabe. Während man die Orientierung einer Person über einen längeren Zeitraum nicht prognostizieren kann, ist es durchaus realistisch, Merkmale und temporale Abhängigkeiten eine festgelegten Strecke zu lernen, auf einem ähnlichen Konzept basieren die VIO-Verfahren.
Des Weiteren kann der Schluss gezogen werden, dass sich möglicherweise das *Foot-mounted*-Szenario auf das Lernen der Route fokussiert, während sich das *Handheld*-Szenario auf das Lernen der eigentlichen Bewegungsform begrenzt.
In [Chen u. a. (2018b)] fand ebenfalls eine Überprüfung auf eine unterschiedliche Anzahl Timesteps statt. Die Ergebnisse ergaben, dass das RNN die beste Leistung bei einer Timestep-Größe von 200, für die in Abbildung 3.5 dargestellten Bewegungsformen erreichen konnte.

Allerdings wurde in dieser Arbeit lediglich auf Positionsdaten gelernt. Aufgrund mangelnder detaillierter Informationen über die Versuchsdurchführung des Experiments in dieser Arbeit ist eine direkter Vergleich der Resultate schwierig zu bewerten. Die Vermutung liegt aber nahe, dass bei einer so kleinen Strecke wie sie die Abbildung 3.5 veranschaulicht, die ganze Strecke als Batch-Größe verwendet worden ist. Folglich bedeutet dies, dass temporale Abhängigkeiten einer kleinere Strecke sich auch in einem kleineren Zeitfenster widerspiegeln können.

Die Resultate des Versuchs 3 spiegeln den in Kapitel 5 beschriebenen Nutzen der Convolutional Layer wieder. Das *Hanheld*-Szenario verdeutlicht die Notwendigkeit der Convolutional Layer zum extrahieren von Merkmalen sowie zur Vermeidung von fehlerbehafteten Sensordaten. Das DL-Modell gelang in Versuch 3 (Kap. 6.4.3) eine Reduzierung des Fehlers mithilfe der Faltungsnetze und des zusätzlichen RNN Layers um $11.6\ m$.
Das *Foot-mounted*-Szenario profitierte ebenfalls durch den Einsatz der Convolutional Layer und kann daher eine Reduzierung des Fehlers um ca. $6\ m$ verzeichnen. Nach [Chen u. a. (2018)] eigenen sich die Convolutional Layer um die Fehlereigenschaften der Sensoren zu beseitigen und zu kompensieren. Aus diesem Grund wurde erwartungsgemäß angenommen, dass auch eine maximale Anzahl an Convolutional Layer sich in den Ergebnissen deutlich hervorheben kann, stattdessen kann in der *Foot-Mounted*-Bewegungsform zwischen 2 und 4 Convolutional Layer kein Unterschied ersichtlich ausgemacht werden. Dem Ergebnis kann daher zugrunde gelegt werden, dass die Sensordaten nicht so fehlerbehaftet sind wie zunächst angenommen. Das würde bedeuten, dass das *Handheld*-Szenario die hohe Anzahl Convolutional Layer aufgrund des hohen Bedarfs an Merkmalen benötigt, die es zu extrahieren gilt.

In einem finalen Training mit verschiedenen Anwendern und Geräten in Kapitel 6.4.4-Versuch 5 lässt sich festhalten, dass sowohl die DL-Modelle als auch die PDR-Algorithmen sich gegenüber unterschiedlichen Anwendern (User) als weniger robust und zuverlässig zeigten. Dabei erzielten die DL-Modelle insgesamt für 55% der Testzeit einen Fehler von $4\ m$. Für die PDR-Verfahren kann für 40% der Zeit ein Fehler von $6\ m$ festgehalten werden. Die negativen Resultate können mit der Individualität der einzelnen Personen begründet werden. Während bei den PDR-Algorithmen die Fehlerquelle in der Schrittlängenschätzung ausgemacht werden kann, ist die mangelnde Verallgemeinerungsfähigkeit der DL-Modelle auf die Vielfalt und Breite der Trainingsdaten zurückzuführen, wohingegen die Resultate der Positionsbestimmung bei unterschiedlichen Geräten (Devices) der DL-Modelle für mindestens 80% der Testzeit einen Fehler kleiner $2\ m$ aufweisen können. Das entspricht eine um $30 - 40\%$ bessere Leistung

gegenüber den herkömmlichen PDR-Algorithmen.

Die *Foot-mounted*-Bewegungsform zeigte zudem immer wieder Anzeichen von einem unruhigen Regelverhalten. Dieses Verhalten konnte speziell bei Datensätzen beobachtet werden, wo auf Testdaten mit unterschiedlichen Anwendern die Positionen bestimmt werden sollte. Daraus lässt sich Schlussfolgern, dass der Einfluss von Trainingsdaten unterschiedlicher Personen eine genauso wichtige Bedeutung gegeben werden muss, wie der durch verschiedene Geräte bzw. Low-cost-Sensoren. Auch das *Hanheld*-Szenario ist davon nicht befreit. Obwohl bei *hh* kein Oszillierendes Verhalten erkennbar gewesen ist, zeigt das Verhalten sich in Form, wie die Ergebnisse in Abb. 6.18b veranschaulichen. Dem DL-Modell gelingt dabei keine fehlerfreie Trajektorie, stattdessen erzeugt das DL-Modell aus den Resultaten eine Schleife. Des Weiteren lässt sich daraus schließen, dass das *Foot-mounted*-Szenario sich gegenüber fehlerhafte Daten als wesentlich robuster präsentiert als es das *Hanheld*-Szenario macht.

Auf vergleichbaren Arbeiten mit demselben Umfang der Arbeit kann nicht zurückgegriffen werden. Die in Kapitel 3 vorgestellte Arbeit von [Chen u. a. (2018b)] erreichte in einem $(5\ m \times 8\ m)$ großen Raum (siehe Abb. 3.5) eine Genauigkeit bzw. einen Fehler von ca. $2\ m$ für 90% der gelaufenen Strecke mit unterschiedlichen Bewegungsformen. Bewertet man lediglich die Leistung des verwendeten DL-Modells mit dem das in dieser Arbeit umgesetzten, kann die Leistung in dieser Arbeit mit einer durchschnittlichen Abweichung für Route 1: ca. $0.74\ m$ und für Route 2 $0.62\ m$, einen durchaus hohen Stellenwert zugesprochen werden. Die Resultate entstammen dabei dem Versuch 5. Im Gegensatz zu Versuch 3, kam in Versuch 5 ein weitaus größerer Trainingsdatensatz mit zum Einsatz. Eine höhere Genauigkeit konnte in [Cortés u. a. (2018)] erreicht werden (siehe Kapitel 3). Auf einer Strecke von ca. $238.38\ m$ wurde eine Abweichung von durchschnittlich $0.62\ m$ erzielt. Hierbei konnte der Fehler so klein gehalten werden, weil die Geschwindigkeit durch eine hervorgehende Klassifikation der Aktivität geschätzt und der verursachte Drift der Orientierung während den *Zero-Velocity*-Phasen gelöscht wurde.

Abschließend veranschaulichte der Versuch 5 die Generalisierungsfähigkeit der DL-Modelle bei Verwendung von Sensordaten einer anderen Strecke. Die Einspeisung der jeweils anderen Route in die entsprechenden DL-Modelle löste bei beiden ein starkes Kompensationsverhalten aus. Des Weiteren gab es keine Anzeichen eines Wiedererkennungswerts der jeweiligen fremden Strecke. Daraus kann geschlossen werden, dass die DL-Modelle nur auf die Strecken verallgemeinern können, für die sie auch trainiert worden sind.

8. Zusammenfassung

8.1. Zusammenfassung

8.2. Fazit

Das Ziel dieser Arbeit ist gewesen, einen Lösungsansatz auf Basis von Deep-Learning Methoden zur inertialen Navigation mittels Low-cost-Inertialsensoren vorzustellen, zu implementieren und im Anschluss zu evaluieren. Es wurde viel Wert darauf gelegt auch Low-cost-Inertialsensoren, wie sie mittlerweile in jedem Smartphone zu finden sind, eine relative Bestimmung der Position zu ermöglichen. Dabei wurden zwei unterschiedliche Bewegungsformen näher untersucht und miteinander verglichen. Die Bewegungsform *foot-mounted* und die Bewegungsform *handheld*. Beide spiegeln den alltäglichen Gebrauch des Smartphones sowie die Anwendungen von Low-cost-Inertialsensoren in der Praxis wieder.
Eine Positionsbestimmung des *Foot-mounted*-Szenario konnte direkt aus den Rohdaten berechnet werden, während das *Handheld*-Szenario zusätzlich von personenbezogenen Daten abhängig gewesen ist.

In einer Reihe von aufwendigen, umfangreichen und anspruchsvollen Experimenten wurde auf Basis von wissenschaftlich fundierten Kenntnisse Parameterwerte für Metaparameter ermittelt und sie im Bezug auf dem gesetzten Ziel dieser Arbeit eine Untersuchung unterzogen. Dabei demonstrierte die Arbeit, wie wichtig eine individuelle Bestimmung von Hyperparametern für die zu lösende Aufgabe sein kann. Die Versuche lieferten wichtige Erkenntnisse über die Möglichkeiten der Umsetzung und sowie das Verhalten und die Auswirkungen der Realisierung einer inertialen Navigation mittels Deep-Learning. Hervorzuheben ist dabei die Erkenntnis, dass sich das *Foot-mounted*-Szenario auf as Lernen der Route fokussierte, während das *Handheld*-Szenario sich auf das Lernen der Orientierung aus der Bewegungsform begrenzte. Die Arbeit kann damit als Grundlage für weitere Arbeiten dienen.

Des Weiteren fand zum Schluss eine Beurteilung der Leistung des DL-Modells auf unterschiedliche Benutzer und Geräte statt. Dieser Versuch hat zum Ziel die Generalisierungsfähigkeit auf weitere Low-cost-Sensoren zu untersuchen und zu validieren. Die Ergebnisse zeigten eine schlechtere Verallgemeinerungsfähigkeit gegenüber unterschiedlicher Personen. Daraus konnte abgeleitet werden, dass die Sensordaten sich durch die individuellen Eigenschaften und Bewegungen von Personen stärker unterscheiden als angenommen. Gegenüber verschiedenen Geräten zeigt sich das Modell als zunehmend zuverlässig und robust.

Außerdem wurde aufgezeigt, inwiefern sich Inertialdaten einer anderen Strecke auf ein DL-Modell übertragen lassen. Dabei konnte festgestellt werden, dass sich die Verallgemeinerungsfähigkeit lediglich auf die zu trainierende Route begrenzt. Der Rückschluss konnte aufgrund eines stark beobachtenden Kompensationsverhalten gezogen werden.

Es konnte demonstriert werden, dass DL-Methoden das Potenzial aufweisen nicht nur die traditionellen PDR-Verfahren zu ersetzen, sondern sich mit der Zeit auch als eine der häufigst verschwendeten Methoden zu etablieren. Obwohl sie einige Defizite bei der Anwendung mit verschiedenen Benutzern im Bezug auf personenbezogenen Daten aufweisen, zeigten sie sich als zunehmend robust, zuverlässig und überwiegend unempfindlich gegen äußere Einfluss wie bspw. fehlerbehaftete Messungen.
Für erste praxisnahe Anwendungen auf Low-cost-Systemen für den alltäglichen Gebrauch würde sie sich bereits ohne weiteres empfehlen.

8.3. Ausblick

Der aktuelle Lösungsansatz ist derzeit noch auf die Bewegungsformen *foot-mounted* und *handheld* begrenzt. Zudem zeigt sich das derzeitige DL-Modell noch besonders anfällig durch den Einsatz verschiedener Benutzer bezüglich individuellen Bewegungen. Außerdem ist eine Unterscheidung zwischen den Bewegungsformen nicht möglich. Daher muss die Art der Navigation im Bezug auf die Bewegungsform im Vorhinein festgelegt werden.

Um den aktuellen Ansatz zu ergänzen, wäre eine Erweiterung des Datensatzes notwendig. Die Erweiterungen sieht dabei Trainingsdaten unterschiedlicher Personen und Geräte (Smartphones) vor. Durch individuelle personenbezogenen Daten, würde eine stärkere Generalisierungsfähigkeit erreicht werden.

Mit einem einzigen DL-Modell verschiedene Bewegungsformen zu vereinigen, um eine inertiale Navigation umzusetzen, wäre sicherlich ein wünschenswerter Ansatz, würde das Modell aufgrund der Menge an Merkmalen die es für die verschiedenen Benutzer, Geräte und Bewegungsformen zu extrahiert gilt, vermutlich schnell an seine Grenzen bringen.
Ein Vorschlag zur Umsetzung wäre aus diesem Grund, den aktuellen Ansatz um einen sog. **Bewegungs-Transformer** zu ergänzen. Die Realisierung sieht dabei eine Transformation der Signale durch ein vorgeschaltetes Encoder-Decoder-Netz vor. Mit dem Ziel invariante Merkmale aus den Rohdaten der Bewegungen zu extrahieren, um sie im Anschluss in das DL-Modell einzuspeisen, um daraus Positionsdaten abzuleiten und eine Trajektorie zu erzeugen. Zum einen würde dadurch eine gewisse Transparenz geschaffen und zum anderen wäre eine Bestimmung der Position unabhängig der Bewegungsformen realisierbar.

A. Inhalte der Begleit-DVD

Die beiliegende DVD enthält die nachfolgend aufgeführten Daten und Artefakte. Die folgende Aufzählung spiegelt dabei die Verzeichnisstruktur wider:

- **MA_MeyerManuel_InertialeNavigationMittelsDeepLearning.pdf.pdf**: PDF-Datei des vorliegenden Dokuments.
- **Datensatz**: Der verwendete Datensatz. Dargestellt in Tabelle 4.1 für:
  (3 Users_Devices /Handheld Pocket_Foot)
- **Source Code**: Ausführbarer Quellcode für Jupyter Notebook sowie JetBrains:
  (DNN_INS.py / main_foot_mounted.py / main_handheld.py)
- **Matlab_INS**: Source Code der PDR-basierten-Ansätze für handheld und foot-mounted sowie den implementierten Madgwick-Algorithmus mit der Quaternions Bibliothek

B. Verwendete Werkzeuge

Nachfolgend werden Werkzeuge und Programme aufgelistet, die in dieser Arbeit zum Einsatz gekommen sind.

- **JetBrains PyCharm**: Version 2018.2.3
  Python Entwicklungsumgebung
- **Matlab**: Version R2018b_x64
  Entwicklungen der PDR-basierten Algorithmen und Ground-Truths
- **Jupyter Notebook**:
  Client-Server-Anwendung zur Übertragung der SW-Architekturen und Ausführung der Simulation auf dem Server
- **TensorFlow Framework**: Version 1.4.0
  Framework zur Entwicklungen der Softwarearchitekturen für Neuronale Netze

Literaturverzeichnis

[Abe u. a. 2016] Abe, M. ; Kaji, K. ; Hiroi, K. ; Kawaguchi, N.: PIEM: Path Independent Evaluation Metric for Relative Localization. In: *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Oct 2016, S. 1–8

[Alvarez u. a. 2006] Alvarez, D. ; Gonzalez, R. C. ; Lopez, A. ; Alvarez, J. C.: Comparison of Step Length Estimators from Weareable Accelerometer Devices. In: *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2006, S. 5964–5967

[Bahillo u. a. 2015] Bahillo, A. ; Angulo, I. ; Onieva, E. ; Perallos, A. ; Fernández, P.: Low-cost Bluetooth foot-mounted IMU for pedestrian tracking in industrial environments. In: *2015 IEEE International Conference on Industrial Technology (ICIT)*, March 2015, S. 3430–3434

[Beauregard 2006] Beauregard, S.: A Helmet-Mounted Pedestrian Dead Reckoning System. In: *3rd International Forum on Applied Wearable Computing 2006*, March 2006, S. 1–11

[Berlin 2015] Berlin, TU: Tutorial - Razor AHRS - Q u. U Lab Project Hosting. (2015). – URL https://www.electronicaembajadores.com/datos/manuales/ss/ssac/ssacim1.pdf. – Zuletzt besucht am 05.06.2019

[Bosch 2019] Bosch, Robert: Funktionsweise des peripheren Beschleunigungssensors. (2019)

[Brownlee 2019] Brownlee, PhD J.: Stateful and Stateless LSTM for Time Series Forecasting with Python. In: *Machine Learning Mastery* (2019). – URL https://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/. – Zuletzt besucht am 11.06.2019

[Burri u. a. 2016] Burri, Michael ; Nikolic, Janosch ; Gohl, Pascal ; Schneider, Thomas ; Rehder, Joern ; Omari, Sammy ; Achtelik, Markus W. ; Siegwart, Roland: The EuRoC micro aerial vehicle datasets. In: *The International Journal of Robotics Research* (2016). – URL http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[Chen u. a. 2018a] Chen, Changhao ; Lu, Chris X. ; Markham, Andrew ; Trigoni, Niki: IONet: Learning to Cure the Curse of Drift in Inertial Odometry. In: *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018

[Chen u. a. 2018b] Chen, Changhao ; Zhao, Peijun ; Lu, Chris X. ; Wang, Wei ; Markham, Andrew ; Trigoni, Niki: OxIOD: The Dataset for Deep Inertial Odometry. In: *CoRR* abs/1809.07491 (2018). – URL http://arxiv.org/abs/1809.07491

[Chen u. a. 2018] Chen, H. ; Aggarwal, P. ; Taha, T. M. ; Chodavarapu, V. P.: Improving Inertial Sensor by Reducing Errors using Deep Learning Methodology. In: *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, July 2018, S. 197–202

[Clark u. a. 2017a] Clark, Ronald ; Wang, Sen ; Markham, Andrew ; Trigoni, Niki ; Wen, Hongkai: VidLoc: 6-DoF Video-Clip Relocalization. In: *CoRR* abs/1702.06521 (2017). – URL http://arxiv.org/abs/1702.06521

[Clark u. a. 2017b] Clark, Ronald ; Wang, Sen ; Wen, Hongkai ; Markham, Andrew ; Trigoni, Niki: VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem. In: *CoRR* abs/1701.08376 (2017). – URL http://arxiv.org/abs/1701.08376

[Cortés u. a. 2018] Cortés, S. ; Solin, A. ; Kannala, J.: Deep Learning Based Speed Estimation For Constraining Strapdown Inertial Navigation On Smartphones. In: *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, Sep. 2018, S. 1–6

[Dam u. a. 1998] Dam, Erik B. ; Koch, Martin ; Lillholm, Martin: Quaternions, interpolation and animation. 1998. – Forschungsbericht

[Davidson und Piché 2017] Davidson, P. ; Piché, R.: A Survey of Selected Indoor Positioning Methods for Smartphones. In: *IEEE Communications Surveys Tutorials* 19 (2017), Secondquarter, Nr. 2, S. 1347–1370

[Diaz u. a. 2015] Diaz, E. M. ; de Ponte Müller, F. ; JimÃ©nez, A. R. ; Zampella, F.: Evaluation of AHRS algorithms for inertial personal localization in industrial environments. In: *2015 IEEE International Conference on Industrial Technology (ICIT)*, March 2015, S. 3412–3417

[Electronics 2019] Electronics, SparkFun: *SparkFun 9DoF Razor IMU M0*. 2019. – URL https://www.sparkfun.com/products/14001. – Zuletzt besucht am 25.10.2019

[Fu u. a. 2018] Fu, W. ; Peng, A. ; Tang, B. ; Zheng, L.: Inertial sensor aided visual indoor positioning. In: *2018 International Conference on Electronics Technology (ICET)*, May 2018, S. 106–110

[Geiger u. a. 2013] Geiger, Andreas ; Lenz, Philip ; Stiller, Christoph ; Urtasun, Raquel: Vision meets Robotics: The KITTI Dataset. In: *International Journal of Robotics Research (IJRR)* (2013)

[Geiger u. a. 2012] Geiger, Andreas ; Lenz, Philip ; Urtasun, Raquel: Are we ready for Autonomous Driving The KITTI Vision Benchmark Suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012

[Géron 2017] Géron, Aurélien: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems.* 1st. O'Reilly Media, Inc., 2017. – ISBN 1491962291, 978-1491962299

[Harle 2013] Harle, R.: A Survey of Indoor Inertial Positioning Systems for Pedestrians. In: *IEEE Communications Surveys Tutorials* 15 (2013), Third, Nr. 3, S. 1281–1293

[Hilsenbeck u. a. 2014] Hilsenbeck, Sebastian ; Bobkov, Dmytro ; Schroth, Georg ; Huitl, Robert ; Steinbach, Eckehard: Graph-based Data Fusion of Pedometer and WiFi Measurements for Mobile Indoor Positioning. In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. New York, NY, USA : ACM, 2014 (UbiComp '14), S. 147–158. – URL http://doi.acm.org/10.1145/2632048.2636079. – ISBN 978-1-4503-2968-2

[Hochreiter und Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-term Memory. In: *Neural Comput.* 9 (1997), November, Nr. 9, S. 1735–1780. – URL http://dx.doi.org/10.1162/neco.1997.9.8.1735. – ISSN 0899-7667

[Hofmann-Wellenhof u. a. 2003] Hofmann-Wellenhof, B. ; Legat, K. ; Lichtenegger, H. ; Wieser, M.: *Navigation.* Springer Vienna, 2003 (Springer Nature Book Archives Millennium). – URL https://www.springer.com/de/book/9783211008287. – ISBN 9783211008287

[Hsu 2008] Hsu, T.R.: *MEMS and Microsystems: Design, Manufacture, and Nanoscale Engineering.* Wiley, 2008. – ISBN 9780470083017

[Inderst u. a. 2015] Inderst, F. ; Pascucci, F. ; Santoni, M.: 3D pedestrian dead reckoning and activity classification using waist-mounted inertial measurement unit. In: *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Oct 2015, S. 1–9

[InvenSense 2019] InvenSense, Inc.: MPU-9250 Product Specification Revision 1.1. (2019). – URL https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

[Jekeli 2001] Jekeli, C.: *Inertial Navigation Systems with Geodetic Applications*. Walter de Gruyter, 2001. – ISBN 9783110159035

[Jin u. a. 2011] Jin, Y. ; Hong-Song Toh ; Soh, W. ; Wai-Choong Wong: A robust dead-reckoning pedestrian tracking system with low cost sensors. In: *2011 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2011, S. 222–230

[Karrasch 2018] Karrasch, Dr. C.: *CIMON - der intelligente Astronautenassistent*. 2018

[Kasei 2019] Kasei, Corporation A.: Datasheet - 3-axis Electronic Compass. (2019). – URL https://www.akm.com/akm/en/file/datasheet/AK8963C.pdf. – Zuletzt besucht am 15.07.2019

[Keneshloo u. a. 2018] Keneshloo, Yaser ; Shi, Tian ; Ramakrishnan, Naren ; Reddy, Chandan K.: Deep Reinforcement Learning For Sequence to Sequence Models. In: *CoRR* abs/1805.09461 (2018). – URL http://arxiv.org/abs/1805.09461

[Keskar u. a. 2016] Keskar, Nitish S. ; Mudigere, Dheevatsa ; Nocedal, Jorge ; Smelyanskiy, Mikhail ; Tang, Ping Tak P.: On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In: *CoRR* abs/1609.04836 (2016). – URL http://arxiv.org/abs/1609.04836

[Kingma und Ba 2014] Kingma, Diederik P. ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. In: *CoRR* abs/1412.6980 (2014)

[Kittel 2005] Kittel, A.: *Physikalische Messtechnik*. Vorlesungsskript 'Physikalische Messtechnik'. 2005. – URL http://www.physik.uni-oldenburg.de/Docs/epkos/Messtechnik.pdf. – Zuletzt besucht am 17.08.2019

[Krizhevsky u. a. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. USA : Curran As-

sociates Inc., 2012 (NIPS'12), S. 1097–1105. – URL http://dl.acm.org/citation.cfm?id=2999134.2999257

[Krohn u. a. 2005] Krohn, Albert ; Beigl, Michael ; Decker, Christian ; Kochendörfer, Uwe ; Robinson, Philip ; Zimmer, Tobias: Inexpensive and Automatic Calibration for Acceleration Sensors. In: *Proceedings of the Second International Conference on Ubiquitous Computing Systems*. Berlin, Heidelberg : Springer-Verlag, 2005 (UCS'04), S. 245–258. – URL http://dx.doi.org/10.1007/11526858_19. – ISBN 3-540-27893-1, 978-3-540-27893-1

[Lakkhanawannakun und Noyunsan 2019] Lakkhanawannakun, P. ; Noyunsan, C.: Speech Recognition using Deep Learning. In: *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, June 2019, S. 1–4

[Lamy-Perbal u. a. 2015] Lamy-Perbal, S. ; GuÃ©nard, N. ; Boukallel, M. ; Landragin-Frassati, A.: A HMM map-matching approach enhancing indoor positioning performances of an inertial measurement system. In: *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Oct 2015, S. 1–4

[Latif u. a. 2019] Latif, J. ; Xiao, C. ; Imran, A. ; Tu, S.: Medical Imaging using Machine Learning and Deep Learning Algorithms: A Review. In: *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Jan 2019, S. 1–5

[Lawrence 2001] Lawrence, A.: *Modern Inertial Technology: Navigation, Guidance, and Control*. Springer New York, 2001 (Mechanical Engineering Series). – ISBN 9780387985077

[LeCun u. a. 1989] LeCun, Y. ; Boser, B. ; Denker, J. S. ; Henderson, D. ; Howard, R. E. ; Hubbard, W. ; Jackel, L. D.: Backpropagation Applied to Handwritten Zip Code Recognition. In: *Neural Comput.* 1 (1989), Dezember, Nr. 4, S. 541–551. – URL http://dx.doi.org/10.1162/neco.1989.1.4.541. – ISSN 0899-7667

[LeCun u. a. 1998] LeCun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-Based Learning Applied to Document Recognition. In: *Proceedings of the IEEE* 86 (1998), November, Nr. 11, S. 2278–2324

[Lecun u. a. 1998] Lecun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nov, Nr. 11, S. 2278–2324

[LeCun u. a. 1998] LeCun, Yann ; Bottou, Léon ; Orr, Genevieve B. ; Müller, Klaus-Robert: Efficient BackProp. In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK : Springer-Verlag, 1998, S. 9–50. – URL http://dl.acm.org/citation.cfm?id=645754.668382. – ISBN 3-540-65311-2

[Lei Fang u. a. 2005] Lei Fang ; Antsaklis, P. J. ; Montestruque, L. A. ; McMickell, M. B. ; Lemmon, M. ; Yashan Sun ; Hui Fang ; Koutroulis, I. ; Haenggi, M. ; Min Xie ; Xiaojuan Xie: Design of a wireless assisted pedestrian dead reckoning system - the NavMote experience. In: *IEEE Transactions on Instrumentation and Measurement* 54 (2005), Dec, Nr. 6, S. 2342–2358

[Li u. a. 2017] Li, J. ; Guo, M. ; Li, S.: An indoor localization system by fusing smartphone inertial sensors and bluetooth low energy beacons. In: *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, April 2017, S. 317–321

[Liu u. a. 2019] Liu, R. ; Shen, J. ; Chen, C. ; Yang, J.: SLAM for Robotic Navigation by Fusing RGB-D and Inertial Data in Recurrent and Convolutional Neural Networks. In: *2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR)*, May 2019, S. 1–6

[Ludwig und Burnham 2018] Ludwig, S. A. ; Burnham, K. D.: Comparison of Euler Estimate using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, S. 1236–1241

[Madgwick 2010] Madgwick, S. O. H.: An efficient orientation lter for inertial and inertial/magnetic sensor arrays, URL https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf, April 2010, S. 1–32

[Madgwick u. a. 2011] Madgwick, S. O. H. ; Harrison, A. J. L. ; Vaidyanathan, R.: Estimation of IMU and MARG orientation using a gradient descent algorithm. In: *2011 IEEE International Conference on Rehabilitation Robotics*, June 2011, S. 1–7

[Mahony u. a. 2008] Mahony, R. ; Hamel, T. ; Pflimlin, J.: Nonlinear Complementary Filters on the Special Orthogonal Group. In: *IEEE Transactions on Automatic Control* 53 (2008), June, Nr. 5, S. 1203–1218

[Marek 2007] Marek, Jiri: MEMS-Sensoren im Überblick. In: *AUTOMOBIL-ELEKTRONIK* (2007)

[Masters und Luschi 2018] Masters, Dominic ; Luschi, Carlo: Revisiting Small Batch Training for Deep Neural Networks. In: *CoRR* abs/1804.07612 (2018). – URL http://arxiv.org/abs/1804.07612

[Meisel 2017] Meisel, A.: *Recurrent-Neural-Networks*. Vorlesungsskript 'Modellierung Dynamischer Systeme'. 2017

[Meyer 2018a] Meyer, Manuel: Klassifizierung Multidimensionaler Zeitreihen mithilfe von Deep Learning. In: *2018 Master Informatik Grundprojekt (minf'18)*, 2018, S. 1–36

[Meyer 2018b] Meyer, Manuel: Klassifizierung Multidimensionaler Zeitreihen mithilfe von Deep Learning. In: *2018 Master Informatik Hauptseminar (minf'18)*, 2018, S. 1–17

[Moder 2011] Moder, Thomas: *Wesen und Nutzen inertialer MEMS Sensoren in der Fahrzeugnavigation*, Technische Universität Graz, Diplomarbeit, 12 2011. – Zuletzt besucht am 24.09.2019

[Ozyagcila 2015] Ozyagcila, Talat: Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference. (2015). – URL https://www.nxp.com/docs/en/application-note/AN4246.pdf. – Zuletzt besucht am 05.06.2019

[Ozyagcilar 2015] Ozyagcilar, Talat: Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensor. In: *NXP Freescale Semiconducto* (2015). – URL https://cache.freescale.com/files/sensors/doc/app_note/AN4248.pdf/. – Zuletzt besucht am 19.07.2019

[Park und Suh 2010a] Park, Sang ; Suh, Young: A Zero Velocity Detection Algorithm Using Inertial Sensors for Pedestrian Navigation Systems. In: *Sensors (Basel, Switzerland)* 10 (2010), 10, S. 9163–78

[Park und Suh 2010b] Park, Sang ; Suh, Young: A Zero Velocity Detection Algorithm Using Inertial Sensors for Pedestrian Navigation Systems. In: *Sensors (Basel, Switzerland)* 10 (2010), 10, S. 9163–78

[Reina u. a. 2018] Reina, Santiago C. ; Solin, Arno ; Rahtu, Esa ; Kannala, Juho: ADVIO: An authentic dataset for visual-inertial odometry. In: *CoRR* abs/1807.09828 (2018). – URL http://arxiv.org/abs/1807.09828

[Ruotsalainen u. a. 2010] Ruotsalainen, L. ; Kuusniemi, H. ; Chen, R.: Overview of methods for visual-aided pedestrian navigation. In: *2010 Ubiquitous Positioning Indoor Navigation and Location Based Service*, Oct 2010, S. 1–10

[Sak u. a. 2014] Sak, H. ; Senior, Andrew ; Beaufays, F.: Long short-term memory recurrent neural network architectures for large scale acoustic modeling. (2014), 01, S. 338–342

[Schubert u. a. 2018] Schubert, David ; Goll, Thore ; Demmel, Nikolaus ; Usenko, Vladyslav C. ; Stückler, Jörg ; Cremers, Daniel: The TUM VI Benchmark for Evaluating Visual-Inertial Odometry. In: *CoRR* abs/1804.06120 (2018). – URL http://arxiv.org/abs/1804.06120

[Sietsma und Dow 1991] Sietsma, Jocelyn ; Dow, Robert J. F.: Creating Artificial Neural Networks That Generalize. In: *Neural Netw.* 4 (1991), Januar, Nr. 1, S. 67–79. – URL http://dx.doi.org/10.1016/0893-6080(91)90033-2. – ISSN 0893-6080

[Smith u. a. 2017] Smith, Samuel L. ; Kindermans, Pieter-Jan ; Le, Quoc V.: Don't Decay the Learning Rate, Increase the Batch Size. In: *CoRR* abs/1711.00489 (2017). – URL http://arxiv.org/abs/1711.00489

[Smits und Ballato 1994] Smits, J. G. ; Ballato, A.: Dynamic admittance matrix of piezoelectric cantilever bimorphs. In: *Journal of Microelectromechanical Systems* 3 (1994), Sep., Nr. 3, S. 105–112

[Solin u. a. 2017] Solin, Arno ; Reina, Santiago C. ; Rahtu, Esa ; Kannala, Juho: Inertial Odometry on Handheld Smartphones. In: *CoRR* abs/1703.00154 (2017). – URL http://arxiv.org/abs/1703.00154

[Tille 2016] Tille, T.: *Automobil-Sensorik: Ausgewählte Sensorprinzipien und deren automobile Anwendung*. Springer Berlin Heidelberg, 2016. – ISBN 9783662489444

[Titterton u. a. 2004] Titterton, D. ; Weston, J.L. ; Weston, J. ; Electrical Engineers, Institution of ; Aeronautics, American I. of ; Astronautics: *Strapdown Inertial Navigation Technology*. Institution of Engineering and Technology, 2004 (Electromagnetics and Radar Series). – ISBN 9780863413582

[Vezocnik und Juric 2019] Vezocnik, M. ; Juric, M. B.: Average Step Length Estimation Models Evaluation Using Inertial Sensors: A Review. In: *IEEE Sensors Journal* 19 (2019), Jan, Nr. 2, S. 396–403. – ISSN 2379-9153

[Wang u. a. 2016] Wang, S. ; Wen, H. ; Clark, R. ; Trigoni, N.: Keyframe based large-scale indoor localisation using geomagnetic field and motion pattern. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, S. 1910–1917

[Waxenegger-Wilfing u. a. 2019] Waxenegger-Wilfing, Günther ; Dresia, Kai ; Deeken, Jan ; Oschwald, Michael: *Neuronale Netze für den Entwurf und Betrieb von Flüssigraketentriebwerken*. 2019. – URL https://www.dglr.de/fileadmin/inhalte/dglr/fb/q3/veranstaltungen/Q34_2019_KI/DGLR_Q34_2019_Abstrakt_Waxenegger_Wilfing_raketentriebwerke.pdf

[Wendel 2011] Wendel, Jan: *Integrierte Navigationssysteme: Sensordatenfusion, GPS und Inertiale Navigation*. 2. Oldenbourg Verlag München, 2011. – ISBN 9783486704396

[Wild-Pfeiffer 2015] Wild-Pfeiffer, Franziska: *DGK: Veröffentlichungen B: Angewandte Geodäsie*. Bd. Heft Nr. 319 : Reihe B, Angewandte Geodäsie, Elektronische Ressource: *Das Potenzial von MEMS-Inertialsensoren zur Anwendung in der Geodäsie und Navigation*. München : Verlag der Bayerischen Akademie der Wissenschaften in Kommission beim Verlag C.H. Beck, 2015. – URL http://publikationen.badw.de/de/044436215. – ISBN 978-3-7696-8598-5

[Wild-Pfeiffer und Schäfer 2011] Wild-Pfeiffer, Franziska ; Schäfer, Bernhardt: *MEMS-Sensoren, auch für die Geodäsie*. 1. Zeitschrift für Vermessungswesen, 1 2011. – (zfv 1/2011)

[Willemsen 2016] Willemsen, Thomas: *Fusionsalgorithmus zur autonomen Positionsschätzung im Gebäude, basierend auf MEMS-Inertialsensoren im Smartphone*. The address of the publisher, HafenCity Universität Hamburg, Dissertation, 6 2016. – Zuletzt besucht am 24.09.2019

[Wilson und Martinez 2003] Wilson, D. R. ; Martinez, Tony R.: The General Inefficiency of Batch Training for Gradient Descent Learning. In: *Neural Netw.* 16 (2003), Dezember, Nr. 10, S. 1429–1451. – URL http://dx.doi.org/10.1016/S0893-6080(03)00138-2. – ISSN 0893-6080

[Yan u. a. 2017] Yan, Hang ; Shan, Qi ; Furukawa, Yasutaka: RIDI: Robust IMU Double Integration. In: *CoRR* abs/1712.09004 (2017). – URL http://arxiv.org/abs/1712.09004

[Yan u. a. 2018] Yan, J. ; He, G. ; Basiri, A. ; Hancock, C.: Vision-aided indoor pedestrian dead reckoning. In: *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2018, S. 1–6

[Yanghuan u. a. 2014] Yanghuan, L. ; Yang, G. ; Qian, S. ; Ming, M.: Trajectory calibration approach using a flexible particle filter for PNS. In: *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Oct 2014, S. 19–23

[Zhang u. a. 2017] Zhang, W. ; Sengupta, R. ; Fodero, J. ; Li, X.: DeepPositioning: Intelligent Fusion of Pervasive Magnetic Field and WiFi Fingerprinting for Smartphone Indoor Localization via Deep Learning. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2017, S. 7–13

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. November 2019 Manuel Meyer