

Map matching methods for assigning vehicle positions to road networks

Adrian Wöltche



University
of Bamberg
Press

51 Schriften aus der Fakultät Wirtschaftsinformatik
und Angewandte Informatik der Otto-Friedrich-
Universität Bamberg

Contributions of the Faculty Information Systems
and Applied Computer Sciences of the
Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik
und Angewandte Informatik der Otto-Friedrich-
Universität Bamberg

Contributions of the Faculty Information Systems
and Applied Computer Sciences of the
Otto-Friedrich-University Bamberg

Band 51

Map matching methods for assigning vehicle positions to road networks

Adrian Wöltche

Bibliographische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de/> abrufbar.

Diese Arbeit hat der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg als Dissertation vorgelegen.

1. Gutachter: Prof. Dr. Andreas Henrich, Otto-Friedrich-Universität Bamberg

2. Gutachter: Prof. Dr. Richard Göbel, Hochschule Hof

Tag der mündlichen Prüfung: 14.07.2026

Dieses Werk ist als freie Onlineversion über das Forschungsinformationssystem (FIS; fis.uni-bamberg.de/) der Universität Bamberg erreichbar. Das Werk – ausgenommen Cover, Zitate und Abbildungen – steht unter der CC-Lizenz CC BY.



Lizenzvertrag: Creative Commons Namensnennung 4.0
<https://creativecommons.org/licenses/by/4.0>

Herstellung und Druck: docupoint, Magdeburg

Umschlaggestaltung: University of Bamberg Press

Umschlaggraphik: Adrian Wöltche

University of Bamberg Press, ubp@uni-bamberg.de, Bamberg 2026

 <https://ror.org/004fa0x02>

ISSN: 1867-7401 (Print)

eISSN: 2750-8560 (Online)

ISBN: 978-3-98989-134-0 (Print)

eISBN: 978-3-98989-135-7 (Online)

URN: [urn:nbn:de:bvb:473-irb-116612x](https://nbn-resolving.org/urn:nbn:de:bvb:473-irb-116612x)

DOI: <https://doi.org/10.20378/irb-116612>

Zusammenfassung

Map Matching ist eine weit verbreitete Technologie zur Behebung von Diskrepanzen zwischen Geokoordinatensequenzen aus Bewegungen und Straßennetzen. Während der Bewegung können Geokoordinaten mithilfe von globalen Satellitennavigationssystemen gemessen werden. Aufgrund der Beschaffenheit dieser Systeme enthalten die gemessenen Geopositionen jedoch erhebliche Störsignale. Infolgedessen treten in der Regel Diskrepanzen zwischen den gemessenen Positionen und den zugrunde liegenden Straßen auf, auf denen die Bewegung ursprünglich stattfand.

Map Matching ist eine Kombination aus definierten Modellen, Algorithmen und Metriken zur Korrektur dieser Diskrepanzen. Das Ziel von Map Matching ist es, Routen innerhalb des Straßennetzes zu finden, die die aufgezeichneten Bewegungen am genauesten wiedergeben. Die Ergebnisse des Map Matching decken sich exakt mit dem Straßennetz, sind frei von dem ursprünglichen Rauschen und wurden hinsichtlich unmöglicher Umwege und Lücken aus den gemessenen Bewegungen korrigiert. Diese Ergebnisse erleichtern weitere Analysen im Bereich der Mobilitätsforschung.

In dieser Arbeit stellen wir eine neuartige Methode zum Map Matching vor, die auf Markov-Entscheidungsprozessen und Algorithmen des verstärkenden Lernens basiert. Unsere neuartige Methode führt verschiedene Verbesserungen in mehreren Aspekten des Map-Matching-Prozesses ein, um die Genauigkeit unserer Ergebnisse im Vergleich zu bestehenden Lösungen zu erhöhen. Unser Prozess umfasst einen verbesserten Vorverarbeitungsschritt zur Vereinfachung der Trajektorien, um die Matching-Leistung sowohl in Bezug auf die Genauigkeit als auch auf die Geschwindigkeit zu verbessern, eine neuartige Funktion zur Kandidatenauswahl, die es dem

stochastischen Prozess ermöglicht, Ausreißer und Rauschcluster dynamisch aufzulösen, eine neue Metrik, die Richtungsänderungen in Routen bewertet, um Umwege und Rundfahrten in den Ergebnissen zu reduzieren, und schließlich einen neuen Vergleichsalgorithmus, der Ergebnisse aus verschiedenen Map-Matching-Lösungen mit unterschiedlicher numerischer Präzision der Koordinaten bewerten kann.

Unsere neuartige Map-Matching-Lösung wird als gebrauchsfertige Open-Source-Software bereitgestellt und wir vergleichen unsere Lösung mit mehreren Open-Source-Lösungen von Drittanbietern hinsichtlich Rechengeschwindigkeit, Ressourcenverbrauch und Genauigkeit der Ergebnisse. Um eine hohe Leistung zu erzielen, enthält unsere Open-Source Referenzimplementierung mehrere spezifische Datenstrukturen, algorithmische Anpassungen und Software-Design-Entscheidungen, die wir im jeweiligen Kontext erläutern.

Für unseren umfangreichen Benchmark verwenden wir mehrere vorhandene Map-Matching-Datensätze, die eine Mischung aus Ground-Truth und handkorrigierten Lösungen enthalten. Darüber hinaus stellen wir einen neuartigen Open-Source-Prozess zur Generierung künstlicher Ground-Truth-Map-Matching-Daten vor und analysieren diesen. Unsere Evaluation zeigt, dass unsere neuartige Map-Matching-Methode robuster gegenüber Rauschen und Ausreißern ist als bestehende Methoden und diese in Bezug auf Genauigkeit und Rechenleistung übertrifft.

Unser neuer Map-Matching-Ansatz umfasst mehrere Schlüsselkonzepte, die einzeln auf andere Lösungen übertragen werden können. Darüber hinaus liefert unser neuer Ansatz weitere Ideen für die zukünftige Forschung, von denen wir einige in dieser Arbeit vorstellen.

Abstract

Map matching is a widely used technology for resolving discrepancies between sequences of geocoordinates from movements and road networks. During movement, geocoordinates can be measured with the help of global navigation satellite systems. However, due to the nature of these systems, the measured geopositions contain considerable noise. Consequently, discrepancies generally occur between the measured positions and the underlying roads on which the movement originally occurred.

Map matching is a combination of defined models, algorithms, and metrics for correcting these discrepancies. The goal of map matching is to find routes within the road network that most accurately represent the recorded movements. The results of map matching exactly overlap with the road network, are free from the original noise, and are corrected concerning impossible detours and gaps from the measured movements. These results facilitate further analyses in the domain of mobility research.

In this thesis, we present a novel map matching method based on Markov decision processes and reinforcement learning algorithms. Our novel method introduces various advances in multiple aspects of the map matching process pipeline for increasing the accuracy of our results in comparison to existing solutions. Our process includes an improved trajectory simplification preprocessing step for improving matching performance both in terms of accuracy and speed, a novel candidate adoption feature that enables the stochastic process to dynamically resolve outliers and noise clusters, a new metric that evaluates direction changes in routes for reducing detours and round trips in the results, and finally, a new comparison algorithm that can evaluate results from different map matching solutions with varying numerical coordinate precision.

Our novel map matching solution is provided as ready-to-use open source software, and we compare our solution with multiple third-party open source solutions in terms of computational speed, resource consumption, and accuracy of the results. For achieving high performance, our open source reference implementation contains several specific data structures, algorithmic customizations, and software design decisions, that we explain in context.

For our extensive benchmark, we use multiple existing map matching data sets, with a mixture of ground truth and hand-corrected solutions. Furthermore, we introduce and analyze a novel open source process for generating artificial ground truth map matching data. Our evaluation demonstrates that our novel map matching method is more robust to noise and outliers than existing methods and that it outperforms them in terms of accuracy and computational performance.

Our new map matching approach contains several key concepts that can be individually transferred to other solutions. Additionally, our new approach generates further ideas for future research, some of which we present in this work.

Contents

1	Introduction	1
1.1	Map Matching Basics	1
1.2	Research Questions	3
1.3	Previous Publications	4
1.4	Open Source Code References	5
1.5	Thesis Organization	5
1.6	Term Definitions	7
1.7	Map Matching Use Cases	12
1.8	Map Matching Challenges	13
1.9	Solution Approach	16
1.10	Scientific Contribution	17
2	Related Work	21
2.1	Global Navigation Satellite Systems	21
2.1.1	GNSS Accuracy	21
2.1.2	GNSS Error Dimension	22
2.1.3	GNSS Autocorrelation	23
2.1.4	GNSS Simulation	25
2.2	Map Matching Models	25
2.2.1	Historic Methods	26
2.2.2	Stochastic Methods	26
2.2.3	Conditional Random Fields	27
2.2.4	Stochastic Model Limitations	28
2.2.5	Online and Offline Map Matching	29
2.2.6	Recent Literature Reviews	30
2.2.7	Further Research Directions	30
2.3	Map Matching Base Algorithms	31
2.3.1	Map Matching Candidates	31

2.3.2	Shortest Path Routing	32
2.3.3	Missing Data Treatment	33
2.3.4	Trajectory Simplification	34
2.3.5	Reinforcement Learning	35
2.4	Map Matching Evaluation	36
2.4.1	Map Matching Data Sets	36
2.4.2	Map Matching Accuracy Metrics	38
2.4.3	State-of-the-Art Implementations	41
2.4.4	Traffic Simulation	43
2.4.5	Floating Car Data Simulation	43
2.5	Related Work Conclusion	44
3	Method	45
3.1	Novel Map Matching Method	45
3.1.1	Trajectory Simplification	46
3.1.2	Candidate Search	50
3.1.3	Candidate Adoption	52
3.1.4	Comparison Algorithm	55
3.1.5	Hidden Markov Models and Algorithms	57
3.1.6	Markov Decision Process and Algorithms	59
3.1.7	Map Matching Metrics	64
3.2	Artificial Ground Truth Generation	69
3.2.1	Artificial GNSS Noise	71
3.2.2	Noise Distribution Comparison	73
3.2.3	Noise Characteristics Validation	77
4	Implementation	83
4.1	Development Goals	83
4.1.1	Road Network Design	84
4.1.2	Programming Languages	86
4.1.3	Spatial Scaling	89
4.2	Essential Software Design Principles	98
4.2.1	Graph Models and Algorithms	98
4.2.2	Candidate Search Algorithms	113
4.2.3	Stochastic Process Performance	116
4.3	Memory Access Optimizations	118
4.4	Code Organization and Transferability	125

5	Evaluation	129
5.1	Default Settings and Algorithms	129
5.2	Default Metrics Weights	132
5.3	Experiment Framework Conditions	133
5.4	Real-World Data Sets	137
5.4.1	Seattle Data Set	141
5.4.2	Melbourne Data Set	146
5.4.3	GIS CUP Data Set	148
5.4.4	Worldwide Data Set	151
5.4.5	Hof Data Set	153
5.5	Artificial Ground Truth Data Scenarios	160
5.5.1	Hypertuned Scenario	160
5.5.2	Baseline Tracks	163
5.5.3	Error Scenario	167
5.5.4	Outlier Scenario	168
5.5.5	Sparse Scenario	170
5.5.6	Missing Points Scenario	171
6	Discussion	175
6.1	Answers to Research Questions	176
6.2	General Applicability of Solutions	177
6.3	Limitations and Future Research	178
6.4	Conclusion	181
	Bibliography	183

List of Figures

1.1	Map Matching Float Map	2
1.2	Didactic Mismatched Track	14
1.3	Didactic Correctly Matched Track	14
1.4	Map Matching Tools Result Comparison	15
3.1	Map Matching Process Overview	46
3.2	Spatial Duplicates Removal	47
3.3	Warp Detection	47
3.4	Douglas Peucker Algorithm	48
3.5	Median-Merge Algorithm	49
3.6	Candidate Search	51
3.7	Candidate Adoption	53
3.8	Candidate Adoption Levels	54
3.9	Comparison Algorithm Steps	56
3.10	Hidden Markov Model for Map Matching	58
3.11	Markov Decision Process for Map Matching	60
3.12	Map Matching Metrics	64
3.13	Artificial Ground Truth Generation Process	70
3.14	Artificial GNSS Gaussian Normal Noise	74
3.15	Artificial GNSS ARMA Noise	75
3.16	Artificial GNSS Ornstein-Uhlenbeck Process Noise	76
3.17	Real and Artificial GNSS Distance Distribution	78
3.18	Real and Artificial GNSS Static Position	79
3.19	Real and Artificial GNSS Coordinates	80
3.20	Real and Artificial GNSS Autocorrelation Functions	81
4.1	Sub-Partial Candidate Lines	86
4.2	Map Administrative Boundaries	91

4.3	Full Memory Mapping Prepare Memory Profiling	95
4.4	Output Memory Mapping Prepare Memory Profiling . . .	96
4.5	In-Memory Prepare Memory Profiling	96
4.6	Memory Mapping Matching Memory Profiling	97
4.7	In-Memory Matching Memory Profiling	97
4.8	Road Network Topology Simplification	108
5.1	Good Map Matching Results	159
5.2	Artificial Hypertuned Float Map	162
5.3	Artificial Noise Baseline Scenarios	164
5.4	Artificial Noise Error Scenarios	167
5.5	Artificial Outlier Error Scenarios	169
5.6	Artificial Sparse Error Scenarios	170
5.7	Artificial Missing Points Error Scenarios	172
6.1	Bad Map Matching Results	181

List of Tables

2.1	Open Source Software Tools Overview	41
2.2	Open Source Software Tools Models and Algorithms . . .	42
4.1	Road Network Metadata	92
4.2	Memory Mapping Prepare Benchmark	92
4.3	Preparation Times Benchmark	94
4.4	Routing Distance Limit Benchmark	100
4.5	Shortest Path Memory Benchmark	101
4.6	Shortest Path Early Stopping Benchmark	102
4.7	Dijkstra and A-star Algorithm Benchmark	103
4.8	Dijkstra and A-star Distance Limit Benchmark	104
4.9	Route Referencing and Copying Benchmark	106
4.10	Edge Weight Precomputation Benchmark	107
4.11	Spatial Reference System Benchmark	107
4.12	Road Network Topology Simplification Benchmark	109
4.13	Road Network Topology Simplification Metadata	110
4.14	Adjacency List and Compressed Sparse Row Benchmark .	112
4.15	R-tree Packing and R-star Insertion Benchmark	112
4.16	Candidate Search Modes and Radius Benchmark	115
4.17	MDP Beneficial-Preordering Benchmark	118
4.18	MDP Index-Mapping Benchmark	118
4.19	Edge Precomputation Benchmark	119
4.20	Cache Layers Benchmark	121
4.21	Memory Allocation and Multi-Threading Benchmark . . .	124
4.22	General Code Applicability and Transferability	127
5.1	Tool Versions and Coordinate Systems	130
5.2	Tool Metrics and Advanced Functionality	131

5.3	Tool Candidate Search and Default Settings	131
5.4	Optimized Metric Weights	132
5.5	Map Matching Data Sets	138
5.6	Map Matching Data Set Road Network Metadata	141
5.7	Map Matching Data Set Metadata	142
5.8	Seattle Data Set Prepare Benchmark Results	143
5.9	Seattle Data Set Match Benchmark Results	144
5.10	Melbourne Data Set Prepare Benchmark Results	146
5.11	Melbourne Data Set Match Benchmark Results	147
5.12	GIS CUP Data Set Prepare Benchmark Results	149
5.13	GIS CUP Data Set Match Benchmark Results	150
5.14	Worldwide Data Set Prepare Benchmark Results	152
5.15	Worldwide Data Set Match Benchmark Results	153
5.16	Hof Data Set Prepare Benchmark Results	154
5.17	Hof Data Set Match Benchmark Results	155
5.18	Hof Data Set MM2 Comparison Benchmark Results	157
5.19	Hof Data Set MM2 Comparison Metadata	158
5.20	Artificial Ground Truth Data Generation Parameters	161
5.21	Artificial Hypertuned Benchmark Results	163
5.22	Artificial Sparse Baseline Benchmark Results	165
5.23	Artificial Noise Error Benchmark Results	168
5.24	Artificial Outlier Error Benchmark Results	169
5.25	Artificial Sparse Benchmark Results	171
5.26	Artificial Missing Points Benchmark Results	173

List of Definitions

1.1	Definition (Point)	7
1.2	Definition (Line)	7
1.3	Definition (Segment)	7
1.4	Definition (Road Network)	7
1.5	Definition (Track)	8
1.6	Definition (Candidate)	8
1.7	Definition (Route)	8
1.8	Definition (Match)	9
1.9	Definition (Ground Truth)	9
1.10	Definition (Hidden Markov Model)	11
1.11	Definition (Markov Decision Process)	11
3.1	Definition (Median-Merge Algorithm)	48
3.2	Definition (Adaptive Circular Candidate Search)	51
3.3	Definition (Beneficial Preordering)	62
3.4	Definition (Distance)	65
3.5	Definition (Length)	65
3.6	Definition (Azimuth)	65
3.7	Definition (Direction Change)	66
3.8	Definition (Turn)	66
4.1	Definition (Distance Limit Factor)	99
4.2	Definition (Road Network Topology Simplification)	108
4.3	Definition (Index-Mapping)	116

List of Equations

2.1	Equation (Error Fraction)	38
2.2	Equation (Correct Fraction)	38
3.1	Equation (Observation Probability)	67
3.2	Equation (Transition Probability)	67
3.3	Equation (Reward)	69
3.4	Equation (Gaussian Normal Distribution)	72
3.5	Equation (Autoregressive Moving-Average Process)	72
3.6	Equation (Ornstein-Uhlenbeck Process)	73

List of Abbreviations

ACF	autocorrelation function
ARMA	autoregressive moving-average
CA	candidate adoption
CPU	central processing unit
CRF	conditional random field
CS	candidate search
DS-A	Seattle data set
DS-B	Melbourne data set
DS-C	GIS CUP data set
DS-D	Worldwide data set
DS-E	Hof data set
FCD	floating car data
FMM	Fast Map Matching
GNSS	global navigation satellite system
GPS	Global Positioning System
HMM	hidden Markov model
KNN	k -nearest neighbor
MDP	Markov decision process
MEM	memory
MM2	Map Matching 2
MMAP	memory mapping
OSM	OpenStreetMap
OSRM	Open Source Routing Machine
SRS	spatial reference system
STM	ST-Matching
SUMO	Simulation of Urban MObility
TS	trajectory simplification
UBODT	upper bounded origin-destination table

List of Math Symbols

e_f	error fraction after Equation 2.1
e_a	length of erroneously added parts in the match over the ground truth
e_m	length of erroneously missed parts in the match from the ground truth
c_f	correct fraction after Equation 2.2
l_c	length of correctly matched parts
l_g	length of ground truth or correct solution
l_m	length of match
r	reward defined as negative individually weighted mixed error sum after Equation 3.3
d_c	distances of candidates to points
l_t	length of track segment
l_r	length of route
δ	angle difference
a_t	azimuth of track segment
a_r	azimuth of route
adc_r	absolute direction changes of route
t_t	turn of track segment
t_r	turn of route
w_d	weight for distance term
w_l	weight for length term
w_a	weight for azimuth term
w_{adc}	weight for absolute direction changes term
w_t	weight for turn term

S	state space or hidden state space, $S = [s_1^1, s_1^2, s_2^1, \dots, s_n^m]$
O	observation space, $O = [o_1, o_2, o_3, \dots, o_n]$
T	time steps space, $T = [t_1, t_2, t_3, \dots, t_n]$
t_n	n -th time step $t \in T$
o_n	observation $o \in O$ at the n -th time step
s_n^m	state or hidden state $s \in S$ number m at the n -th time step
$\hat{\cdot}$	normalized expression
$\mathbb{P}_O$	observation probability between observations and states after Equation 3.1
$\mathbb{P}_S$	transition probability between state transitions after Equation 3.2
P	points space, $P = [p_1, p_2, p_3, \dots, p_n]$
C	candidates space, $C = [c_1^1, c_1^2, c_2^1, \dots, c_n^m]$
A_s	action space in state s
p_n	point $p \in P$ at the n -th time step
c_n^m	candidate $c \in C$ number m at the n -th time step
a	action $a \in A_s$
X	random variable after Equation 3.4, 3.5, and 3.6
$\mathcal{N}$	Normal distribution
$\sim$	‘is distributed as’
μ_0	zero (mean)
$\sigma_{\mathcal{N}}$	error (standard distribution)
φ_i	autoregressive terms
θ_i	moving-average terms
ε_t	white noise
o_p	autoregressive order
o_q	moving-average order
θ	rate of mean reversion
μ	long-term mean
σ	volatility
W	Wiener process (standard Brownian motion)
Δt	time step size

1 Introduction

The world is full of devices that recurrently track and record their geospatial positions. With the general availability of global navigation satellite systems (GNSSs), open data road networks, such as OpenStreetMap (OSM)¹, and the widespread distribution of smartphones and modern cars with integrated GNSS sensors, large amounts of geospatial tracks are collected. The systematic collection of such tracks recorded from car navigation systems is also called floating car data. Analyzing floating car data supports various use cases in the domain of mobility research, for example, traffic flow analysis. **Figure 1.1** shows a floating car data set of matched raw trajectories and a resulting float map representing the road usage. Map matching is used here to correct the noise in the given floating car data by mapping the raw trajectories onto the underlying road network.

1.1 Map Matching Basics

Map matching is a well-established and non-trivial technology for aligning geospatial point sequences, i.e., tracks, with road networks. One major challenge that has been repeatedly identified over many decades is the accuracy of the results. There are two reasons for this. First, map matching has inevitably become a complex optimization process due to the practically infinite number of ways geospatial tracks can be mapped onto road networks. Second, there is a general lack of available ground truth data to evaluate map matching algorithms in all situations. This means that we don't know which results are correct, and we can't verify them thoroughly.

¹<https://www.openstreetmap.org/copyright>

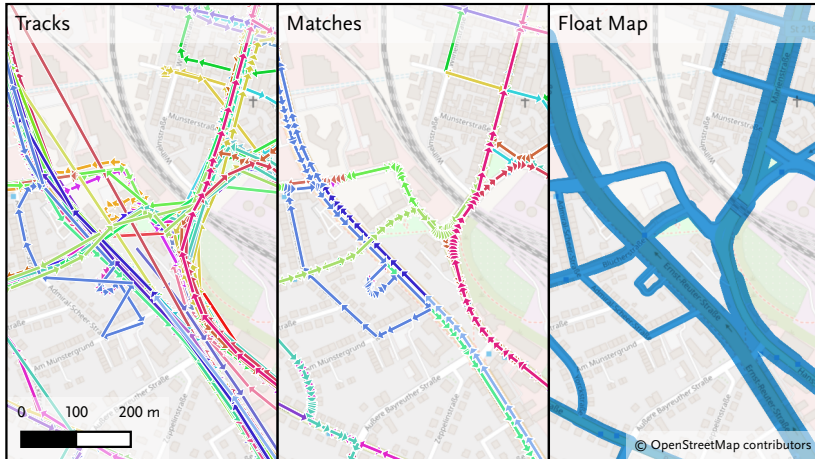


Figure 1.1: *Map Matching* – The left panel shows the original floating car data tracks, each represented by a line in a different color with an arrow on each line segment. The middle panel shows the matches from our novel map matching approach with equivalent colors. The right panel shows a float map generated from the stack size of the overlapping segments of these matches. The width of the line segments in the float map indicates the amount of traffic based on the matches. The background is a cartography image of OpenStreetMap¹ road network data.

Despite many recent advances in map matching accuracy, the issue remains unresolved after more than three decades of research in this domain. Additionally, the historical unavailability of public map matching implementations makes comparative evaluations problematic due to the complexity of the described algorithms. Most research articles from the past few decades do not provide open source algorithms. However, the availability of data and code has recently improved, and today, several open source map matching implementations, public data sets, and research articles exist. To support this development, we provide a complete and ready-to-use open source reference implementation of our novel map matching approach, an additional public data set with hundreds of anonymized recorded movements, and a technology pipeline for generating artificial map matching ground truth data with adjustable noise characteristics.

1.2 Research Questions

In this work, we attempt to answer the following research questions. We will delve deeper into the details in the subsequent sections.

1. Which methods can improve the robustness of map matching in challenging situations, particularly in the presence of noise and outliers?

This is the original question that inspired this work. We encountered many incorrect matching results with existing solutions in challenging situations, so we asked ourselves if the accuracy of map matching could be improved in these situations specifically as well as generally.

2. Can the stochastic process be made robust against outliers?

Map matching today is performed via stochastic optimization, but strong noise and outliers pollute the optimization model through the metrics. The question is whether this pollution can be avoided or circumvented in order to make the stochastic process more robust. Although stochastic processes minimize errors, outliers and strong noise can still lead to significant deviations. Providing an alternative path with lower error in such situations could lead to improved optimization.

3. Are there robust ways to resolve noise algorithmically?

If the stochastic process cannot be made robust enough, can we support it with additional preprocessing algorithms to reduce noise in the data? This could improve the model metrics by providing the stochastic process with cleaner data to optimize.

4. Can detours be penalized stochastically?

Regarding outliers, it has been shown in the literature that manual penalties improve results in such situations. However, manual intervention may only address issues above a specific threshold. Rather than manually penalizing according to a defined threshold, the idea is to introduce a metric into the model that penalizes any detour within the stochastic process.

5. **Can map matching ground truth data be generated artificially?**

One challenge in evaluating map matching performance is the limited availability of labeled data. As a result, it is difficult to determine whether our algorithms perform well in different situations. For this reason, we are considering generating artificial map matching ground truth data with customizable error settings to create various evaluation scenarios.

1.3 Previous Publications

This thesis reuses text, images, tables, and data from the following previously published open-access works:

1. Adrian Wöltche (Oct. 2021). *Evolving map matching with markov decision processes*. Hof. DOI: [10.57944/1051-119](https://doi.org/10.57944/1051-119)

In this work, we presented our initial ideas for solving map matching using a Markov decision process (MDP) model design. Using a prototype, we demonstrated the success of our concept.

2. Adrian Wöltche (Oct. 2023). “Open source map matching with Markov decision processes: A new method and a detailed benchmark with existing approaches”. In: *Transactions in GIS* 27.7, pp. 1959–1991. DOI: [10.1111/tgis.13107](https://doi.org/10.1111/tgis.13107)

This article describes our complete approach, which is based on an older version of our implementation that was evaluated against multiple existing solutions. We demonstrated that our approach outperforms others in terms of accuracy and performance in virtually all scenarios.

3. Adrian Wöltche (Jan. 2025). “Artificial Ground Truth Data Generation for Map Matching with Open Source Software”. In: *Tagungsband FOSSGIS-Konferenz 2025*. FOSSGIS2025. Zenodo, p. 17. ISBN: 978-3-00-081664-2. DOI: [10.5281/zenodo.14774143](https://doi.org/10.5281/zenodo.14774143)

In this conference article, we describe our approach of artificially generating map matching ground truth data. We generated and evaluated multiple scenarios with different noise characteristics. The results demonstrated that our approach was the most robust in comparison.

1.4 Open Source Code References

Our open source software reference implementation is available under AGPL v3.0² at GitHub:

<https://github.com/addy90/map-matching-2>

Prebuilt executables are available for Windows and Linux. Please review the provided README and help.txt for usage instructions.

The benchmark suite that was used to generate the results in this work is available under AGPL v3.0² at the University of Bamberg:

<https://doi.org/10.48564/unibafid-wt3wn-78r32>

1.5 Thesis Organization

This thesis is organized into multiple chapters that build upon each other. In this [chapter 1](#), we provide an introduction to map matching based on our previous publications and open source software reference implementations. After presenting the research questions that motivate this work and defining the necessary terms, we describe the ideas and use cases behind map matching and the challenges that arise from them. Subsequently, we present an outlook on our approach toward solving the map matching problem and addressing its challenges.

In [chapter 2](#), we cite related work and provide an overview of existing map matching solutions. We discuss GNSS errors as a primary source

²<https://www.gnu.org/licenses/agpl-3.0.en.html>

of inaccuracies in map matching and we describe various map matching models and algorithms. Additionally, we examine different real-world and artificial data sets and methods for evaluating map matching solutions.

On this foundation, in [chapter 3](#), we introduce our novel approach to solving map matching. Building on multiple refined base algorithms, including data preprocessing and preselection, we present our new map matching model as an evolution of the current leading model. Along with our new evaluation algorithm for comparing map matching results, we introduce a new method for generating artificial map matching ground truth data sets for evaluation purposes.

Our novel map matching model relies on various optimization technologies to perform well. In [chapter 4](#), we explain the essential software design principles and benchmark different data structures, algorithms, and optimizations that we incorporated into our reference implementation to promote their use in other map matching solutions. Additionally, many of our optimizations can be applied to related fields, such as routing engines, stochastic optimization, and general software.

Based on this evaluation, in [chapter 5](#), we compare our novel map matching reference implementation with multiple existing open source software solutions using various real-world and artificial data sets. We demonstrate that our map matching solution outperforms existing solutions in several aspects, notably accuracy, robustness, and computational performance relative to achieved accuracy.

Finally, in [chapter 6](#), we summarize our work and use our findings to answer the initial research questions. While many of our new and improved algorithms are applicable to other map matching solutions, we also discuss the limitations of our approach and provide ideas for future research to continue our work.

1.6 Term Definitions

In this thesis, we use several terms from the domain of map matching. For clarification purposes, we define them as follows:

Geometric Terms

Definition 1.1 (Point): A point is a geospatial coordinate with two dimensions. It contains an x and a y value. The unit depends on the corresponding spatial reference system. In a geographic spatial reference system, the x value represents the longitude and the y value represents the latitude.

Points are used, e.g., for representing GNSS measurements, such as GPS points. In this case, the spatial reference system is WGS-84. Points are also used for representing the geospatial locations of nodes and edges in a road network.

Definition 1.2 (Line): A line is a sequence of at least two points.

In the case of multiple connected points, we speak of a geometric line. Such lines are used for representing connected sequences of GNSS measurements and for representing edges in a road network, i.e., roads and streets.

Definition 1.3 (Segment): A segment is a line with exactly two points.

Segments are a special type of line with exactly two points. Any line with more than two points consists of more than one segment. For example, a line with five points contains four segments. Therefore, the length of a line is the sum of the lengths of its segments.

Definition 1.4 (Road Network): The road network for map matching is a directed graph consisting of nodes and edges connecting these nodes. As additional data, the nodes contain points, and the edges contain lines.

The road network forms the basis for map matching and serves as a reference. It is defined as a graph with additional geometric data. Each node is assigned a point, and each edge is assigned a line. An edge can be a straight line between two nodes, i.e., a segment, or it can contain a more complex line, such as a curved road (approximated by many small segments) between two junctions. The graph is directed, meaning the edges have a direction from one node to another. This resembles the driving direction in practice, i.e., one-way or two-way roads. In a directed graph, roads that can be traveled in both directions are represented by two directed edges between the adjacent nodes with opposing directions.

Definition 1.5 (Track): A track is a line consisting only of recorded points. It represents the movement of a GNSS tracking device.

A track is a special line that represents a recorded movement. Sometimes, tracks are also referred to as ‘trajectories’ in literature. Tracks are usually recorded by tracking devices such as smartphones, modern vehicles, or GNSS trackers. We use the term ‘track’ to distinguish the recorded track from the underlying roads on which the movement originally took place. A track generally does not overlap with the road network, but it is close to its nodes and edges. The goal is to correctly map it onto the road network.

Definition 1.6 (Candidate): A candidate in the domain of map matching is a possible road position to which a track point can be mapped.

In map matching, for each point of a recorded track, a search for possible candidates in the road network is performed. Usually, multiple candidates per point are found initially. The group of candidates found for each point of a track is also called a ‘candidate set’. Map matching then selects, for each track point, one candidate from each corresponding candidate set, so that each track point can be mapped to a road position.

Definition 1.7 (Route): A route is a line between two candidates that lies exactly within the road network. A route can also exist between candidates with the same geospatial coordinates, in which case no movement occurs.

We use the term ‘route’ for describing lines that lie exactly within the road network, in order to distinguish them from ‘tracks’. Routes typically consist of multiple connected edges, though they can also be just a part of an edge. The only requirement for a route is that it originates from the coordinates of the road network. Since routes are computed between candidates, this is always true in our case. In practice, there are generally multiple candidates for each track point. Thus, for each track segment between adjacent track points, a cross-product of candidate origins and destinations is available. For each entry in the cross-product table, a route can be computed. Therefore, there are many alternative routes for each track segment available for evaluation.

Definition 1.8 (Match): A match is the final result of a map matching algorithm. It consists of one or more routes that are concatenated into one final route, which we call the match. It also lies exactly within the road network.

The match is one of the best concatenations of one or multiple routes, i.e., joined into one complete route, according to the defined metrics and models. In case a track only consists of two points, i.e., one segment, only one of the best routes representing the single track segment is sufficient. However, since tracks generally have more than two points, i.e., more than one segment, multiple routes for all track segments have to be concatenated into one match. If there are multiple best alternatives with the same score, only one is selected as the match, and the others are discarded, because they are equally optimal. Thus, the match is a most plausible representation of the recorded track in the underlying road network. Ultimately, a match is extracted from the final selection of candidates, meaning a match routes through and connects all selected candidates.

Definition 1.9 (Ground Truth): Ground truth in the domain of map matching is the actually correct match according to the defined road network.

The term ‘ground truth’ requires special attention in the domain of map matching because it is defined differently here than in the general domain of geographic information science. In the latter case, ‘ground truth’ refers to the true position in the world. However, map matching does not recover the ‘true position’ of GNSS measurements, but rather maps them onto a given road network. That *digital* road network itself generally has discrepancies with the real-world road network. For example, curves are usually approximated by many small segments, which cannot perfectly represent true curves in the real world. Additionally, junction and intersection points, as well as lanes of roads, in the digital road network may not lie in the exact locations of their real-world counterparts. The digital road network is a model of the real world and generally does not represent true positions exactly. For instance, OpenStreetMap roads are sometimes derived from satellite images or self-recorded GNSS tracks and may be corrected or altered multiple times.

In the domain of map matching, this is usually not an issue, because the use cases employing map matching generally can tolerate these typically small discrepancies between the road network and the real world. The assumption is that the road network is often measured much more accurately, i.e., with professional equipment, than movements are tracked, e.g., with smartphones. As such, mapping tracks onto the road network is generally assumed to be more beneficial than the other way round. However, this is not necessarily the case when the road network is incorrect, which is a different topic also called ‘map generation’, where a map is derived from aggregations of many movements.

As such, ‘ground truth’ in the domain of map matching is the correct match *in combination with* the specific road network that the match lies on or is derived from. However, we only consider a match to be ‘ground truth’ if it is the *actually* correct solution within the defined road network. There are many hand-corrected map matching solutions for which the authors do

not know the correct solution, but only *assume* it. This is *not* ‘ground truth’ by our definition, because the *correct* solution remains unknown. In these cases, we talk about ‘hand-corrected’ map matching data, i.e., providing supposedly correct matches. ‘Ground truth’ as defined in this work is only valid if the route driven in the road network is *known* and *correct* for the given road network. We will discuss this topic again later on multiple real-world data sets in [section 5.4](#).

Stochastic Models

We also need to define two stochastic model terms in advance, since they will be used frequently in the following chapters.

Definition 1.10 (Hidden Markov Model): A hidden Markov model (HMM) is defined by an *observation* $o_n \in O$ of zero or more *hidden states* $s_n^m \in S$ at every time step $t_n \in T$. Each observation of a hidden state has an *observation probability* $\mathbb{P}_O$, and each transition from one hidden state to another has a *transition probability* $\mathbb{P}_S$.

In map matching, a HMM is built from the track points representing the observations and the candidates representing the hidden states. It is possible that an observation has no assigned hidden states when no candidates are found for a given track point. A diagram describing the HMM for map matching is presented later in [Figure 3.10](#).

Definition 1.11 (Markov Decision Process): A Markov decision process (MDP) is defined by a *state space* S and an *action space* A_s that depends on each specific *state* $s \in S$. Thus, in each state s an *action* $a \in A_s$ can be selected. When an action a is selected in state s , the MDP transitions to the next state $s' \in S$ and returns a *reward* r .

In our novel map matching approach, we define a state $s = s(p_n, c_n^m, p_{n+1})$ by the current track point p_n at time step $t_n \in T$, the current candidate c_n^m and the next track point p_{n+1} at time step $t_{n+1} \in T$. After selecting an

action $a \in A_s$ which points to the candidate c_{n+1}^a , the state transitions to $s' = s(p_{n+1}, c_{n+1}^a, p_{n+2})$ and returns a reward for the transition. A diagram presenting our novel map matching MDP is later described in [Figure 3.11](#).

1.7 Map Matching Use Cases

Map matching is sometimes divided into two separate topics: Online and offline map matching (Quddus et al. 2007). Online map matching refers to real-time map matching during navigation. It involves mapping measured geopositions onto the road network or planned route while navigating along it. Online map matching enables a routing device, for example, to determine whether a user is still following the suggested route or has deviated from it (Xi et al. 2007). In that case, adjustments to the suggested route can be computed and presented. Additionally, large-scale online map matching can detect real-time traffic, especially congestion, such as stop-and-go traffic patterns (Li et al. 2015). Such data can be transferred to other navigating individuals to suggest alternative routes for avoiding traffic jams in real time.

In contrast, offline map matching involves mapping a track after it has been fully recorded. There are various use cases for this, some of which are described in Huang et al. (2021), such as detecting abnormal driving behavior or taxi passenger points of interests. The most straightforward use case for offline map matching is aligning individually recorded tracks onto the road network, e.g., from excursions, trips, or other movements. This can satisfy the individual interest of mapping and analyzing personal records. Analyzing individual matches can also provide insight into the accuracy of the underlying road network (Zhao et al. 2011). For example, this can be used to verify the speed tags of digital roads, or to identify map errors, such as missing connections or edges in the road network, by looking for large matching discrepancies. Analyzing turns can reveal illegal turning behavior. For example, cars or bicycles may turn in situations where there are turn restrictions in reality or in the digital road network. Such analyses can be used to improve the quality of digital road networks

or suggest practical adjustments in the real-world (Karich 2015). On a side note, during our work, we actually found and corrected a missing turn connection in the OpenStreetMap road network while analyzing the results of our novel map matching method.

The large-scale analysis of many tracks, e.g., floating car data, allows for various analyses in the domain of mobility research, some of which are described in Sun (2022). Since floating car data contains the trajectories of many movements, it can provide insight into general mobility patterns in the covered areas. Ideas for mapping floating car data include, e.g., creating float maps, such as in Figure 1.1, or directed traffic flow maps. Information deduced from these maps, such as high traffic density at specific locations, high road utilization, or repeating movement patterns between specific points (Jin et al. 2017), can support city and transportation planning. For example, travel time analysis (Rahmani 2015) can be used for public transportation planning such as determining the locations of stops and routes, or for road construction planning. Furthermore, floating car data of corporate vehicle fleets can support fleet control (Charanaraj et al. 2019) or goods tracking to provide insights into travel time optimization.

There may be additional use cases or ideas for map matching that have not yet been mentioned. Sometimes, an intermediate solution lays the groundwork for future work, so the list of use cases is not meant to be exhaustive.

1.8 Map Matching Challenges

One of the biggest challenges in map matching is dealing with uncertainty, such as GNSS noise and outliers. For each measured GNSS point, there can be multiple candidates in the road network, and it is generally unclear which one is the correct choice.

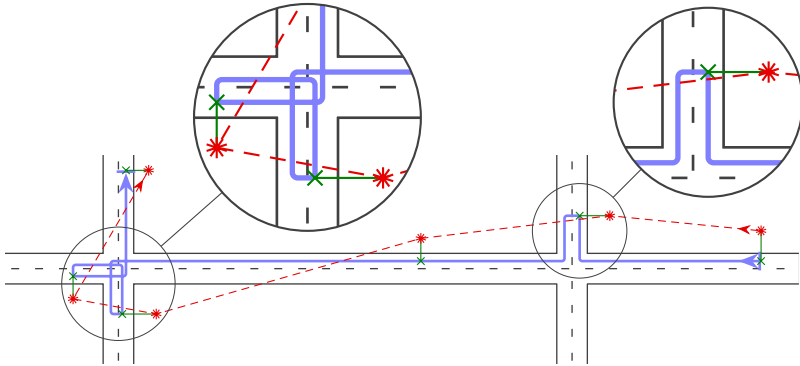


Figure 1.2: *Didactic example of a mismatched recorded track* – The red stars mark the recorded measurements. They are connected by the dashed red line, which represents the recorded track. The green lines project each recorded measurement to the matched road positions (i.e., the candidates, marked by green cross-marks). The light-blue line shows the matched route (i.e., the match, visually shifted at the crossroads for better visibility of direction) resulting from connecting all routes between all subsequent pairs of matched road positions in order.

In Figure 1.2 we demonstrate a didactic example of a track that is matched for every point to its corresponding nearest road position. This naive map matching approach results in multiple U-turns at the intersections.

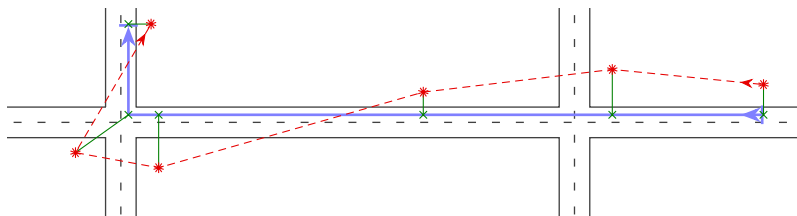


Figure 1.3: *Didactic example of a correctly matched track* – The red recorded measurements are matched to the most probable road positions (i.e., the candidates, marked by the green cross-marks). This results in the light-blue line showing the correctly matched route.

In Figure 1.3, the same didactic track is mapped to the hypothetical correct positions using a more complex map matching approach. This example demonstrates that accurate map matching is an optimization problem

requiring the evaluation of multiple alternatives to find an optimal solution. An ‘optimal’ solution is one that scores highest according to defined metrics. Such metrics can, for example, penalize U-turns in favor of straight routes. This enables map matching to identify matches that more accurately reflect actual movement patterns, as approximately represented by the recorded tracks.

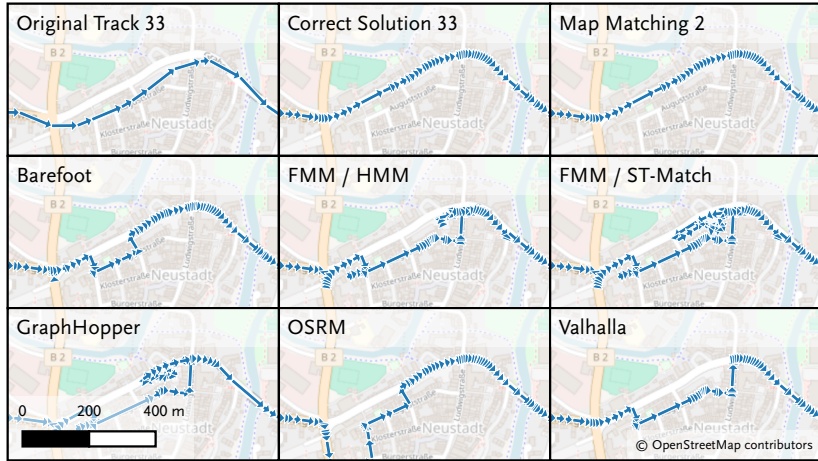


Figure 1.4: *Result comparison of different map matching tools* – We compare the original track and the hand-corrected solution to all open source software map matching tools that are evaluated in this work. Our novel approach Map Matching 2 (MM2) is the only solution that finds the correct result in this example. Abbreviations: Fast Map Matching (FMM), Hidden Markov model (HMM), Open Source Routing Machine (OSRM)

In [Figure 1.4](#) we can see how, based on a real-world scenario, a small shift to the side of the road in the measured original track leads to most solutions not being able to find good and correct results. In this example, our novel solution Map Matching 2 that we present in this work is the only one able to find the correct result.

1.9 Solution Approach

The main idea behind our novel map matching approach is to make the optimization process more robust against outliers and strong noise. Map matching is trivial when the tracks have only minor discrepancies with the road network, i.e., in the range of millimeters to centimeters, without additional noise or outliers. However, in practice, as seen in the left panel of the real-world example in [Figure 1.1](#), there is noise up to ≈ 50 m and there are outliers of more than ≈ 100 m. In such cases, finding an optimal match becomes a non-trivial optimization task.

To address such non-trivial situations, we present a novel map matching approach in this thesis. Our solution incorporates various advanced optimization technologies while maintaining high computational performance. To achieve this, we combine trajectory simplification algorithms with an improved candidate search and candidate refinement process, called candidate adoption, combined with a new metric that includes direction changes of the calculated routes. We ultimately integrate these components into a novel map matching model based on our MDP model design that is solved using reinforcement learning algorithms. For evaluating our results, we further introduce a new geometric line comparison algorithm that is optimized for map matching evaluation.

Starting from the next chapter, we first derive our concept from a thorough literature review. Then, we present our approach and describe its components. After that, we evaluate the effects of the individual and combined components on accuracy and computational performance. And finally, we perform a detailed benchmark experiment comparing our novel map matching approach to existing solutions using real-world and artificial data sets.

1.10 Scientific Contribution

With regard to the scientific contributions of this thesis: The focus of this thesis revolves around improving the performance of map matching in terms of both accuracy and computational speed. Based on our findings, we make a significant contribution to the research domain of map matching. The ongoing challenge of addressing the uncertainty of GNSS measurements in map matching requires substantial attention. By enhancing map matching performance, we can directly improve the outcomes of all use cases that depend on map matching in their processing pipeline.

In this thesis, we present numerous improvements to existing and new technologies relevant to the map matching processing pipeline. A common principle in current map matching algorithm pipelines is the order of operations for matching a given track to a route in a road network. First, a candidate search is performed to find probable mapping positions for the given GNSS measurements. Then, an optimization algorithm routes through these sets of candidates to find an optimal sequence according to the defined model and metrics.

Many existing map matching solutions rely on inappropriate candidate limits for the candidate search, which causes issues depending on the density of the road network. These limitations arise from the road density and the candidate search algorithm used, whether the limit is based on distance or the number of candidates. For example, an appropriate candidate limit for sparse, rural road networks may be too low for dense, urban road networks, where GNSS error is generally higher due to more and tighter buildings around the streets. Furthermore, very noisy GNSS measurements and outliers are not adequately addressed by regular candidate search limitations, as they regularly exceed them. They require a more sophisticated solution. For these reasons, we present a combined candidate search algorithm with a novel candidate adoption technology that dynamically adjusts the candidate set limits depending on the track and road network situation. This candidate adoption algorithm greatly increases the candidate limit in noisy situations, providing the optimization process with all the necessary information to address specific GNSS error situations, such as high noise

and outliers. Our candidate search methods and candidate adoption technologies can be adapted individually or in combination to existing or new map matching solutions.

Furthermore, most existing map matching solutions rely on HMMs that require probabilities during optimization. These probabilities are typically derived from the distances between GNSS measurements and the lengths of potential routes between candidates. However, such units are not necessarily well suited to represent probabilities, and the conversion into them can result in a loss of information, reducing the achievable accuracy. Therefore, we present a novel MDP that can use the metrics, such as distances, lengths, azimuths, and direction changes in routes directly, without converting them into probabilities. This makes our new MDP more robust against outliers in the tracks, which can result in vanishing probabilities in the HMM due to small probabilities. Our new MDP is a direct substitute for the HMM and can replace it in existing or new map matching solutions. Furthermore, our new metrics can be individually applied to existing or new map matching solutions, even without our specific model implementations.

In addition, we present multiple trajectory simplification algorithms for preprocessing GNSS tracks. These algorithms improve computational performance and accuracy by removing redundant data, low-noise data, and data with little additional information relative to existing data. These algorithms can also be applied individually or in combination to new or existing map matching solutions.

Our higher dynamic candidate limits require a high-performance computational algorithmic foundation. Therefore, we demonstrate various optimization technologies that can accelerate existing or new map matching solutions. The most important of these underlying algorithms is the routing performance, since the optimization process requires computing routes between all adjacent candidates. Routing performance is generally the slowest part of map matching algorithms, and many proposals with complex strategies for addressing this issue have already been researched. In this thesis, we present multiple strategies for improving routing perfor-

mance that can be applied individually or in combination to significantly improve routing performance in map matching.

Due to the large variety of new technologies presented in this thesis and the lack of sufficiently available real-world map matching ground truth data for evaluating our methods, we also introduce a new artificial map matching ground truth data generation process. Unlike previous GNSS error generation using Gaussian distributions, our process uses an autocorrelated and mean-reverting Ornstein-Uhlenbeck process with additional algorithms to generate outliers or missing GNSS measurements, in order to produce more realistic GNSS tracks with corresponding ground truth matches. This allows future map matching research to generate accurate artificial ground truth map matching data sets for evaluation. Furthermore, we present a new comparison algorithm that implements two existing accuracy and error metrics to thoroughly analyze and compare the performance of different map matching methods. This algorithm can be applied to any real-world or newly generated artificial data sets and map matching applications that produce geometric lines. Based on these technologies, we will demonstrate that existing map matching applications have different strengths and weaknesses depending on the metadata of the data set, whereas our novel algorithm performs consistent in all situations.

This thesis advances map matching to a whole new level. We will demonstrate that our new algorithms, models, and metrics, which are all integrated into our open source software reference implementation Map Matching 2, generally outperform existing solutions and are more robust when dealing with varying map matching data quality. Our novel method requires less configuration prior to matching arbitrary tracks because our algorithms and models are highly flexible by design. In this way, we provide a more intelligent solution that is less biased toward specific data sets. It requires less preconfiguration than existing solutions, as it provides greater freedom within its algorithms to produce meaningful and accurate map matching results. We will eventually show that, depending on the metadata of the data set, most existing solutions have strengths and weaknesses, whereas our novel map matching algorithm performs strong in all evaluated data sets, even when other algorithms fail significantly.

2 Related Work

In this chapter, we establish the foundation for our novel map matching approach through a comprehensive literature review. We will cover topics related to input and output data, algorithms, models, metrics, and existing state-of-the-art open source software implementations.

2.1 Global Navigation Satellite Systems

The input data for map matching includes a digital road network and geospatial global navigation satellite system (GNSS) measurements, i.e., the tracks. Therefore, an understanding of GNSS measurement errors is essential for map matching.

2.1.1 GNSS Accuracy

The topic of map matching originates from the inaccuracy of GNSS positioning. Currently, there exist various GNSSs, e.g., the United States of Americas Global Positioning System (GPS)³, the European Unions Galileo⁴, Russia's GLONASS⁵, and China's Beidou⁶.

³<https://www.gps.gov/>

⁴<https://www.euspa.europa.eu/>

⁵<https://glonass-iac.ru/en/>

⁶<http://en.beidou.gov.cn/>

The inaccuracy of GNSS means that measuring a position with satellite-based positioning systems is subject to significant measurement error. There are many reasons for this, e.g., unreachable GNSS satellites (Grabs et al. 2008), atmospheric disturbances or space weather phenomena (Kos et al. 2013), or multipath inference, for example, due to signal reflections on large buildings (Marais et al. 2005). Specifically, using smartphones for GNSS positioning is prone to higher error (Hu et al. 2023), and the accuracy varies between different devices (CartONG 2019).

Since 2017, there have been smartphones on the market with GNSS receivers for multiple providers (Zangenehnejad and Gao 2021). As such, the differences between the various GNSS providers are blurring, and measured positions often stem from a combination of multiple GNSS providers today. However, at the same time, smartphones are particularly sensitive to measurement errors due to their antenna design and battery-saving features (Fabricio and Nascimento 2025; Zangenehnejad and Gao 2021). Consequently, positioning quality is often not as good as expected, despite the combination of multiple GNSS providers.

As the concrete GNSS providers regularly blend in practice, we mean arbitrary providers or combinations of them when talking about GNSS in this work.

2.1.2 GNSS Error Dimension

For map matching, the horizontal GNSS error, i.e., the circular error on the surface around the true position, is very relevant because it provides insight into how distant the measured points may lie next to the road network. In contrast, the vertical GNSS error, i.e., the altitude error on the surface, is barely relevant in practice because public road networks, such as OpenStreetMap, generally lack height information about roads, making an elevation metric challenging.

For the horizontal error, several analyses have been conducted. For example, in Diggelen and Enge (2015) the error range with smartphones in an open

space was measured to be within ≈ 4.9 m. In Izet-Ünsalan and Ünsalan (2020) the ≈ 5 m range of smartphone GNSS accuracy in an open space is confirmed, and it is shown that the error also depends on the GNSS site, i.e., the spatial location on the globe. Furthermore, it is shown that the error is $\approx 3 - 4$ times higher near buildings, up to ≈ 15 m – 20 m, which is especially relevant for floating car data, i.e., driving in cities while navigating with smartphones. When satellite opacity (Grabs et al. 2008) occurs, i.e., when a GNSS receiver loses contact with enough satellites for locating its position, e.g., because of buildings, hills, bridges, or tunnels, larger outliers can happen. These outliers are outside the regular noise distribution and can reach several hundred meters in practice. For example, the ground truth map matching data set from Newson and Krumm (2009) contains an outlier track point with a > 100 m distance to the ground truth. Additionally, the positioning can also fail completely, resulting in missing data points in the measured sequence, which also happens around tunnels in the data set from Newson and Krumm (2009).

One important property of sequential GNSS measurements for map matching metrics is that they overshoot the actually traveled distances (Ranacher et al. 2015). This means that the distance between two consecutive, temporally close GNSS measurements is often longer than the actually traveled distance. For example, when moving in a straight line, the GNSS measurements oscillate around the actual movement. In other words, GNSS measurements suggest movement even when no movement occurs, such as in a static position. The intrinsic inaccuracy of GNSS measurements is the main reason why there is no trivial solution to the map matching problem.

2.1.3 GNSS Autocorrelation

GNSS error is spatially and temporally autocorrelated, as extensively researched by analyzing stationary GNSS records (Giofre et al. 2017; Grabs et al. 2008; Ranacher et al. 2015; Rostami et al. 2019; Soundy et al. 2020). In Giofre et al. (2017) the authors show that the GNSS distance error distribution mostly follows a Rayleigh distribution. A Rayleigh distribution

is a probability distribution for non-negative Gaussian-distributed values. The GNSS longitude and latitude measurements mostly follow a Gaussian distribution, as shown in Grabs et al. (2008). This makes the GNSS distance error, which is the ‘geometric sum of latitude and longitude errors’ (Grabs et al. 2008), similar to a Rayleigh distribution (Grabs et al. 2008). The authors in Giofre et al. (2017) also state that using an error distribution for simulating GNSS error that does not account for autocorrelation leads to a different kind of error because ‘the data would fluctuate randomly without the autocorrelation phenomena’ (Giofre et al. 2017). This would occur, for example, when using a Gaussian distribution for simulating GNSS error on successive points.

Autocorrelation means that subsequent measurements taken close to each other in time generally have similar shifts on the axes within the accuracy margins. Although GNSS measurements depend on previous values, this dependency decreases over time. This is because, within close timestamps, the satellite positions, atmospheric disturbances, and earth rotation are similar. However, they diverge over time. In Grabs et al. (2008) the authors present a GNSS error autocorrelation function, GNSS error over time, and a moving average smoothing of the error over time, to illustrate the autocorrelation of GNSS measurements. The authors also demonstrate how satellite opacity, which produces outliers, significantly increases the distance error. Such outliers are far outside the original error distribution and autocorrelation. In Ranacher et al. (2015) the spatial and temporal autocorrelation is also confirmed. Furthermore, it is confirmed that GNSS distance errors tend to overestimate actual distances, i.e., a GNSS track recorded at a static position will always be longer than the distance the non-moving recording device has actually moved.

The spatiotemporal autocorrelation of GNSS is also relevant for map matching when it comes to designing metrics. The more autocorrelated the measured points are, the more reliable their geometric relationship. However, measuring points while driving at high speeds negatively impacts autocorrelation, i.e., the greater the distance between consecutive GNSS measurements, the lower the autocorrelation (Ranacher et al. 2016).

2.1.4 GNSS Simulation

Simulating GNSS error allows for the creation of artificial tracks for map matching evaluation purposes. There are different approaches for this. For example, in Rostami et al. (2019) the authors simulate GNSS error with an autoregressive model, and they validate their synthetic GNSS data with real-world collected data to demonstrate the capabilities of the artificial model. In Soundy et al. (2020) the Ornstein-Uhlenbeck process with the Euler-Maruyama method is applied for simulating GNSS noise. The same work also evaluates the simple Gaussian distribution, the autoregressive model, the moving-average model, and the autoregressive moving-average (ARMA) model. The authors explain that, while the autoregressive, the moving-average, and the ARMA models are all autocorrelated, only the Ornstein-Uhlenbeck process is also mean-reverting. This means that, over time, the generated noise depends not only on previous values, but also returns to a defined value, for example, zero, with greater attraction the farther the current error is from the defined value. The authors in Soundy et al. (2020) also demonstrate that noise generated with the Ornstein-Uhlenbeck process is similarly autocorrelated as real GNSS noise, in their case most closely comparable up to two hours. Because the Ornstein-Uhlenbeck process is based on a Gaussian process for generating random values, it produces GNSS error distributions similar to real GNSS error over time.

2.2 Map Matching Models

To address the inherent inaccuracy of GNSS measurements and to minimize the discrepancies between measured points and road networks, various map matching models and algorithms have been developed over the last few decades.

2.2.1 Historic Methods

Map matching has a long and comprehensive research history. Early geometric approaches used simple point-to-curve and curve-to-curve mapping (Bernstein and Kornhauser 1996; White et al. 2000). In Brakatsoulas et al. (2005) more advanced approaches employing geometric curve-to-curve mapping with Discrete Fréchet Distance (Eiter and Mannila 1994) and Free Space Diagrams (Alt et al. 2003) have been described. A major issue with these geometric approaches is that they are easily affected by measurement noise and outliers. They are not designed to evaluate multiple alternative paths against each other. In Quddus et al. (2007) the authors provide a comprehensive overview of these and other historical map matching approaches. At that time, the authors demanded new technologies to address the persistent inaccuracies in map matching.

2.2.2 Stochastic Methods

Recent map matching algorithms have shifted towards stochastic methods. Unlike previous methods, stochastic methods allow for evaluating multiple alternative paths and selecting an optimal solution. An optimal solution means that there are no alternative paths with a higher total score according to the stochastic model. However, there can be multiple alternative paths with the same score, and any of these paths is an optimal solution. Therefore, an optimal solution in a stochastic model is not necessarily a perfect match to reality, but rather depends on the defined model dynamics and metrics. In other words, the closer the model and metrics represent reality, the more accurate the match.

The authors in Hummel (2006) first described the map matching problem using a hidden Markov model (HMM). In their work, they used the HMM to describe road network connectivity. They then used the Viterbi dynamic programming algorithm (Forney 1973) to find an optimal path through their model. In Krumm et al. (2007) the HMM idea was then remodeled to use travel time constraints, whereby the travel time of a computed path through

the HMM was compared with the recorded travel time. This concept was further evolved in Newson and Krumm (2009), which uses only geometric distance information for the HMM metrics. In this work, the basic idea was that each GNSS measurement can be represented by an observation in the HMM, and for each observation, several hidden states, which are the potential candidates, can exist in the road network. These states are called ‘hidden states’ because the actual states, i.e., the actual positions, remain unknown. Each observation then receives a probability for the corresponding hidden states of the respective time step, calculated from the point distance between the GNSS points and the candidates. And each time-adjacent pair of hidden states receives a transition probability, calculated from the routing distance between the corresponding candidates and the length of the corresponding track segment. Again, the Viterbi algorithm computes an optimal sequence of hidden states based on the observations. The work in Newson and Krumm (2009) was a breakthrough in the domain of map matching at its time because it could handle a certain degree of GNSS noise by evaluating multiple alternative paths before selecting the final match.

The stochastic methods can also be described as ‘global max-weight’ optimization (Wei et al. 2012). Consequently, tuning the metrics of the optimization algorithms has a significant impact on the quality of the matches. Various alternative formulas and terms were tested for the metrics, such as distances, times, speeds, azimuths, and combinations of these. Wei et al. (2012) also contains an overview of the metrics that had been tried up to that moment. Later approaches also tried combining stochastic and geometric methods, i.e., hybrid methods, such as Discrete Fréchet Distance combined with HMMs (Wei et al. 2013) or Vertex-Monotone Fréchet Distance combined with HMMs (Chen et al. 2021).

2.2.3 Conditional Random Fields

Conditional random fields (CRFs) (Lafferty et al. 2001) were also explored in map matching (Li et al. 2024; Liao et al. 2007; Liu and Lu 2015; Xu et al. 2015). The basic idea of CRFs was to overcome the ‘label bias’ that Maximum-

Entropy Markov Models (MEMMs) entail when the weights are normalized locally per state. Both models, also known as ‘discriminative’ models, work with *conditional* probabilities. In CRFs, the weights are normalized globally over all states, which prevents getting stuck in a local optimum. HMMs are different models, also known as ‘generative’ models, that work with *joint* probabilities. They do *not* suffer from ‘label bias’, as stated in the original work (Lafferty et al. 2001), confirmed in an additional work (Dietterich 2002), and thoroughly explained in Hannun (2019). However, this fact is often cited incorrectly in the literature, and CRFs are presented as an alleged solution over HMMs to solve ‘label bias’ (Li et al. 2024; Liu and Lu 2015; Xu et al. 2015). This misunderstanding has been present for over a decade in map matching, stating an issue of HMMs that they do not have. In addition, HMMs are less computationally expensive than CRFs because they do not require global normalization, and the states have a strict independence constraint (the Markov property), i.e., a state only depends on the immediately preceding state. This strong independence assumption allows for the efficient solving of HMMs.

2.2.4 Stochastic Model Limitations

As we have seen, stochastic map matching methods are generally suited for evaluating multiple alternative solutions, as opposed to simple geometric approaches. However, they can still become inaccurate when the input data contains outliers or significant noise (Newson and Krumm 2009). For instance, noise emitted from a static position, such as standing at a traffic light while the GNSS measurement continues, can cause a stochastic model to move back and forth within a junction, similar to what is shown in Figure 1.2.

For this reason, some approaches use a U-turn penalty (Chao et al. 2020; Yang and Gidófalvi 2018) on the metrics to correct this behavior. However, this approach may not work reliably with outliers because they impact the metrics of the model significantly. For example, candidates may be present at the outlier position, which the model then has to incorporate in

its alternatives. Hence, outliers can pollute the stochastic model metrics and interfere with optimization.

Additionally, stochastic methods can become significantly slower as a track or road network becomes denser due to repeated routing invocations and increased probability computations caused by a higher candidate count. This issue is often mitigated by using only a few candidates around each track point, for example, only 8 candidates (Yang and Gidófalvi 2018). However, this also reduces the number of possibilities that the stochastic model can consider for finding suitable matches.

2.2.5 Online and Offline Map Matching

Online map matching is a technology that is performed while a track is being recorded, e.g., during navigation within a road network (Goh et al. 2012). In contrast, offline map matching is a technology used to match previously recorded tracks (Pereira et al. 2009). Offline map matching has access to the entire recorded track before the matching process begins. However, it generally lacks access to additional real-time sensors, such as dead reckoning sensors that provide acceleration, heading, and velocity data, which are available in online use cases (Ochieng et al. 2009). Therefore, online map matching is a different approach from offline map matching because it may have access to additional real-time data and needs to fulfill real-time computation requirements that offline map matching does not.

However, a common approach for making offline map matching methods work as online methods is to use sliding windows (Newson and Krumm 2009), e.g., fixed windows (Liang et al. 2016) or variable sliding windows (Goh et al. 2012) for the number of preceding points, or fixed interval window sizes as the time delta between subsequent points (Yin et al. 2018). These approaches enable offline map matching algorithms to be used for online map matching without requiring additional sensor information.

2.2.6 Recent Literature Reviews

In recent years, several comprehensive literature reviews and surveys have been conducted on the current state of map matching (Chao et al. 2020; Dima and Popa 2023; Gupta and Shanker 2022; Huang et al. 2021; Kubicka et al. 2018). To gain a deeper insight into current opinions and facts on map matching, we recommend consulting these works. While each research comes to different conclusions about future directions in map matching, there is also a notable consensus that GNSS data quality is one of the biggest challenges to date. Therefore, we conclude that map matching needs to more effectively address this challenge for being practically useful.

2.2.7 Further Research Directions

One notable further research direction in map matching is the use of approaches based on artificial intelligence, such as deep learning models (Jiang et al. 2023; Liu et al. 2021; Shen et al. 2020; Zhao et al. 2019) or transfer learning approaches (Ayfantopoulou et al. 2022; Jin et al. 2022). These approaches may outperform traditional stochastic methods in terms of accuracy and runtime performance. However, they have some limitations in practice. First, they require training on large volumes of labeled data, which is rare, as we will see in subsection 2.4.1. Second, they are often location-dependent because they are trained on the spatial locations of the road network or tracks (Shen et al. 2020), or trajectory patterns (Jiang et al. 2023), which depend on specific road shapes. Finally, the internal mechanics of why a match is performed in a specific way are usually difficult to explain and comprehend in approaches based on artificial intelligence, which can limit their reliability and trustworthiness in practice. Nevertheless, transferring learned driving behavior from specific locations to degraded tracks with large noise or outliers can improve accuracy in situations where track-independent stochastic models cannot achieve this.

2.3 Map Matching Base Algorithms

In order to find matches according to the defined map matching models, various additional base algorithms and concepts are necessary. Using these appropriately directly impacts the quality of the matches.

2.3.1 Map Matching Candidates

A primary subprocess of map matching is searching for candidates. The term ‘candidate’ has already been used in earlier literature (Bernstein and Kornhauser 1996; White et al. 2000). Stochastic methods perform a candidate search for each GNSS point (Krumm et al. 2007; Lou et al. 2009; Newson and Krumm 2009) for finding possible road positions, i.e., candidates. Providing multiple alternative candidates per GNSS point allows stochastic methods to compare alternative mappings for a given track. Thus, the quality of the candidate search has a direct impact on the quality of map matching. Candidates are organized into candidate sets for each GNSS point, and the stochastic model then evaluates all possible connections, i.e., routes, between adjacent candidate pairs, i.e., a cross-product of all candidates from adjacent candidate set pairs.

In stochastic methods, the candidate search is either carried out with a circular (Newson and Krumm 2009), or a k -nearest neighbor search combined with circular filtering (Yang and Gidófalvi 2018). Various search radii have been proposed for the circular search in the literature, e.g., 15 m (Fiedler et al. 2022) for large-scale map matching, 30 m or 50 m (Goh et al. 2012; Wei et al. 2012) for high computational speed, or 200 m (Krumm et al. 2007; Newson and Krumm 2009) for high matching accuracy. For k in the k -nearest neighbor search, a value of 8 has been used in combination with a 300 m distance filter (Yang and Gidófalvi 2018). These different approaches naturally lead to the idea of combining efficiency and accuracy by using dynamic search radii, e.g., depending on the sampling interval of the GNSS measurements (Che et al. 2018).

2.3.2 Shortest Path Routing

Shortest path routing is an essential part of map matching used for computing routes in the road network. These are usually computed using either the single-source-multiple-target (i.e., one-to-many) Dijkstra’s shortest path algorithm (Dijkstra 1959) or the single-source-single-target (i.e., one-to-one) A-star shortest path algorithm (Hart et al. 1968). Some map matching algorithms use Dijkstra’s algorithm, while others use the A-star algorithm (Lou et al. 2009; Saki and Hagen 2022). Some argue that the A-star algorithm is more efficient than Dijkstra’s algorithm (Yang 2020). However, the A-star algorithm may not fulfill the consistency and admissibility requirements (Fiedler et al. 2022) in map matching that would be needed for finding optimal routes. In Hu and Lu (2019) the authors leave the decision of Dijkstra versus A-star open.

Map matching is generally a multiple-source-multiple-target (i.e., many-to-many) problem, due to computing all routes of the cross-product of adjacent pairs of candidates sets. Different techniques exist for speeding up the many-to-many route search, e.g., contraction hierarchies, bidirectional search, and landmarks (Knopp 2006). However, these techniques are designed for finding long routes in large graphs. In contrast, routing in map matching is performed in a ‘clustered space’ (Jagadeesh and Srikanthan 2019), i.e., a closely bounded geospatial area, stemming from the spatial extent of the cross-products of the adjacent candidate sets. In addition to presenting a custom algorithm to solve this problem, the authors explain in their introduction that, in its simplest form, the problem can be solved by running Dijkstra’s algorithm multiple times for each source node. They state that this is generally faster than one-to-one solutions in such a ‘clustered space’. Additionally, the authors measured that, in their use case, routing accounted for around 90% of the map matching runtime with noisy tracks before their customizations. Therefore, improving the routing runtime performance is essential for optimizing the overall map matching runtime performance.

An additional solution for addressing the clustered characteristics of shortest path routing in map matching is to use an upper bound to limit the

search radius (Yang and Gidófalvi 2018). The authors of this work compute all shortest paths within a certain distance limit for a road network in advance and store them in an upper bounded origin-destination table (UBODT). However, this approach may not be practical for large road networks, particularly when only a few tracks are matched, as it results in many unnecessary calculations for creating the UBODT.

Another interesting way to improve the speed of shortest path routing is to remove unnecessary nodes and edges from the road network graph without altering its geometric representation, for example, using a road network topology simplification algorithm. The authors of OSMNx⁷ (Boeing 2017) provide such an algorithm and explain that it is essential for OpenStreetMap networks due to their node and edge representation. We will explore this topic in more detail later in subsection 4.2.1 with an illustration of the output of such an algorithm in Figure 4.8.

2.3.3 Missing Data Treatment

Under the assumption that a road network is available, two cases of missing data can occur in map matching. The first is missing GNSS measurements, and the second is missing candidates, for example, when a GNSS measurement is too far away from the roads.

The former leads to a more sparse track, meaning the resolution or sampling interval is lower. This impacts both the accuracy and computational speed of map matching. For example, fewer GNSS measurements lead to fewer candidates to be examined, thus saving computational performance. However, the routes between the candidates become longer, which may require more time to compute them. Furthermore, fewer points reduce total noise, but also reduce the amount of information about the actual movement. Thus, missing GNSS measurements can benefit the matching process, especially when the missing points are outliers. However, they can also decrease the accuracy of the results when the information density becomes too low.

⁷<https://github.com/gboeing/osmnx>

Nevertheless, the latter issue of missing candidates is more severe. If there are no candidates for a given GNSS measurement, for example, due to missing roads in the road network or because the measurement is an outlier too far away from the roads, the HMM has no hidden states available for the observation. Ultimately, this means that the HMM cannot be solved because there is no path through the model connecting all observations through hidden states. In this case, several workarounds exist in map matching, for example, removing the observation with missing hidden states and gluing together the remaining parts, or splitting the HMM into multiple ones and solving them individually, or a combination of both (Newson and Krumm 2009).

2.3.4 Trajectory Simplification

One way to reduce the noise of GNSS tracks is to apply trajectory simplification algorithms to them before they are matched. Trajectory simplification aims to remove ‘unnecessary’ points from a track to reduce small and simple noise, which makes the track more sparse. This can increase the accuracy of the matches because the stochastic methods have to deal with less noise, and it improves the matching speed because fewer candidates have to be examined.

One trajectory simplification approach is the Douglas-Peucker algorithm (Douglas and Peucker 1973; Ramer 1972) to remove points without significantly altering the geometric shape of a track, provided the algorithm is applied correctly (Ebisch 2002). However, as a purely geometric approach, the Douglas-Peucker algorithm is unaware of the time information that GNSS measurements contain. Therefore, there are trajectory simplification algorithms, sometimes also known as trajectory compression algorithms, that try to preserve timing information, e.g., the TD-TR algorithm (Meratnia and By 2004), which is adopted from the Douglas-Peucker algorithm, and the more recent CASC algorithm (Zhong et al. 2022), which offers further enhancements over the TD-TR algorithm.

In Li et al. (2014, 2015) several advanced geometric trajectory simplification algorithms are evaluated with map matching. The authors conclude that runtime and matching accuracy increase when a trajectory simplification algorithm is applied, as long as the simplification is only applied in a narrow frame to high-sampled GNSS sequences. They also note that applying too broad simplifications to sparse GNSS sequences may, in turn, increase the matching error.

Another notable work is that of Zhang and Masoud (2020), in which an actor-critic model from the reinforcement learning domain is trained by applying artificial noise to an existing GNSS track. The trained model then acts as a trajectory simplification algorithm to predict denoised points from the noisy input.

2.3.5 Reinforcement Learning

Reinforcement learning is a third domain of machine learning, separate from the more widely known supervised and unsupervised learning (Sutton and Barto 2018). In its intended sense, reinforcement learning does not require labeled data for training, but rather learns from the experience it samples on its own. The central concept is that an agent takes actions in an environment, transitions from one state to another, and receives a reward for each transition. Eventually, the agent learns to distinguish between ‘bad’ and ‘good’ decisions based on the current state by optimizing the action selection strategy to maximize the total reward. The environment is generally modeled with a Markov decision process (MDP), and there are various algorithms for the agent, for example, Monte Carlo methods, dynamic programming methods, such as Value Iteration and Policy Iteration, Temporal Difference Learning (TD-Learning) methods, such as Q-Learning (Watkins 1989; Watkins and Dayan 1992) and SARSA, and more sophisticated methods, such as n-step bootstrapping, Lambda-methods, and approximate methods for continuous state and action spaces. The reinforcement learning reference book (Sutton and Barto 2018) contains detailed information on this topic, as well as visualizations and pseudocodes for the aforementioned algorithms.

In the map matching domain, different reinforcement learning ideas have already been researched. In Ziebart et al. (2008), inverse reinforcement learning (IRL) with MDPs was used to learn the reward function based on expert demonstration, i.e., labeled data. In Osogami and Raymond (2013), the authors use inverse reinforcement learning to learn from observation, and they combine their results with HMMs. In Sun and Xiong (2016), the authors use an existing map matching solution to train their History Markov decision process (HMDP) on. This process is then used to infer partial solutions from high-sampled matches to low-sampled inputs. In Wu et al. (2016), the authors apply inverse reinforcement learning to a MDP. It should be noted that all of these approaches rely on labeled data for training and do not employ reinforcement learning in its original form of learning from its own sampled experience, as our novel approach does.

2.4 Map Matching Evaluation

In order to measure the accuracy of map matching, it is necessary to have access to data sets and accuracy metrics so that different implementations can be evaluated.

2.4.1 Map Matching Data Sets

For evaluating map matching results, labeled data is necessary. Ideally, this would be ground truth data. Two of the most well-known public ground truth map matching data sets are the Seattle data set from Newson and Krumm (2009), available at *Microsoft Research*⁸, which contains one track with ground truth in Seattle, Washington, USA, and the Melbourne data set from Li et al. (2014), available at the *Internet Archive*⁹ (originally available

⁸<https://www.microsoft.com/en-us/research/publication/hidden-markov-map-matching-noise-sparseness/>

⁹<https://web.archive.org/web/20170301001019/https://people.eng.unimelb.edu.au/henli/projects/map-matching/>

at the *University of Melbourne*), which contains one track with ground truth in Melbourne, Australia.

Then, there is the map matching GIS CUP data set from Mohamed et al. (2012), available at the *Internet Archive*¹⁰ (originally available at the *University of Washington*). It contains 10 tracks with their solutions, all of which are also located in Seattle, Washington, USA. This data set was created for a map matching competition, and was presumably hand-corrected, since it contains multiple errors in the provided solutions, especially in tracks 2 and 7, as discussed later in subsection 5.4.3.

One of the largest available map matching data sets is the Worldwide data set (Kubicka et al. 2016), referenced in Kubicka et al. (2015), containing 100 worldwide tracks. It is available under creative commons license CC BY-SA 4.0¹¹ at *Zenodo*¹². It is also not a ground truth data set because the authors manually matched and hand-corrected the tracks that they took from the OpenStreetMap GPX (GPS Exchange Format) ‘Planet.gpx’¹³ project (Kubicka et al. 2015). In other words, the ground truth was unknown to the authors. However, due to its worldwide spatial distribution, the data set incorporates various map matching situations with different challenges.

All data sets provide the road network in a custom text format, along with the tracks and the solutions as lists of edge IDs, which depend on the provided road network. From the lists of edge IDs, the geometric lines of the results can be extracted from the edges of the road networks. Due to the nature of all provided tracks being recorded by GPS, all data sets are given in the GPS spatial reference system WGS-84.

¹⁰<https://web.archive.org/web/20130127211936/http://depts.washington.edu/giscup/home>

¹¹<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

¹²<https://zenodo.org/record/57731>

¹³<https://wiki.openstreetmap.org/wiki/Planet.gpx>

2.4.2 Map Matching Accuracy Metrics

For evaluating the accuracy of map matching results, various proposals have been made in the literature. In any case, the accuracy is computed by comparing the matches from an algorithm with the provided solutions.

One way to compute the accuracy metric is by using the error fraction formula from Newson and Krumm (2009), see Equation 2.1. In this formula, the lengths of the segments added (e_a) and missed (e_m) from the correct solution are divided by the length of the correct solution (l_g).

Equation 2.1 (Error Fraction):

$$e_f = \frac{e_a + e_m}{l_g} \quad (2.1)$$

Unfortunately, the range of the results is open-ended here, meaning there can be error fractions > 1.0 . This makes it difficult to comprehend how accurate one result is compared to another. For example, when the provided solution is very short, i.e., only a few meters, and the match loops around the block for several hundred meters, contributing to the erroneously added parts, the error fraction becomes very large > 1.0 . Conversely, if the provided solution goes around the block and the match is only a few meters, this contributes to the erroneously missed parts, and the error fraction is < 1.0 . Furthermore, for example, an error fraction of 3.0 does not indicate which result has matched more segments correctly than another result with an error fraction of 2.0, only that the former contains more errors compared to the solution.

Therefore, there also exists the accuracy metric after Equation 2.2, which is already used in Ayfantopoulou et al. (2022), Wu et al. (2016), and Zheng et al. (2012). It calculates the percentage of correctly matched segments, i.e., the accuracy in percent, by dividing the length of the correctly matched segments (l_c) by the greater of the length of the correct solution (l_g) or the match (l_m).

Equation 2.2 (Correct Fraction):

$$c_f = \frac{l_c}{\max(l_g, l_m)} \quad (2.2)$$

In this formula, the metric range lies between 0.0 to 1.0, whereas 0.0 means there is not a single correct segment, and 1.0 means there is a perfect match to the provided solution. A value of 0.5 means that half of the match is correct, and half is not, i.e., it is either erroneously missing or erroneously added over the provided solution, or contains traces of both. For example, if one result has an accuracy of 0.6 and another has an accuracy of 0.4, it is clear that the first match has more correctly matched segments and less error than the second match. Recalling the error fraction example above, if the provided solution is a few meters and the match is a few hundred meters around the block, the correct fraction is a small percentage, i.e., the part of the match that is correct. If the data is reversed and the provided solution is a few hundred meters and the match is only a few meters, the correct fraction is the same. In both cases, only a few meters were correctly matched, which is what the correct fraction represents. For this reason, we use the correct fraction rather than the error fraction in this work to represent accuracy.

Historically, various other algorithms have been used to compare two lines, e.g., Fréchet Distance or Hausdorff Distance can be used to compute a distance value, or spatial comparisons (e.g., equals, touches, within) return a Boolean result for the respective condition. However, none of these algorithms can calculate the percentage of overlap between two lines and are therefore unsuitable for determining the accuracy of a match to a provided solution. Regarding a direct comparison of two routes, only a few approaches exist (Huang and Lin 2019). However, these approaches are also not applicable in the map matching case due to design limitations. As an example, it is not possible to compare two results from different map matching implementations when their match outputs contain small floating point discrepancies in the coordinates. Such floating point inaccuracies naturally emerge when different computations in different implementations are carried out on modern computers.

Another issue with comparing a match to a provided solution is that, when the match starts at a different position along an edge than the solution, the point at which the segments overlap is required but missing in the longer segment. The same thing happens when the two lines start to overlap again after diverging, and the point at which they re-merge is also missing from the other line. We will discuss our solution to this issue in [subsection 3.1.4](#), with an illustration in [Figure 3.9](#).

For these reasons, previous map matching evaluations were usually made by comparing the IDs of the edges from the road network with each other, i.e., the match was returned as a list of edge IDs, and the solution was also provided as a list of edge IDs. However, comparing only complete edges by ID cannot capture forward-backward, i.e., zigzag, movements along an edge without leaving it or the previously mentioned issue of the match starting later along the edge. In practice, tracks rarely start or end exactly at the beginning or ending of an edge, i.e., exactly within an intersection or T-junction, but rather from somewhere along a road, e.g., a parking spot at the side of the road or a private garage. As such, there is generally a discrepancy between a match and the provided solution when it is given as a list of edge IDs.

Most articles do not describe how they compare matches to the provided solutions. As such, we must assume that, in most cases, matches are stretched to complete edges and zigzag movements within an edge are ignored. This is because, when comparing edge IDs, subsequent points mapped to the same edge generate adjacent duplicate edge IDs in the resulting list. These duplicate edge IDs need to be removed before comparing them to the provided solution, which generally does not contain duplicate edge IDs in the route. However, this process may eliminate information, such as removing duplicate edge IDs from zigzag movements, as they cannot be distinguished anymore from duplicate edge IDs from subsequent mappings. Therefore, the accuracy of previous studies that rely on comparing lists of edge IDs should be taken carefully. Furthermore, it also means that when we compare matches with the provided solutions after converting them to geometric lines, discrepancies between the solutions spanning complete edges and the matches that map the tracks starting somewhere along the edge are inevitable.

2.4.3 State-of-the-Art Implementations

The major state-of-the-art open source software map matching implementations reviewed in this work are Barefoot¹⁴ (Louail et al. 2014), Fast Map Matching (FMM)¹⁵ (Yang and Gidófalvi 2018), GraphHopper¹⁶, Open Source Routing Machine (OSRM)¹⁷ (Luxen and Vetter 2011), and Valhalla¹⁸. All of these existing approaches implement HMMs based on the work of Newson and Krumm (2009), except for Fast Map Matching, which also contains a simpler stochastic model called ST-Matching from Lou et al. (2009) for matching larger tracks more quickly. We compare all these implementations with our own novel map matching solution Map Matching 2¹⁹ (Wöltche 2023), which primarily uses MDPs but also includes HMMs for comparison purposes. In Table 2.1 we provide an overview of the programming languages, interfaces, and licenses of the aforementioned tools.

Table 2.1: Overview of existing state-of-the-art open source software map matching tools in comparison to our novel implementation Map Matching 2 (MM2). Abbreviations: Fast Map Matching (FMM), Open Source Routing Machine (OSRM), Command Line Interface (CLI), Hypertext Transfer Protocol (HTTP), Berkeley Software Distribution (BSD), Massachusetts Institute of Technology (MIT), Affero General Public License (AGPL)

Software	Languages	Interfaces	License
Barefoot ¹⁴	Java	CLI, HTTP	Apache 2.0
FMM ¹⁵	C++	CLI, Python, HTTP	Apache 2.0
GraphHopper ¹⁶	Java	CLI, HTTP	Apache 2.0
OSRM ¹⁷	C++	HTTP	BSD-2
Valhalla ¹⁸	C++	CLI, HTTP	MIT
MM2 ¹⁹	C++	CLI	AGPL v3.0

¹⁴<https://github.com/bmwcarit/barefoot>

¹⁵<https://github.com/cyang-kth/fmm>

¹⁶<https://github.com/graphhopper/graphhopper#map-matching>

¹⁷<https://github.com/Project-OSRM/osrm-backend>

¹⁸<https://github.com/valhalla/valhalla>

¹⁹<https://github.com/addy90/map-matching-2>

In one of our previous works (Wöltche 2023), we also reviewed pgMap-Match²⁰ (Millard-Ball et al. 2019). However, in comparison with the other evaluated tools, the results did not indicate the need for another review, so we excluded this implementation in this work.

The impact of the work from Newson and Krumm (2009) becomes apparent when reviewing the models and algorithms used in the aforementioned tools. All implementations are based on the proposed HMM design and use the Viterbi algorithm, as can be seen in the overview in Table 2.2. Nevertheless, further enhancements were sometimes made, such as changes in the metrics or additional models.

Table 2.2: *Models and algorithms of existing state-of-the-art open source software map matching tools* in comparison to our novel implementation Map Matching 2 (MM2). Abbreviations: Fast Map Matching (FMM), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP)

Software	Models	Algorithms
Barefoot ¹⁴	HMM	Viterbi
FMM ¹⁵	HMM, STM	Viterbi, STM
GraphHopper ¹⁶	HMM	Viterbi
OSRM ¹⁷	HMM	Viterbi
Valhalla ¹⁸	HMM	Viterbi
MM2 ¹⁹	HMM, MDP	Viterbi, Policy Iteration, Value Iteration, Q-Learning

The aforementioned tools can be divided into two groups. The first group consists of tools specifically designed for map matching, including Barefoot, Fast Map Matching and our own solution Map Matching 2. The second group consists of general routing engines with a map matching algorithm on top, including GraphHopper, Open Source Routing Machine, and Valhalla. The main difference is that the first group received specific optimizations tailored for map matching, while the second group required less costly integration of map matching but has a foundation that is not

²⁰<https://github.com/amillb/pgMapMatch>

specifically designed for map matching tasks. This difference should be kept in mind when evaluating and comparing the performance of these tools.

2.4.4 Traffic Simulation

Concerning traffic simulation, various implementations have been shown to accurately model traffic (Baza-Solares et al. 2022; Ma et al. 2021; Schweizer et al. 2021). The application Simulation of Urban MObility (SUMO) (Krajewicz et al. 2002; Lopez et al. 2018) is an open source micro traffic simulation software that supports OpenStreetMap road networks and floating car data (FCD) output for generating routes that are virtually driven while simultaneously emitting GNSS measurements. Traffic can be generated in various ways, such as randomly, using origin-destination (OD) matrices, based on road counting data, turn probabilities, or based on activities, for example, based on population distribution using the ‘SUMO Activity GenerAtion’ (SAGA) tool-chain (Codeca et al. 2022).

2.4.5 Floating Car Data Simulation

There are a few previous works that generate artificial map matching ground truth data based on traffic simulations. However, the processes are not openly available. In Erdmann and Ebendt (2014) traffic is simulated using SUMO, and the onboard ‘traceExporter.py’ tool is used to apply artificial noise based on a simple Gaussian normal distribution. Consequently, the artificial error is not autocorrelated, unlike real GNSS error. The authors use a sampling interval of 30 s with a Gaussian distribution standard deviation σ of 5 m. Using a low sampling interval mitigates the effects of autocorrelation because samples that are farther apart are less autocorrelated, especially with moving vehicles in cities. In Lindholm (2020) traffic is also simulated using SUMO, and the open source tool GNSS-INS-SIM²¹ is used to generate artificial noise, again based on a simple Gaussian normal

²¹<https://github.com/Aceinna/gnss-ins-sim>

distribution. They use sampling intervals of 1 s, 5 s, and 10 s with respective Gaussian distribution standard deviations σ of 5 m, 10 m, and 20 m. Since the applied noise distributions in these works are not autocorrelated, only low noise can be simulated with high sampling intervals. Otherwise, the track shape degrades significantly when high uncorrelated normal noise is applied at a high sampling interval, as we will see later in [subsection 3.2.2](#).

2.5 Related Work Conclusion

To conclude the literature review: Even after a decade of ongoing research on applying HMMs in map matching, the accuracy or quality of the matches still remains unsatisfactory. A recent survey literature confirms this, stating that issues with accuracy still exist, specifically concerning ‘unnecessary detour, matching break and matching uncertainty’ (Chao et al. 2020). According to this work, unnecessary detours occur especially in situations with high-sampling measurements with measurement error. Furthermore, matching breaks occur when large outliers exist in a track. The authors also state that trajectory simplification ‘fails to prune outliers’, yet it still performs well because such algorithms ‘preserve the shape of the trajectory’ while reducing noise. Regarding matching uncertainty, especially in dense road networks, matching ‘is much harder’ due to the high ‘map density’, leading to more uncertainty for the optimizer, more unnecessary detours, and thus, less accuracy. In the following chapters, we will present our novel map matching approach, which addresses these issues.

3 Method

In this chapter, we will describe the method behind our novel map matching approach. First, we will outline the overall process and then we will explore its specific components. In preparation for a systematic evaluation of our approach, we will also present our new method for generating artificial map matching ground truth data with different noise characteristics.

3.1 Novel Map Matching Method

The general process of our map matching approach is shown in [Figure 3.1](#). As input data, we import a map (i.e., road network) and recorded GNSS points (i.e., tracks). In a preparatory step, we construct a directed graph from the map with additional spatial indices generated on the nodes and edges.

During the map matching initialization step, the processed map data (i.e., the directed graph and the spatial indices) is loaded, and the GNSS tracks are imported. On each track, first, our customized and improved trajectory simplification algorithm is applied. Then, for each simplified track, a candidate search is performed using the spatial indices to find the candidates. Next, our novel candidate adoption is performed to enrich the candidate sets with surrounding candidates. Finally, the track and the found candidate sets are passed to the stochastic optimizer.

In the optimization loop, for each pair of adjacent candidates under consideration (i.e., a combination), a shortest path route is computed on the

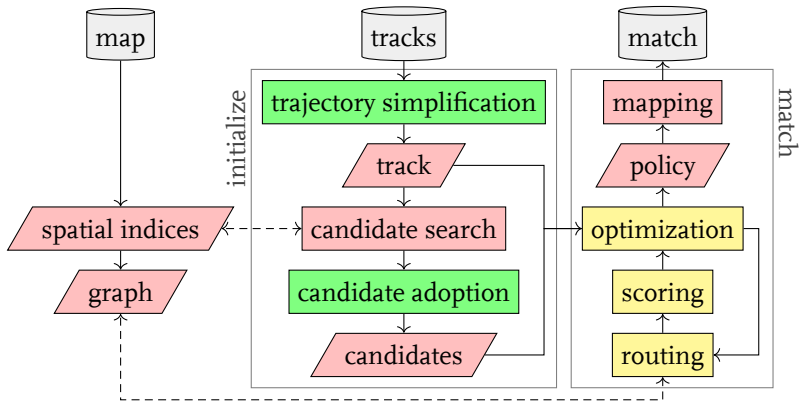


Figure 3.1: *Algorithm and data flow chart for our map matching approach* – We marked in **red** the map matching artifacts (diamonds) and processes (rectangles), in **green** our new and improved features to address outliers and strong noise of the records, and in **yellow** the parts of our new optimization model based on MDPs. This figure shows only the most important steps in our map matching process.

directed graph. Each combination is evaluated using defined metrics, resulting in a reward with MDPs or a probability with HMMs. By globally maximizing either the rewards or the probabilities over all combinations, an optimal policy can be determined. This policy (i.e., a sequence of actions or mapped road positions) is then converted into a match by concatenating the partial routes between the mapped road positions.

3.1.1 Trajectory Simplification

Trajectory simplification algorithms can improve the speed as well as the accuracy of map matching. By denoising the given tracks, they reduce the amount of data that needs to be processed. The cleaner track shapes then facilitate the stochastic optimization, which is intrinsically sensitive to noise. Thus, trajectory simplification is a critical step in making map matching more robust against noise and, to some extent, outliers. Our custom trajectory simplification approach, which we present in this work, has four stages.

In the first step, we remove all spatially duplicate points that are directly connected. This does not change the shape of the track, but it reduces the number of points for which a candidate search needs to be performed. Consequently, fewer candidates need to be processed, which improves computational performance. This can be seen in [Figure 3.2](#), which contains multiple points lying on top of each other in a stacked manner. In practice, these points have the same coordinates and cannot be visually distinguished.



Figure 3.2: *Trajectory simplification duplicates removal* – All spatially duplicate points are removed (i.e., adjacent points with the same geospatial coordinates). The dotted square spans multiple spatially duplicate points that are not visually distinguishable in practice.

In the second step, we apply a custom ‘warp-detection’ feature that removes obvious outliers. In our case, we remove points when the speed required to reach them exceeds > 1500 km/h, which is faster than any vehicle has ever traveled on Earth. Since speed is calculated per segment based on the time and distance between two points, either point in the segment can be removed. The removal is carried out in such a way that the remaining points are connected by the shortest remaining segment. In [Figure 3.3](#) is an illustration of this feature.

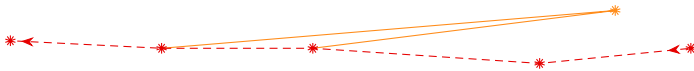


Figure 3.3: *Trajectory simplification warp detection* – In case a subsequent point of a track exceeds a very high speed to be reached, e.g., > 1500 km/h, which is calculated from the time and distance differences, it is detected as a so-called ‘warp’ between the positions (i.e., the [orange](#) point). Such highly improbable ‘warps’ are then removed.

In the third step, we apply a customized Douglas-Peucker algorithm. This algorithm simplifies geometric shapes by removing points within defined boundaries while keeping only those outside them, including the first and last points within the bounds. The Douglas-Peucker algorithm iteratively creates a line between the start and end points of a track, divides the track

at the point with the greatest distance from the boundary, and proceeds recursively in the same manner with the two resulting parts. When a part of the track is reached where all points between the current start and end points are within the boundary, they are removed, as shown in [Figure 3.4](#).

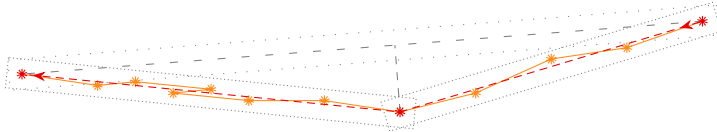


Figure 3.4: *Trajectory simplification Douglas-Peucker algorithm* – The Douglas-Peucker algorithm is applied for removing the [orange](#) points that blur the actual [dashed red](#) movement. Only points that lie within small, defined bounds (i.e., dotted gray rectangles) are removed, excluding the first and last points. This also applies to small reverse movements, provided that the bounds are not exceeded.

The main customization of our Douglas-Peucker algorithm implementation is a setting that enables extra handling of zigzag- or reverse-movements, which can occur in the matches. The regular Douglas-Peucker algorithm removes such reverse-movements, which is beneficial for reducing noise in the input track. However, this alters the shape of a match when compared to a provided solution, as described in our comparison algorithm in [subsection 3.1.4](#). This algorithm internally uses our customized Douglas-Peucker algorithm in a preparatory step while keeping the zigzag-movements. The regular Douglas-Peucker algorithm would remove these zigzag-movements, which produces incorrect results when comparing matches in which one has a reverse-movement and the other does not.

Definition 3.1 (Median-Merge Algorithm): The median-merge algorithm collapses points within a defined radius into the median point coordinates of the point-cloud. Such point-clouds usually occur when GNSS measurements continue around static positions.

In the final step, we apply our custom median-merge algorithm, which is illustrated in [Figure 3.5](#). Especially for tracks that jitter, i.e., at stop lights, this significantly reduces the amount of noise clusters. In these situations, we ensure to only collapse points directly connected within the search

radius. This way, we prevent removing points from later parts of the track that revisit the same location.

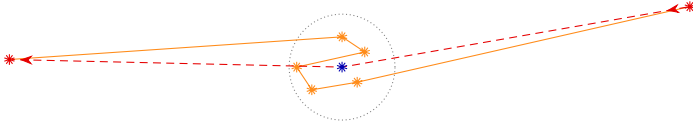


Figure 3.5: *Trajectory simplification median-merge algorithm* – The median-merge algorithm collapses the strong noise (i.e., orange points) around a static position to the median blue point, so that the actual dashed red movement is retained. Only if multiple points are connected within small, defined bounds (i.e., marked by dotted gray circle) a merge to the median point is performed.

By default, we use a radius of 10 m for our median-merge algorithm. We inferred this value from the fact that a regular personal vehicle is usually < 5 m long and occupies a traffic lane that is usually < 4 m wide, at least in the European Union. With a radius of 10 m, we allow for an error margin of twice this amount in all directions when such a vehicle stands in a static position on a street.

The median-merge algorithm contains an adaptive distance component for point-clouds far away from roads. This can occur, e.g., when a tracking device is taken off-road to a static location where measurements are taken in an area far away from roads. Such point-clouds are usually larger (i.e., over a longer time) than regular ones, e.g., at stoplights, and are more difficult to match due to their distance from the roads. The adaptive distance component enlarges the median-merge radius by up to half the distance to the next road to address such off-road situations. This behavior is optional but enabled by default.

If timing- or speed-dependent metrics are used, caution must be taken with our trajectory simplification steps because they alter the average speed and time differences between successive points. However, our own metrics only rely on geometric terms, as shown in [subsection 3.1.7](#).

3.1.2 Candidate Search

Since we don't know the ground truth position of any measured GNSS point within the road network, there are technically unlimited mapping possibilities. In other words, the point could be mapped to any position along any edge of the directed graph. To reduce this to a reasonable amount, candidate search is performed in map matching. The candidate search has two combined constraints. First, only the closest point along each candidate edge is chosen. Second, only candidate edges within a certain range are considered. The second constraint is based on the hypothesis that the measured GNSS points are close to the nearby road edges. Distant road edges, for example, in another city, are irrelevant for the current track point.

Candidate search creates for each track point a set of candidate points that lie exactly, i.e., within technical limitations, on nearby edges of the road network. Map matching then finds a match by computing routes between subsequent candidates in the same direction as the original track, mapping each track point to one candidate. The key challenge is to find the list of candidates that most optimally (i.e., according to defined metrics) represents the given track.

Selecting good candidate sets is crucial for finding good matches. If too few candidates are chosen, a good solution may not be found. However, selecting too many candidates may not improve accuracy any further due to irrelevant data. This can drastically increase the computational load and may also introduce additional uncertainty into the model due to the large number of possibilities. Thus, finding the right balance between matching speed, computational resources, and matching accuracy is challenging.

Candidate search is typically performed using either a circular search or a k -nearest neighbor search around each GNSS point, or a combination of both. The combined method uses the k -nearest neighbor search to limit the number of candidates in dense areas and subsequently uses the circular radius to limit candidates by maximum distance in sparse areas. This combined method is illustrated in [Figure 3.6](#).

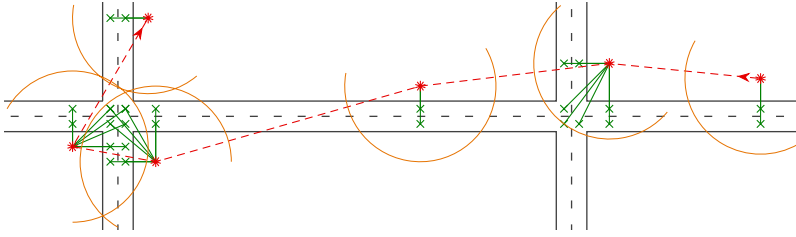


Figure 3.6: *Candidate search* – A didactic example of a candidate search. For every **red** recorded position, a spatial k -nearest neighbor candidate search within a certain radius (i.e., **orange** circle) is performed for finding all possible road positions (i.e., candidates, marked by **green** cross-marks). This so-called combined candidate search creates a candidate position on each road in every direction within the candidate search limits.

Since each candidate is bound to a road, it automatically inherits its direction. This means that, for a typical road that can be traveled in both directions, two candidates share the same spatial location but point to two different edges with opposing directions. This is because bidirectional edges are transformed into two spatially equal but oppositely directed edges in our graph model.

Concerning the circular candidate search limit, in areas with large distances between measured points, it is often unclear whether the large distances are caused by high noise, outliers, or by actual large distances traveled (i.e., low sampling interval). Therefore, the safest approach is to use large search circles. However, in dense areas of the same track with short distances between the measured points, large search circles could result in extremely high numbers of candidates. In such cases, a small search radius would be preferable.

Definition 3.2 (Adaptive Circular Candidate Search): An adaptive circular candidate search is a technology that adjusts the search radius dynamically depending on the situation. In our method, this is achieved by checking the distances between the current track point and the previous and next adjacent track points, using the smaller distance if that is still larger than the configured minimum search radius.

To address such cases, we use an adaptive circular candidate search, which reduces the number of candidates found in dense areas for high-sampled tracks. Additionally, when no candidates have been found for a given track point, we double the search radius up to a configurable upper limit. Thus, depending on the situation and the given radius limits, our search circle can increase or decrease. Consequently, we can dynamically use smaller search circles in dense regions with high-sampled tracks and larger ones in sparse regions with low-sampled tracks.

3.1.3 Candidate Adoption

Since one of the main challenges in map matching are outliers and strong noise, we specifically developed our novel candidate adoption feature for addressing this topic. For regular GNSS noise, the usual candidate search is sufficient for finding different alternative candidates for the stochastic optimization to probe and select from. However, in the presence of strong noise and outliers, the usual candidate search provides the stochastic optimizer with corresponding noisy and outlying candidates. This leads to inaccurate results, since stochastic optimization is generally sensitive to strong noise and outliers, which introduce uncertainty into the model and blur the optimal path probabilities.

Our idea with candidate adoption is to make the stochastic optimizer ‘context-sensitive’ to render it capable of dealing with strong noise and outliers by ‘collapsing’ them into the most probable positions. With candidate adoption, the local errors that strong noise and outliers introduce into the model can be ‘bypassed’. This makes the stochastic optimizer practically insensitive to strong noise and outliers, i.e., it becomes robust to them.

In [Figure 3.7](#) we show a didactic example of the candidate adoption functionality. By enriching the candidate sets of all track points with the adjacent and nearby candidate sets, the stochastic optimizer becomes able to collapse outlier points that would otherwise increase the matching error.

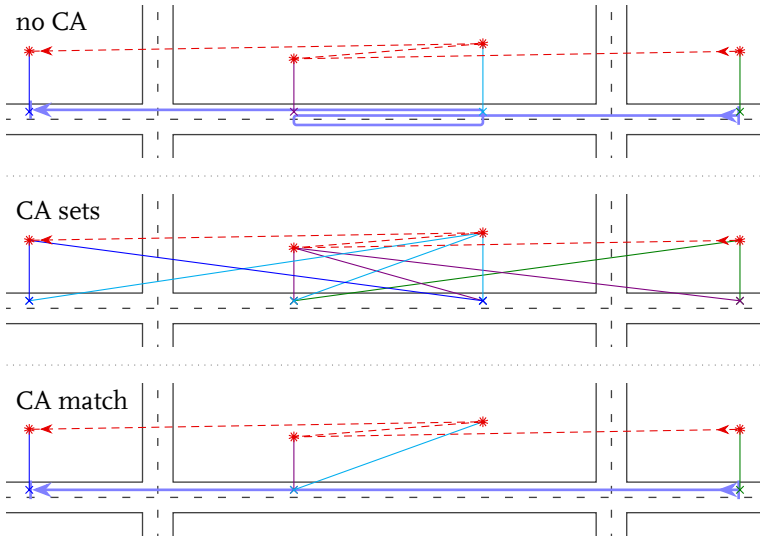


Figure 3.7: *Candidate adoption (CA)* – For every red GNSS point, a candidate search is performed. In this example, only the nearest edge is selected. Without candidate adoption, this results in the lightblue match containing a zigzag-movement with two U-turns in the middle of the road. With candidate adoption, for every GNSS point, the nearby (in this case sibling) candidate sets (i.e., blue, violet, cyan, and green lines) are adopted. This allows the second and third points to be collapsed into the same candidate road position, eliminating the U-turn.

In practice, we use two stages of candidate adoption (CA), as shown in Figure 3.8. Both stages are performed sequentially while adopting candidates no more than once.

In the first stage, ‘CA siblings’, we adopt for each track point the candidate sets from the preceding and succeeding track points. With this procedure, each candidate (i.e., road position) is referenced by up to three track points. This allows for collapsing any singular outliers, regardless of distance, into the same road positions as the adjacent ‘non-outlier’ track points.

The second stage, ‘CA nearby’, is specifically designed for dealing with strong noise or jitter in static positions, for example, when waiting at a stoplight with continued GNSS measurement, that has not been resolved

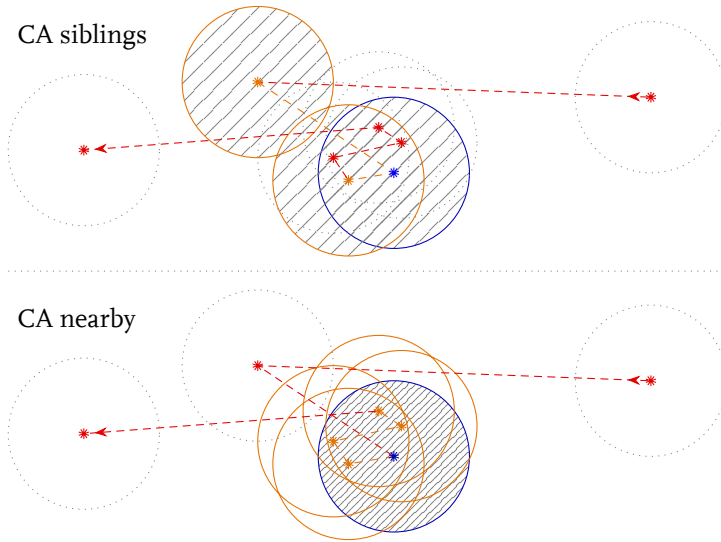


Figure 3.8: *Candidate adoption (CA) technology levels* – The first level is ‘CA siblings’, which adopts for each red track point the preceding and succeeding candidate sets, e.g., for the blue point candidate set (i.e., blue line pattern circle) the sibling candidate sets (i.e., orange line pattern circles around orange points). This results in the line patterned area candidate set for the blue point. The second level is ‘CA nearby’, which adopts all candidates from other candidate sets (i.e., orange circles) that are located within the range of its own candidate set (i.e., blue circle), resulting in the density increased candidate set around the blue point (i.e., denser line patterned area).

by trajectory simplification. In this step, all candidates from any other close candidate sets within the bounds of the own candidate set are adopted. This allows a single road position to be referenced by many surrounding track points for potential collapsing.

With candidate adoption enabled, erroneous GNSS measurements, strong noise, and outliers can be collapsed for reducing the overall matching error by minimizing detours and U-turns. Candidate adoption leverages outlier and strong noise removal onto the stochastic process instead of using manual trajectory simplification algorithms. Nevertheless, since trivial noise is already resolved by our trajectory simplification algorithm,

the stochastic optimizer can focus on the more complex situations that are outside the scope of trajectory simplification.

This also indicates that trajectory simplification and candidate adoption work best in combination. Without trajectory simplification, thus keeping small noise and point-clouds, candidate adoption would adopt many candidates for trivial situations. This would drastically increase the candidate density, thereby dramatically impacting the map matching speed. However, trivial situations can be resolved with trajectory simplification algorithms, candidate adoption is not necessary. Thus, using trajectory simplification for removing trivial noise and unnecessary track points leaves difficult, complex, and strong noise and outlier situations to candidate adoption. This creates a balance between the computational impact of candidate adoption and the improvement in matching accuracy of trajectory simplification and candidate adoption in combination, as we will confirm later in our benchmarks in [chapter 5](#).

3.1.4 Comparison Algorithm

Comparing matches with the provided solutions is more challenging than it first seems. Previous map matching data sets generally provide the solutions as lists of edge IDs. Consequently, the matches have to be encoded as lists of edge IDs to be comparable against the solutions ID by ID. This allows computing the accuracy after [Equation 2.2](#) by extracting the corresponding edge lengths for each ID. However, as discussed in [subsection 2.4.2](#), this approach has limitations. For example, matches do not always span complete edges from node to node, but rather can start anywhere along an edge. Thus, comparing IDs introduces significant error in the results. Although the list of edge IDs could be converted into a geometric line, we have not yet encountered any algorithms that can calculate the accuracy for a geometric line compared to another geometric line after [Equation 2.2](#). The main reason for this lies in the floating-point inaccuracies in modern computers, i.e., geometric coordinates as floating-points generally cannot be directly compared when they come from different sources because they are not ‘binary equal’. Since candidates are computed as new points along the

edges, they are usually not binary equal to previously computed solutions that have been converted to geometric lines.

Therefore, we have developed a comparison algorithm that can accurately compare provided solutions with matches from our novel map matching approach, as well as third-party solutions, when all data is provided as geometric lines. This algorithm can extract the similarities and differences between two geometric lines, even when they have floating-point inaccuracies or different GNSS precision. The key idea behind our comparison algorithm is to converge the two given geometric lines within defined parameters so that the floating-point inaccuracies and precision differences are equalized. This allows both lines to be compared segment by segment by simply following them with two markers in one sweep.

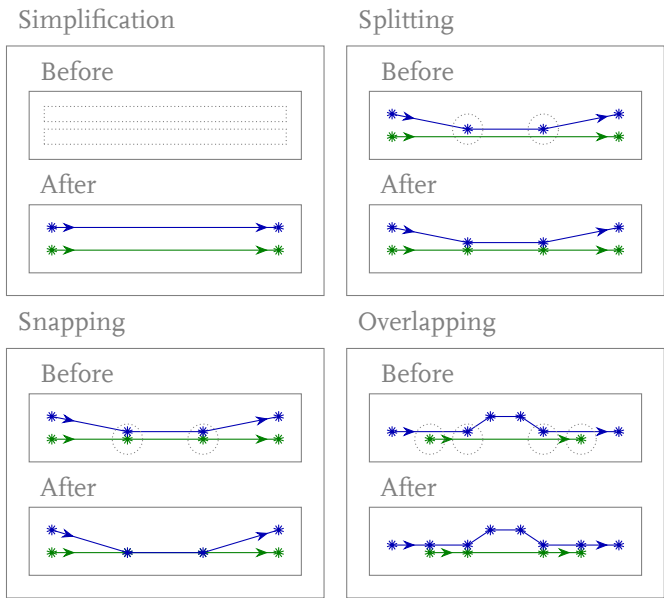


Figure 3.9: *Comparison algorithm steps* – The steps contain the Douglas-Peucker algorithm simplification, to reduce the number of points for eliminating tiny noise, the splitting of the tracks in close positions, and the snapping of points close to both tracks. This sequence of splitting and snapping creates comparable segments, even for tracks with partially overlapping lines.

Before the actual algorithm starts, trivial cases are checked, such as when one or both of the provided lines are empty. If this is not the case, the lines are prepared for the segment-wise comparison, as shown in [Figure 3.9](#). The key concept here is to align the segments of both lines according to a provided alignment configuration. First, a narrowly bounded custom Douglas-Peucker algorithm is applied on both lines to remove unnecessary points while retaining zigzag-movements, ensuring that U-turn round-trips remain detectable. Then, points close to each other are snapped to the same binary coordinates. Next, new points are introduced by splitting the respective segments when one line does not contain a point close to a corresponding point on the other line. This is followed by another round of snapping of the newly introduced points. The intermediate results of this preparatory step are two geometric lines with binary equal points for all overlapping segments.

Now, the comparison algorithm checks for each segment if the segments are equal by performing a simple binary equality comparison. If they are not equal, then either the lines drift apart in different directions, or one line reverses while the other does not. In both cases, the different segments can be summed up until both lines re-merge and overlap again. This allows for extracting the correctly matched segments, as well as the segments that the match missed or added over the solution. From this data, the accuracy of the match to the solution can be calculated after [Equation 2.2](#).

Our complete comparison algorithm can be reviewed in our open source software reference implementation Map Matching 2¹⁹.

3.1.5 Hidden Markov Models and Algorithms

To gain a more thorough understanding of the differences between our novel MDP to the HMMs used in previous map matching algorithms, we first describe the HMM in map matching, as defined in [section 1.6](#).

Hidden Markov Model

A hidden Markov model (HMM) consists of ‘hidden states’ and ‘observations’. The idea is that the true states are unknown, but a most probable sequence of states can be computed using a sequence of observations of the situation. Each observation can have multiple hidden states assigned with a probability of occurrence, i.e., an ‘observation probability’. Due to the Markov property, a state depends only on its directly preceding state. Thus, when a state transitions into multiple different states, each transition has an independently assigned probability, i.e., a ‘transition probability’.

The key idea of HMMs in map matching is that the GNSS measurements can be represented as the observations, and the candidates can be represented as the hidden states. Then, the probabilities can be computed according to the metrics, for example, from length differences and distances. Consequently, some sequences of hidden states are more probable for the given observations than others. An example of a HMM for map matching can be seen in [Figure 3.10](#).

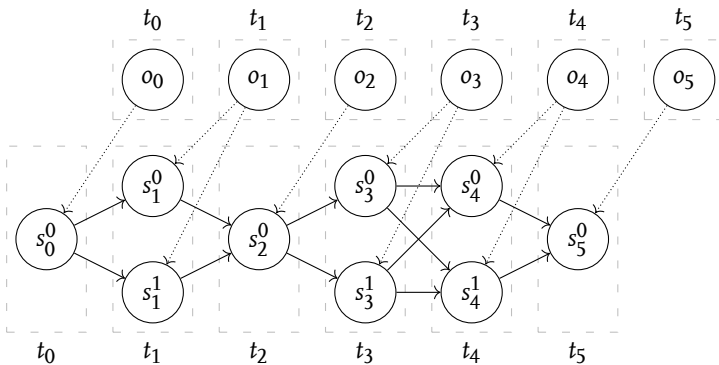


Figure 3.10: *Hidden Markov model (HMM) for map matching* – In this example, every state s_n^m has assigned an observation o_n . Each solid arrow between two states has a unique *transition probability*, and each densely dotted arrow between an observation and its assigned state has a unique *observation probability*. For visual clarity, the observations are visually shifted to the corresponding hidden states in the same time steps t_n .

The Viterbi Algorithm

The Viterbi algorithm is then used to determine an optimal sequence of hidden states based on the computed probabilities. It calculates, for each hidden state at each time step, the highest probability of the sequence ending in that state, as well as the previous state from which that highest probability originated. This is achieved by multiplying the probability of reaching the previous state with the probability of transitioning from the previous state to the current state, and with the probability of observing the current observation given the current state. This process uses ‘dynamic programming’, which means that intermediate results are stored for avoiding redundant (i.e., brute force) computations. When there are multiple paths that lead to one state at a given time step, only the path with the highest probability is kept, and the others are discarded. The probabilities and previous states are computed iteratively from the first to the last time step. Once the final time step has been reached, an optimal path can be backtracked by following the list of the highest probability states back to their previous states from which they originated. The final result is a path with the highest overall probability, i.e., the most probable sequence of candidates.

3.1.6 Markov Decision Process and Algorithms

As we have seen with the HMM, this is a ‘top-down’ approach that models the complete situation within one model to solve it. In other words, this approach leads to a ‘static’ model that is precomputed prior to solving it. When, for example, there is no route between all hidden states between two time steps, e.g., due to an ‘off-road condition’ where a road is missing in the road network, the HMM needs to be split into two models, as a complete solution cannot be computed with missing transitions. However, solving the two HMMs individually may not lead to an overall globally optimal solution, but rather two locally optimal solutions.

Markov Decision Process

In our novel map matching approach, we redesigned the way map matching is performed. Let's imagine map matching as having an agent that is located somewhere in the road network, with a GNSS measurement representing its current position. The task for this agent is to move to the next road position that is represented by the next GNSS measurement, until no more GNSS measurements are available to move to. The goal is to find a sequence of movements (i.e., actions) in the road network that best represents the provided sequence of GNSS measurements. To achieve this goal, the agent receives a reward for each action it takes, with a higher reward when the movement better represents the current track segment between the recent and the next GNSS measurements. In this way, a Markov decision process (MDP) from the reinforcement learning domain can be defined. Our MDP design for map matching, as defined in [section 1.6](#), can also be seen in the illustration in [Figure 3.11](#).

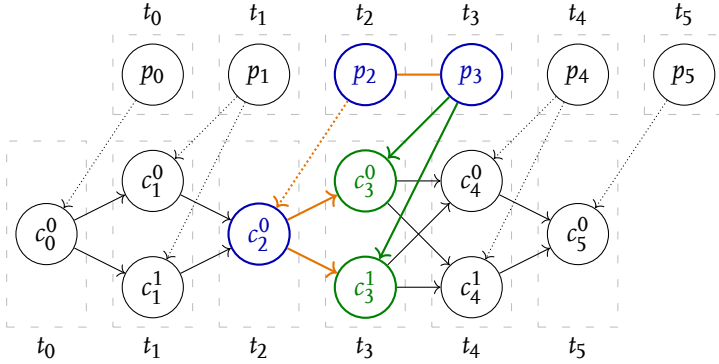


Figure 3.11: *Markov decision process (MDP) for map matching* – In blue is a possible state $s = (p_n, c_n^m, p_{n+1})$, which is composed of the current road position, defined by the current track point p_n and current candidate c_n^m at time step $t_n \in T$, and the next track point p_{n+1} , which is the direction or goal to move to at the next time step t_{n+1} . In green are the possible actions $a \in A_s$ that allow to choose the next candidate c_{n+1}^a . The reward after [Equation 3.3](#) is computed based on the orange lines and the chosen green action.

This map matching design has multiple advantages. First, we can use machine learning without labeled data, i.e., sampling own experience with reinforcement learning, which allows for various new algorithms to be applied to map matching that were not accessible before. For example, it enables the use of temporal-difference (TD) algorithms, such as Q-learning, which can find matches by visiting only a fraction of the state space. This saves computational resources and guarantees suboptimal solutions within strict time constraints. At the same time, it allows dynamic programming algorithms, such as Value Iteration or Policy Iteration, to guarantee optimal solutions.

Second, the ‘dynamic’ model approach of the MDP in exploring the state space incrementally enables graceful recovery from off-road situations, i.e., when roads or connections are missing. In such cases, within the same model instance, the agent can transition (i.e., ‘skip’ as in warping) to a subsequent candidate within the same model rather than being trapped by the network’s structure. In other words, our MDP is inherently capable of handling missing data, i.e., missing candidates, unlike the typical HMM design, which requires workarounds for this issue. Since our model is not split, the partial matches are globally optimal together. This makes our MDP approach more robust when dealing with incomplete road networks than HMMs, which necessarily relies on connections between hidden states.

Third, the probabilities in HMMs have a normalization issue, i.e., there needs to be a defined limit for what is 1.0 and what is 0.0. This can lead to cutting off values that exceed the limit or making the probabilities very one-sided on the scale. In turn, this can lead to ‘vanishing’ probabilities because multiplying tiny floating-point numbers by large ones may not make a difference due to rounding errors in how modern computers store floating-point numbers. Our MDP works with absolute rewards that do not have normalization or vanishing number issues. This provides a more consistent scale, meaning large errors effectively discourage exploration of improbable states while allowing the agent to clearly distinguish between promising alternative paths.

There are various methods to optimize a MDP. The reinforcement learning reference book (Sutton and Barto 2018) contains a thorough overview of such methods. In our approach, we primarily use the tabular ‘dynamic programming’ method ‘Value Iteration’ for finding an optimal solution, and the tabular temporal-difference (TD) method ‘Q-learning’ to demonstrate the application of direct reinforcement learning.

Value Iteration

Definition 3.3 (Beneficial Preordering): Beneficial preordering organizes the state space in a cache-aware manner. In other words, states are arranged according to successive access patterns, enabling them to be prefetched from main memory into the central processing unit (CPU) cache. This improves computational performance when iterating through the states for optimizing policies.

Value Iteration shares similarities with the Viterbi algorithm in that both use a data storage to keep track of the most probable paths while discarding the less likely ones. In Value Iteration, the value (i.e., the expected cumulative reward) is computed for each state, which is the maximum reward that can be acquired by following an optimal policy from that state onwards. Each value is computed by taking all possible actions in each state, calculating the expected rewards by adding the immediate action rewards to the values of the subsequent states, and keeping only the highest reward. The process of iterating through all states and actions is also called a ‘sweep’. Initially, all values are unknown, meaning all subsequent values are also unknown except for the final time step, which has no subsequent states. In that case, the immediate action rewards are equivalent to the expected rewards. For this reason, Value Iteration needs multiple sweeps until it converges, i.e., until the values become stable. With each sweep, the values propagate from the last time step to the first one step by step until no further changes to the values are made. The efficiency of Value Iteration can be improved with techniques such as beneficial preordering (which effectively leads to reversely traversing the states) and ‘in-place’ updating of the values instead of exclusive data storage per sweep. These techniques allow for faster value

propagation and significantly reduce the number of sweeps needed. Once the values have converged, an optimal policy can be extracted by selecting the actions that lead to the subsequent states with the highest values.

Q-Learning

Q-learning is similar to Value Iteration in that it also uses a data storage to keep track. However, rather than storing the values of the states, the values of each action for each state are stored independently. Value Iteration calls all actions for each state, whereas Q-learning only needs to call one action per state with this storage design. Q-learning does not sweep through the whole state-action space, but rather iterates through individual paths in so-called ‘episodes’. Each episode involves taking one action in a state at each time step until the end of the MDP is reached, i.e., until the entire track is mapped. As such, Q-learning samples one match with each episode, which is why it generally needs many episodes to find a good match. The advantage is that not all states and actions need to be visited, and results can be obtained with only a fraction of the state-action space visited. However, an optimal solution cannot be guaranteed, except when unlimited episodes are performed, because only then the state-action values theoretically converge. The reason for this is that the usual way of performing action selection in Q-learning is by employing an ‘ ϵ -greedy’ policy. This means that with probability ϵ , a random action is selected to favor exploration of the state-action space, and with probability $1 - \epsilon$, the currently best action is selected to favor exploitation of learned behavior. Thus, random action selection requires an unlimited number of episodes to converge to an optimal policy. Therefore, in practice, it is advisable to stop the convergence early when no improvement to the currently best policy has been made after a defined number of episodes or elapsed time. This does not guarantee that the currently best policy is optimal, but it allows for stopping the convergence early to obtain a result. For example, in real-time scenarios or in scenarios with huge state-action spaces, this method can find suboptimal to optimal solutions within defined bounds, which may not be possible with Value Iteration that needs to sweep through the entire state-action space. To our knowledge, applying Q-learning from the reinforcement learning domain

directly to solving map matching without labeled data is a novel approach that has never been done before.

3.1.7 Map Matching Metrics

The metrics for HMMs and MDPs can both be computed from the same data, such as distances, lengths, and azimuths. However, for the HMM, they need to be computed as observation and transition probabilities, whereas for the MDP, they are computed as rewards.

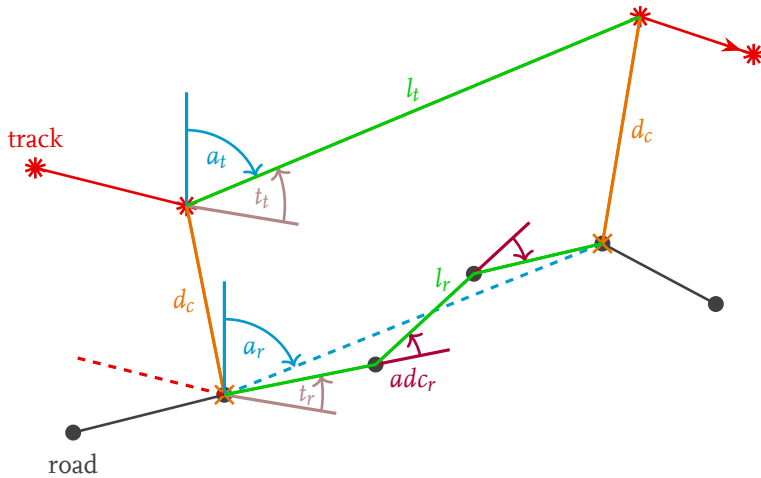


Figure 3.12: *Metrics for map matching* – The red exemplary track with four GNSS measurements is shown in comparison to a possible match on the black road network for the second segment of the track. For the second and third track points, the possible orange candidates are shown by the lines that cross at the second and fifth road points. The metrics from Equation 3.3 are marked in different colors. The distances (d_c) are in orange, the lengths (l_t and l_r) are in green, the clockwise azimuths (a_t and a_r) are in cyan, marked by the dashed cyan line for the route, the absolute direction changes (adc_r) are in purple, and the turns (t_t and t_r) are in pink, with the turn of the route being heuristic, i.e., dashed copy of the previous track segment.

The data used for our metrics is shown in [Figure 3.12](#). In our approach, only geometric data is used. This means that, in theory, any geometric line can be matched, even when the time information is missing. This can occur, for example, when a track is hand-drawn as a test case, or when tracks have been simplified in advance concerning their shape.

Metrics Terms

Definition 3.4 (Distance): The ‘distance’ is the geographic, spherical, or Cartesian distance between two geospatial points, depending on the spatial reference system. In our metric, we calculate the ‘distance’ between a track point and each candidate from the associated candidate set.

The hypothesis for the ‘distance’ metric is that the measured track point is close to the original position in the road network due to GNSS error distributions. Therefore, a shorter distance between the track point and the candidate road position results in a smaller error.

Definition 3.5 (Length): The ‘length’ is calculated after the point-to-point distances for lines and segments. To distinguish the ‘length’ from the ‘distance’, we use the term ‘length’ for lines and segments that are part of either the track or a route, but not for other point distances. In our metric, we compare the length of a track segment to the respective calculated route length.

The hypothesis for the ‘length’ metric is that the length of the route is close to the length of the track. If the route is significantly longer or shorter than the track segment, it is less likely than a route that accurately reflects the movement performed.

Definition 3.6 (Azimuth): The ‘azimuth’ is an angle on the unit circle in a clockwise direction between $0^\circ < 360^\circ$ with 0° being north, 90° being east, 180° being south and 270° being west. Each segment of a line has an ‘azimuth’, and each line has a ‘total’ azimuth from the first to the last point.

We compare the ‘azimuth’ of a track segment to the respective azimuth of the calculated route.

The hypothesis for the ‘azimuth’ metric is that the track and route go in the same direction, thus having a similar azimuth. When comparing two azimuths, we compute the difference between the two angles, which gives a result in the range between $-180^\circ < 180^\circ$. Negative values represent angles to the right, and positive values represent angles to the left, which stems from the calculation formula used. We also account for overflowing angles. For example, comparing an angle of 355° to 5° results in a difference of 10° .

Definition 3.7 (Direction Change): The ‘direction change’ is an angle between two adjacent segments of a calculated route. We calculate the direction changes using the differences between the adjacent azimuths of the route segments. The direction change shows the angle between the continuation line of the first to the second segment, not the angle between the two segments.

To our knowledge, the ‘direction change’ in this definition is new to map matching metrics. For example, a ‘direction change’ of -10° is a slight movement to the right, whereas the angle between the two segments would be 170° . In our metric, we use the sum of the absolute direction changes of a route, ensuring that all direction changes contribute, rather than letting positive changes cancel out negative ones. The hypothesis is that routes with a lower total sum of absolute direction changes (i.e., straighter routes) are more likely than routes with many ‘round-trips’ or ‘U-turns’. This term works particularly well in combination with candidate adoption because it facilitates the collapsing of candidates over detours.

Definition 3.8 (Turn): The ‘turn’ is defined as the direction change between two adjacent track segments. The ‘turn’ is also applied to the first segment of the computed route so that the difference between the track and the route turns can be calculated.

The final term in our metric is the ‘turn’. The hypothesis is that routes starting into the same direction as the track turns into (e.g., at junctions) are more likely than routes that start into, for example, opposing directions. It has to be noted that this is a heuristic approach, as we cannot compute the turn of the route, for example, when it starts inside a junction with multiple incoming segments.

In conclusion, the metric terms indicate that the closer the road is located to the track, the fewer the direction changes along the route, and the smaller the differences between the azimuth and turn angles, the better the route fits to the track segment, and the more likely it will be part of the final match.

Hidden Markov Model Probabilities

For the HMM, the weighted distances are used for the observation probabilities, see [Equation 3.1](#).

Equation 3.1 (Observation Probability):

$$\mathbb{P}_O = [w_d * \hat{d}_c] \quad (3.1)$$

The remaining data, i.e., the weighted lengths, azimuths, direction changes, and turns, are used for the transition probabilities, see [Equation 3.2](#).

Equation 3.2 (Transition Probability):

$$\begin{aligned} \mathbb{P}_S = & [w_l * |l_t - \hat{l}_r|] * [w_a * |\delta(a_t, a_r)|] \\ & * [w_{adc} * [a\hat{d}c_r]] * [w_t * |\delta(t_t, \hat{t}_r)|] \end{aligned} \quad (3.2)$$

Since probabilities are defined between $0.0 \leq 1.0$, the metrics are normalized relative to a reference value. For example, the distances are normalized to the largest distance within a candidate set, the lengths are divided by the

longer of the two, and the azimuth differences are normalized to 180° . With the direction changes, normalization is challenging because some routes may have significant absolute direction changes while others do not. Using a very large divisor for normalization makes the probabilities of small *adc* terms very small, which can cause vanishing probabilities. However, when we use a small normalization value, we cut off any large *adc* terms, so we cannot distinguish between them anymore. Additionally, we don't know how high the largest of these terms will be for the respective track segment at the time of computing the *adc* term unless we take a second pass. For these reasons and because HMMs are not our primary research focus, we simply normalize the (absolute) *adc* term to 360° without a second pass. The hypothesis is that the routes between adjacent candidates are mostly very short and contain few direction changes. They should also not contain more than one turnaround to be considered a good match. Nevertheless, this solution has the mentioned drawbacks. However, our new MDP does not suffer from this particular limitation, since normalization of the data is not necessary. Therefore, we believe that this compromise for our HMMs is reasonable.

Markov Decision Process Rewards

For the MDP, the metrics don't need to be normalized and can be used directly, since we use absolute values instead of probabilities. However, different units, such as meters or degrees, have to be respected by the weights so that they can be summed appropriately, depending on the desired impact. A sum of different reward terms is sometimes also referred to as a 'mixed reward' in the literature. In MDPs, positive or negative rewards can be used. The agent's goal is to maximize the rewards, or in other words, to avoid negative rewards. Thus, we can define the reward for our MDPs as a negative mixed sum of individually weighted absolute error terms, as shown in [Equation 3.3](#). Since all terms in our reward formula are positive numbers, negating the sum always gives a negative reward. Therefore, the larger the errors of the individual terms, the lower the reward. In other words, the smaller the error, the higher the reward and the closer it is to

negative zero. Maximizing the reward means avoiding the more negative rewards.

Equation 3.3 (Reward):

$$r = - \{ [w_d * d_c] + [w_l * |l_t - l_r|] + [w_a * |\delta(a_t, a_r)|] + [w_{adc} * adc_r] + [w_t * |\delta(t_t, t_r)|] \} \quad (3.3)$$

The weights have been hypertuned by applying `gp_minimize` from Scikit Optimize²². They are described in more detail later in [section 5.2](#).

3.2 Artificial Ground Truth Generation

As we have seen in the literature review in [subsection 2.4.1](#), there are only a few map matching data sets, and even fewer are ground truth. For this reason, we also present in this work an open process based on open source software for artificially generating map matching ground truth data.

First, we simulate ground truth routes on virtual road networks using a microscopic traffic simulation software. Then we export GNSS measurements along these routes as tracks. Next, we apply artificial autocorrelated GNSS noise to the tracks based on the Ornstein-Uhlenbeck process. This step allows for applying different noise settings to create diverse data sets with various noise characteristics for evaluating map matching algorithms. Finally, in [section 5.5](#), we match the artificial GNSS tracks with different map matching algorithms and compare the results with the generated ground truth for evaluating the individual strengths and weaknesses of the various algorithms according to different noise characteristics.

The entire process can be reviewed step-by-step in [Figure 3.13](#). We use OpenStreetMap road network data and the open source traffic simulation software Simulation of Urban MObility (SUMO). First, the OpenStreetMap

²²https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html

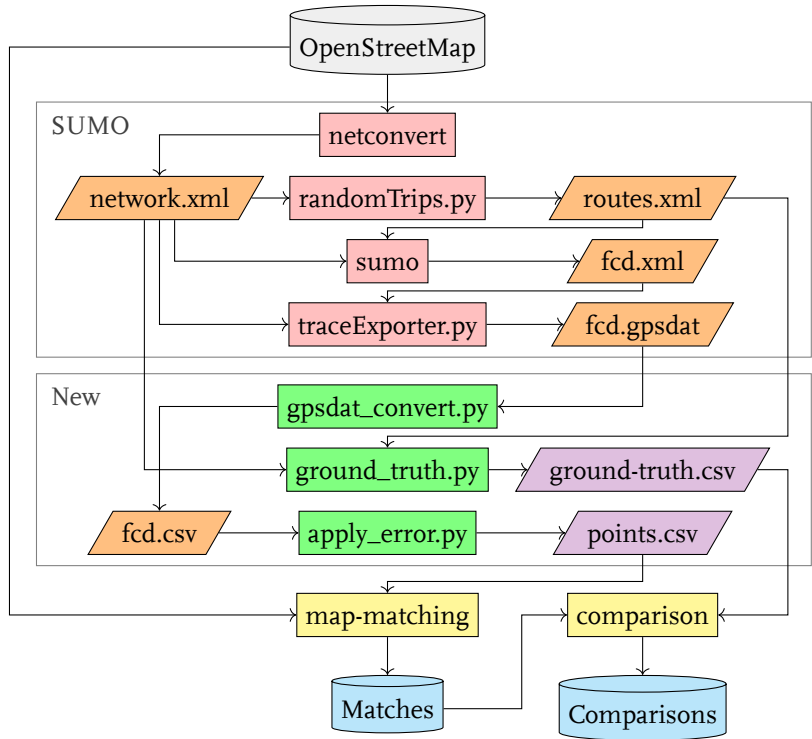


Figure 3.13: *Technology roadmap for our artificial ground truth generation open source tool-chain* – The rectangle shapes show processes and tools, the trapezium shapes show artifacts, and the cylinders show the input and output data. The processes and tools in the ‘SUMO’ frame are provided by the application Simulation of Urban Mobility (SUMO), the tools in the ‘New’ frame below are newly provided by us. SUMO simulates the movements as floating car data (FCD) measurements. The ‘ground-truth.csv’ and ‘points.csv’ files are the essential results of our pipeline that undergo further matching and evaluation. The ‘map-matching’ and ‘comparison’ processes use existing open source algorithms. The rest are intermediate files.

road network is converted into the SUMO format using the provided ‘net-convert’ tool. We use some additional parameters for reducing the internal lane and junction area sizes, ensuring that the emitted GNSS points remain close to the original OpenStreetMap geopositions. Without these parameters SUMO ‘inflates’ the road lanes and junctions so that virtual vehicles

have more realistic space. However, map matching algorithms usually don't take into account lanes or positions within junctions, since this data is not directly present in road networks such as OpenStreetMap. Next, we generate random movements using the provided SUMO tool 'randomTrips.py'. Instead of random movements, more sophisticated trip generation algorithms may be used, such as origin-destination-matrices (OD-matrices) or activity plans. However, random movements can be generated for any region or city without additional knowledge, and the focus here is on creating data sets for evaluating map matching algorithms, not for evaluating realistic regional movement patterns. The generated movements are then simulated with the 'sumo' application, i.e., the generated routes are driven by virtual vehicles. During this process, the GNSS data is emitted as floating car data. This raw floating car data output contains no GNSS error, since there are no sources of measurement error in the virtual emission process. After conversion of the raw floating car data into an intermediate file format with the 'traceExporter.py' utility, the floating car data can be converted into a comma-separated values (CSV) file with the 'gpsdat_convert.py' script into well-known text (WKT) point representations for facilitating further processing in our pipeline. We also export the ground truth routes from the SUMO network, i.e., the exact OpenStreetMap edges, into WKT line representations to prepare them for our comparison algorithm. Finally, we apply our artificial GNSS noise generation algorithm to the floating car data tracks. Then, the noisy GNSS points can be matched with a map matching algorithm, and the resulting matches can be compared with our comparison algorithm with the extracted ground truth. In this way, we can evaluate the accuracy of a map matching algorithm depending on the defined artificial noise characteristics of the simulated floating car data tracks.

3.2.1 Artificial GNSS Noise

We primarily use the Ornstein-Uhlenbeck process to apply artificial GNSS noise. For comparison purposes, we also provide an ARMA model and a simple Gaussian distribution model. Additionally, we provide moving-average smoothing for accounting excessive jittering of the underlying

Gaussian random values that are used as a base for all models. Furthermore, we provide a simple random outlier generation process and a missing measurement process with parameterizable distribution closeness parameters describing the likelihood of outliers or missing points appearing directly next to each other for simulating potential satellite opacity. The outliers are sampled from a Gaussian distribution because they do not follow the autocorrelation of regular GNSS noise, and the missing points are simply discarded when selected. Finally, we provide additional settings and tools for further analysis of the generated noise distributions, such as statistical analyses and plot generation. In the following sections, we will also compare the artificially generated GNSS noise with real GNSS recordings to evaluate the similarities and limitations in practice.

The Gaussian (normal) distribution after Equation 3.4 is uncorrelated and, despite being frequently used in previous research, does not accurately represent GNSS error over time.

Equation 3.4 (Gaussian Normal Distribution):

$$X \sim \mathcal{N}(\mu_0, \sigma_N^2) \quad (3.4)$$

The autoregressive moving-average (ARMA) process after Equation 3.5 is autocorrelated, but it contains no term for mean-reversion. This means that it deviates further from the defined mean, such as zero, the longer it runs. The white noise term is sampled from a simple Gaussian distribution.

Equation 3.5 (Autoregressive Moving-Average Process):

$$X_t = \varepsilon_t + \sum_{i=1}^{o_p} \varphi_i X_{t-i} + \sum_{i=1}^{o_q} \theta_i \varepsilon_{t-i} \quad (3.5)$$

The Ornstein-Uhlenbeck process with the Euler-Maruyama method after Equation 3.6 contains a mean-reversion term, which keeps it close to the zero-mean even over long periods of time. This makes it more similar to GNSS noise autocorrelation. The Wiener process term is also sampled from a simple Gaussian distribution.

Equation 3.6 (Ornstein-Uhlenbeck Process):

$$X_t = X_{t-1} + \theta(\mu - X_{t-1})\Delta t + \sigma W_{t-1}\sqrt{\Delta t} \quad (3.6)$$

Regarding the additional moving-average smoothing of the sampled noise, for high-sampled noise the autocorrelated processes still jitter more than regular GNSS noise due to the randomness of the underlying Gaussian distribution. With low sampling intervals, the smoothing can gradually be disabled, as the autocorrelation decreases with longer time distances between the samples. By default, the smoothing decreases gradually as the sampling interval increases, though this can be adjusted as necessary.

3.2.2 Noise Distribution Comparison

To illustrate how the noise generation methods described above compare to each other, we created exaggerated noise distributions for an example track using simple Gaussian normal noise, ARMA noise, and Ornstein-Uhlenbeck process noise. The seed for the random generation is the same for all variants. We show the original simulated track with smoothed noise after applying a moving average smoothing with a window size of 10, as well as the original and smoothed noise coordinates on both axes over time. Additionally, we show the autocorrelation function (ACF) plots for both axes.

As shown in [Figure 3.14](#), the simple Gaussian noise does not produce practical results for high-sampled tracks because it is not autocorrelated. This results in very irregular noise patterns over time.

The ARMA process noise in [Figure 3.15](#) is much more authentic. However, it diverges from the zero-mean over time, leading to larger differences between the input and output tracks, especially at the latest time steps.

The autocorrelated, mean-reverting Ornstein-Uhlenbeck process in [Figure 3.16](#) is the only noise generation process in comparison that is suitable for long-term, high-sampling noise generation. Compared to the ARMA

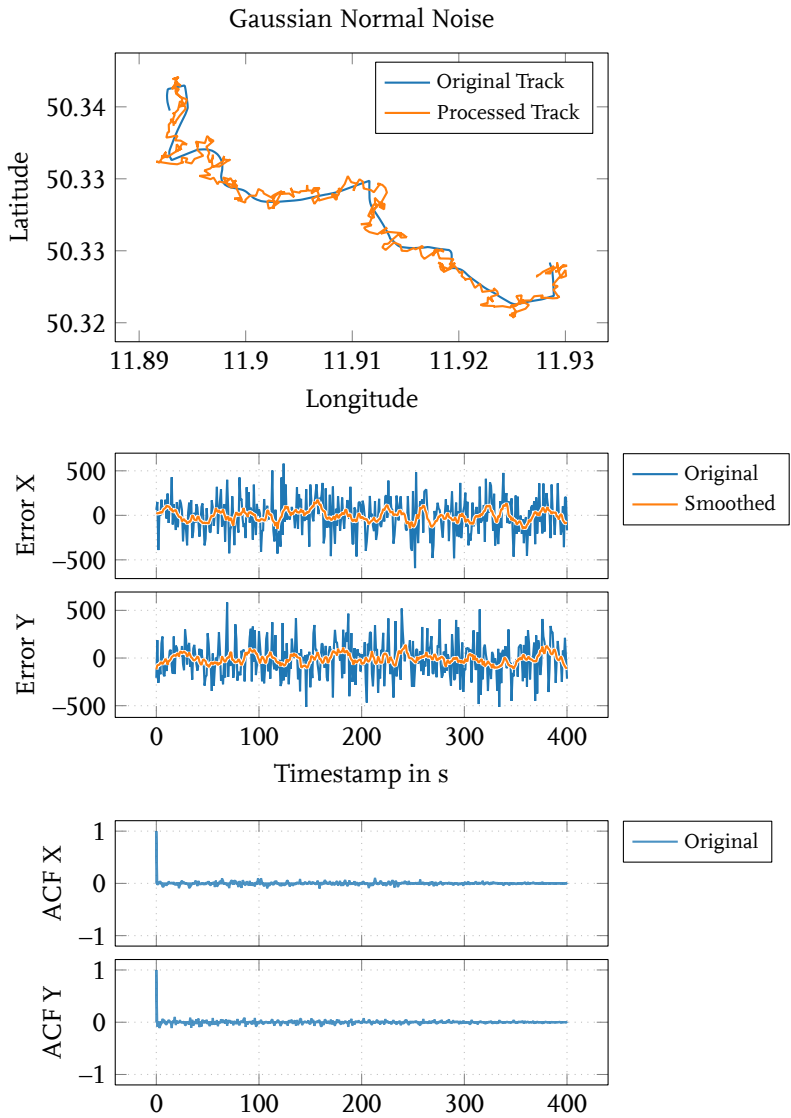


Figure 3.14: *Artificial GNSS Gaussian normal noise* – The Gaussian noise is not autocorrelated, as can be seen in the coordinate and autocorrelation function (ACF) plots.

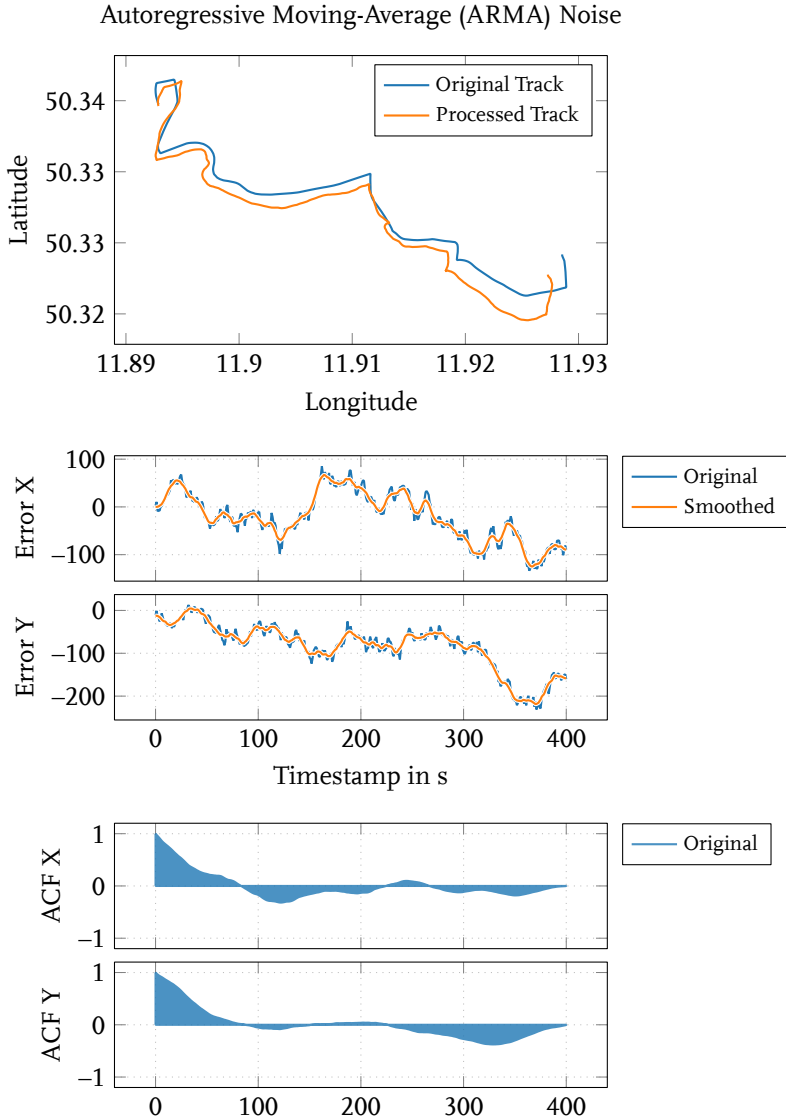


Figure 3.15: *Artificial GNSS ARMA noise* – The autoregressive moving-average (ARMA) process noise is autocorrelated, but diverges from the zero-mean over time. Abbreviation: Autocorrelation function (ACF)

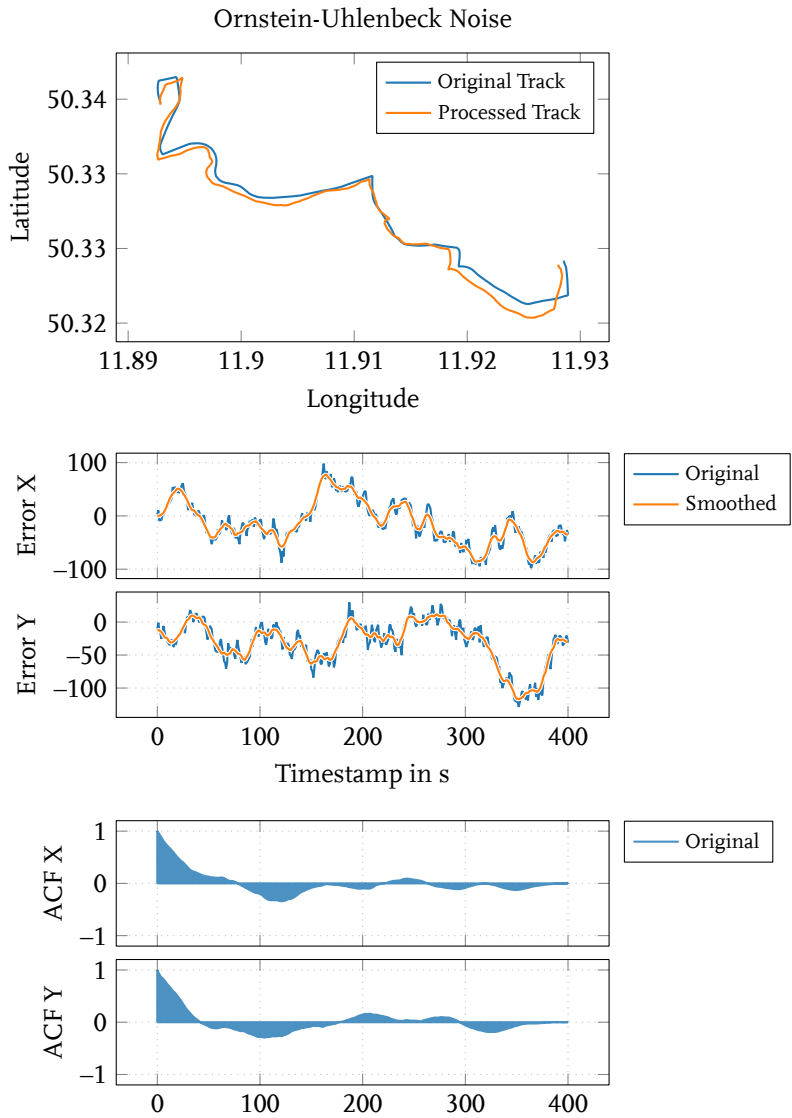


Figure 3.16: *Artificial GNSS Ornstein-Uhlenbeck process noise* – The Ornstein-Uhlenbeck process noise is autocorrelated and mean-reverting, resulting in the most realistic noise over time. Abbreviation: Autocorrelation function (ACF)

process, it slowly reverts toward the zero-mean over time. Additionally, the moving-average smoothing further reduces the jittering of the underlying Gaussian process.

3.2.3 Noise Characteristics Validation

For evaluating the artificially generated GNSS noise, we conducted two experiments using real-world GNSS data from GPS records. In the first experiment, we compared the GNSS noise distributions of the artificially generated data with those of real GNSS data from a floating car data set. For this experiment, we learned the error and outlier distributions from the real floating car data data using hyperparameter optimization. However, since we lack ground truth for the floating car data, we cannot determine the actual error and outlier distributions. As a workaround, we first map match the floating car data with our Map Matching 2 solution, and export the candidate distances (i.e., ‘target distances’), which are the distances between the track points and the road positions to which the track points were mapped. The hypothesis is that, by matching the artificially generated floating car data in the same way and exporting the same candidate distance data, we can hypertune the error and outlier parameters for the artificial data until the ‘target distances’ closely overlap. We used the Python library Optuna (Akiba et al. 2019) until the ‘hypertuning distances’ (i.e., the distances between the artificially generated track points and mapped road candidate positions) were similar to the real ‘target distances’. The outliers were analyzed and extracted using Tukey’s fences with $k = 3$ on the distances so that they could be learned separately from the basic noise error distribution. For the time delta, we used 5 s to reflect the median time delta of 5 s from the real floating car data data.

In [Figure 3.17](#), we can see that the error distance distributions are similar to a Rayleigh distribution and to each other after hypertuning. This demonstrates that our artificial noise generation process can learn the parameters from real GNSS data to some extent, and can generate similar artificial noise. The real floating car data data has $\approx 3.34\%$ outliers, while

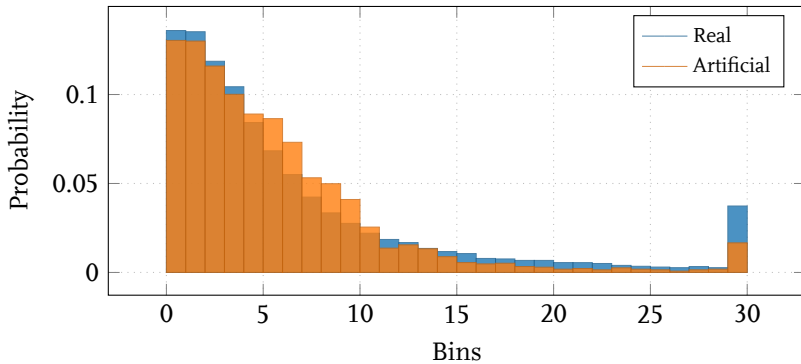


Figure 3.17: *Real and artificial GNSS distance distribution* – Comparison of the probabilities of the occurrence of a distance between the track points and matched road positions for real floating car data and a hypertuned artificial GNSS noise distribution. Both GNSS distance distributions follow a Rayleigh distribution. All distances ≥ 30 m are aggregated in the far-right bin.

the artificial floating car data after hypertuning has $\approx 3.38\%$ generated outliers.

In the second experiment, we personally recorded GPS points using a smartphone in a static position at a window in a building for about one hour. We anonymized the data by translating it to the $(0, 0)$ coordinate, and then we hypertuned the artificial process on a simulated track created from the centroid of the recorded GPS track, as the ‘assumed real position’. In this experiment, we learned the error, the Ornstein-Uhlenbeck reversion speed, the moving-average smoothing, and the random seed parameters. We chose these parameters because the real GPS points in this experiment have an unknown error distribution and a very long runtime due to being static, making a finer autocorrelation and smoothing factor tuning necessary. Additionally, we explored 100 distinct random seed parameters to identify one that more accurately represents the actual curve of the recorded track than other seeds. In practice, the seed has no impact on the distributions on average over many random values. However, for comparing a specific track shape, it is helpful to select a favorable seed that generates the noise in a similar direction.

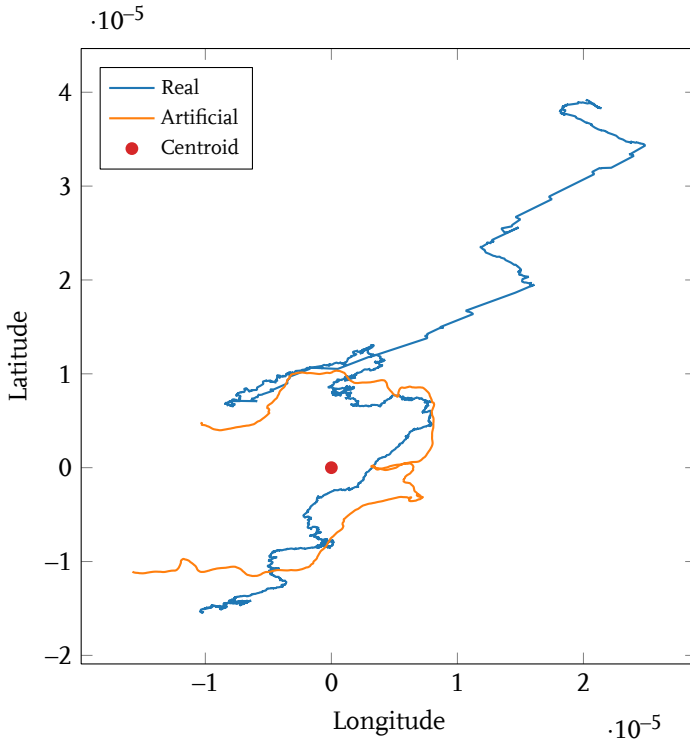


Figure 3.18: *Real and artificial GNSS static position comparison* – A self-recorded static position GNSS track is compared to an artificially generated GNSS track. Both tracks start in the north and end in the south. The artificial GNSS noise generation is able to generate noise with a comparable spatial extent and movement pattern.

In [Figure 3.18](#) we demonstrate that it is possible, to some extent, to generate an artificial track with a shape that resembles that of an original real GPS record within comparable bounds. While the beginning is not represented as well as the rest, as can be seen in [Figure 3.19](#), the dimensions and the autocorrelations, as seen in [Figure 3.20](#), especially for the latitudes, show that our process can generate fairly authentic GNSS noise, even over longer durations.

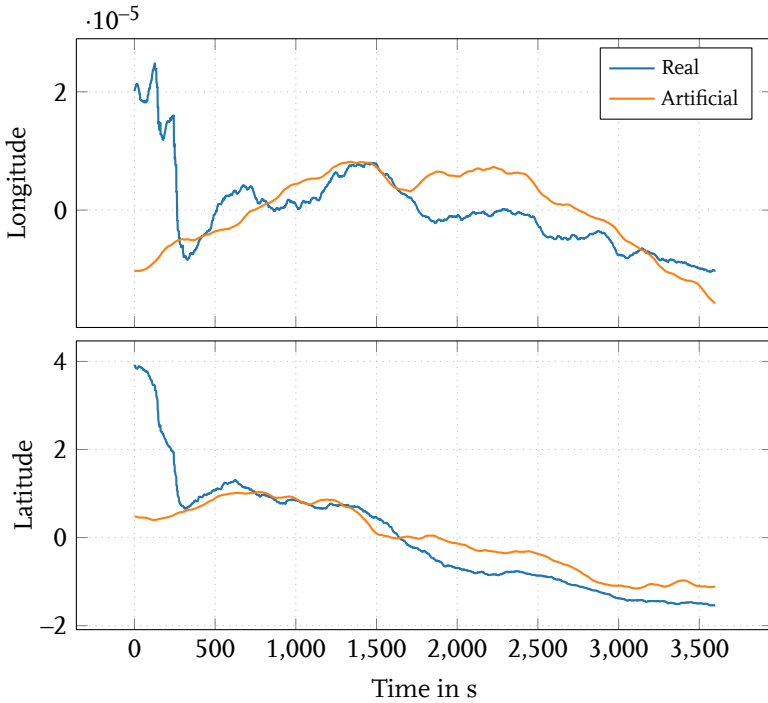


Figure 3.19: *Real and artificial GNSS static position coordinates* – The longitude and latitude axes of [Figure 3.18](#) in direct comparison. The movement pattern of artificial GNSS noise is comparable to that of real GNSS noise over time.

We can see that our artificial noise generation has some limitations. For example, real GNSS noise contains a very fine jittering that we cannot yet accurately represent. Without moving-average smoothing, the artificial jittering is too high, as can be seen by comparing the original noise from [Figure 3.16](#) with the original noise in [Figure 3.19](#). With moving-average smoothing, the jittering is smaller in direct comparison. With additional work, it should be possible to reapply fine noise after smoothing. However, we believe that the absence of fine jittering is not a significant issue for map matching purposes since the jittering is very minor compared to the actual distance error, which plays the major role in GNSS noise.

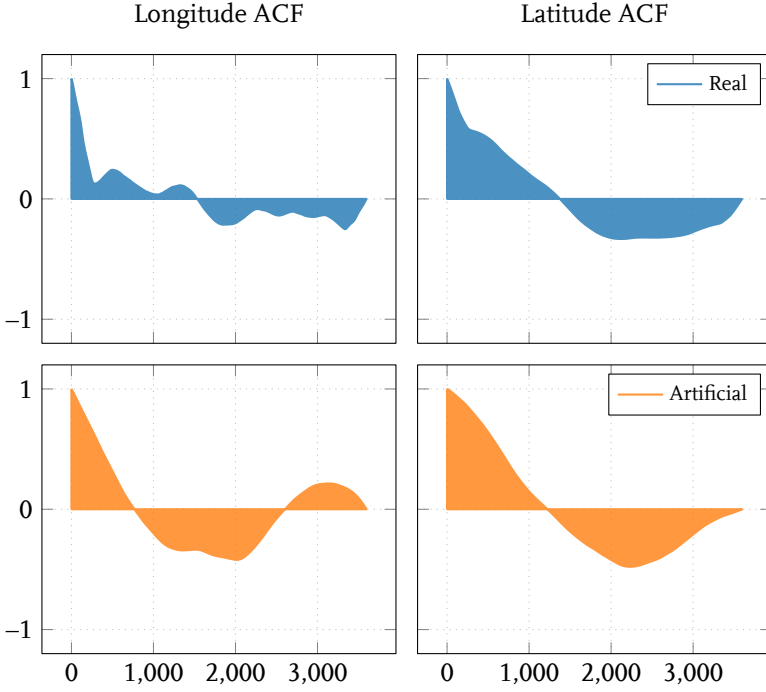


Figure 3.20: *Real and artificial GNSS autocorrelation function (ACF) plots* – For both longitude and latitude from [Figure 3.19](#), the autocorrelation function plots are compared. The autocorrelation of both axes is similar over time. In this example, the latitudes are more similar than the longitudes.

Although the artificially generated GNSS noise is only a model of real GNSS noise, we hypothesize that it is sufficient for evaluating map matching. First, our artificial GNSS noise generation process can generate all relevant noise characteristics for map matching use cases, such as autocorrelated noise, large outliers, and missing data points (as analyzed in [section 2.1](#)). For this reason, our artificial noise generation process is effective for evaluating such data challenges in map matching. Second, map matching metrics are very simplified in contrast to real GNSS error dynamics. The metrics in map matching can also contain terms such as direction changes of the computed road network routes, as shown in [Figure 3.12](#), which are not even present in the GNSS tracks. Thus, the artificial GNSS error does not

3 Method

need to be perfectly accurate, as map matching algorithms and metrics are not designed to accurately resolve GNSS errors, but rather to find a most fitting solution to recorded GNSS tracks with their respective noise characteristics.

4 Implementation

In this chapter, we will describe some of the essential implementation aspects and development goals of our reference implementation Map Matching 2, which implements our novel map matching approach. We will not go into technical details, as the documentation and open source program code provide a clearer picture of these. However, to better understand how and why our software behaves in the benchmarks the way it does, an explanation and evaluation of several of our software design principles is helpful. Furthermore, we will outline which parts of our reference implementation are crucial to our novel approach, which parts can be transferred to other implementations, and which parts are essential for the approach to work effectively.

4.1 Development Goals

The goal of our reference implementation is to provide a working, generally available software solution that allows applying our novel map matching approach to arbitrary tracks with reliable performance. Many scientific works do not provide an implementation at all, which we believe hinders continued research and public availability. To make our research accessible to future researchers and the public, we are providing our reference implementation Map Matching 2 as open source software under AGPL v3.0² at GitHub¹⁹. It offers an extensive interface and supports various input data and matching configurations. This enables future researchers to fine-tune and benchmark different settings, and it allows the public to configure the software according to their needs and data.

At the time of writing this thesis, Map Matching 2 is in the third iteration of its implementation. The first iteration, a Python prototype, began its development in June 2019. It is still available as a proof of concept under AGPL v3.0², also at GitHub²³. In the prototype, we initially focused on implementing our MDP concept and trying out various reinforcement learning algorithms. Technically, our concept worked, but we encountered two major issues. First, there are two disjoint ways of implementing candidate search on a road network in combination with routing. Both have advantages and disadvantages, but one is more suited than the other for map matching. We will discuss these findings next in [subsection 4.1.1](#). Second, there was a drastic limitation in runtime performance when using Python as the programming language, which made implementing candidate adoption practically impossible. This is the main reason why we switched to C++, which we will discuss in [subsection 4.1.2](#). As such, the initial C++ implementation is the second iteration. However, we encountered some further limitations in our second iteration which made it practically impossible to match on very large road networks. This is also an issue in some of the other existing open source map matching solutions. We will discuss these findings in [subsection 4.1.3](#). These limitations resulted in our third and current iteration of our reference implementation Map Matching 2.

4.1.1 Road Network Design

When performing a candidate search, we recall that each candidate corresponds to a point along a road edge. We identified two methods for computing this point in combination with subsequent routing between the candidates. The first method appears simpler because it enables precise routing between any candidates without modifying the default routing algorithms. However, it alters the road network graph, which creates additional issues. The second method does not alter the road network graph. However, it requires additional algorithms and data structures around the core routing algorithms to address the fact that routing algorithms can only route between nodes of a graph by default, not between points along edges.

²³<https://github.com/addy90/map-matching>

In our Python prototype, we used the first, simpler method, and in our C++ reference implementation, we used the second, more complex method.

The first method for computing candidates with respect to eventual routing between them is to split the graph edge at the closest point, introducing a new graph node if the point is not already a node. The advantage of this approach is that standard routing algorithms can be executed on the altered graph between the candidates without additional changes. Such standard routing algorithms include Dijkstra's shortest path algorithm and the A-star algorithm. By introducing new nodes at the positions of the candidates along the edges, these algorithms can operate directly on the altered graph without further modification. Thus, concatenating an optimal sequence of routes directly results in the match. However, the major drawback of this approach is that it modifies the graph to be specific for a single track. These changes need to be discarded for the next track, otherwise the number of graph nodes increases dramatically with each matched track, making routing increasingly slow. One idea is to make copies of the graph, but this is very expensive for large road networks. Another idea is to create a view of the graph that only includes the changes. However, this is more difficult to implement because the routing algorithms needs to transparently switch between the altered nodes and edges of the view and the original ones in the graph when no changes have been made in those locations. All these solutions have one common issue in that introducing nodes to a graph slows down routing because the routing time complexity depends on the number of nodes and edges. Since routing can be one of the slowest aspects in map matching, introducing additional nodes and edges contradicts our computational performance goals.

For these reasons, we implemented the second, more complex method. We leave the graph unmodified, compute the candidates as new points, and extract the incoming edge part from the current candidate to the source node, as well as the outgoing edge part from the target node to the next candidate, storing these as extra lines with the candidates. With this custom data type, the routing algorithms can compute the shortest path on the existing, unaltered graph between the target node of the source candidate and the source node of the target candidate, and the respective outgoing and incoming lines are added to the route, see [Figure 4.1](#). This method allows

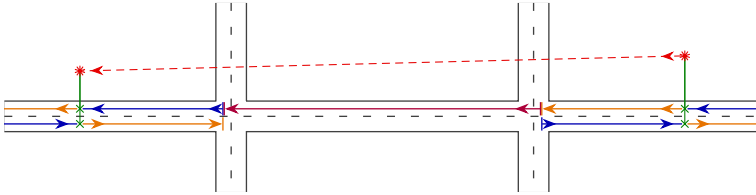


Figure 4.1: *Sub-partial candidate lines* – Blue incoming and orange outgoing sub-partial lines from a candidate to the outgoing junction (i.e., next graph node) and from the incoming junction (i.e., previous graph node) to the same candidate, for all edge directions, for all candidates. These sub-partial lines are concatenated, as required, to the purple shortest path route between the junctions, which is computed in the direction of the track.

the graph to be reused for every track without changes, even in parallel. Only the candidate data specific to each track is computed in addition to the shared graph. However, concatenating partial routes (i.e., the outgoing line, the graph route, and the incoming line) into a combined route can lead to many unnecessary data copy operations. Therefore, we use a custom route data type that allows to concatenate references instead of copying the data. Nevertheless, routes between candidates within the same edge, where the outgoing and incoming lines overlap, need to be addressed specifically to avoid loops between the target node of the source candidate and the source node of the target candidate (i.e., the same source and target nodes of the same edge on which both candidates lie in this case). Obviously, the second method is more complex in practice, but with the appropriate customizations, high routing performance can be achieved.

4.1.2 Programming Languages

One of the main goals of our reference implementation is high runtime performance. This is necessary for our candidate adoption feature to work in practice because candidate adoption dramatically increases the number of computed routes, evaluations according to our metrics, and states in our MDP. Although Python is a popular programming language in science, we encountered several computational performance issues that prevented

us from implementing candidate adoption in the intended way. Since the goal of our reference implementation is a high runtime performance, we established the following requirements for selecting a proper programming language:

- The language should be compiled directly into machine code to avoid the runtime impact of interpretation or just-in-time compilation.
- The language should explicitly allow working with references to data objects to allow for avoiding unnecessary copying of data.
- The language should allow for explicit memory management to avoid the often unpredictable runtime impact of garbage collection.
- The language should allow for explicit runtime checks of data and type integrity, as well as memory boundary checks, to allow for avoiding unnecessary checks when accessing data in defined, safe situations.
- The language should have a rich third-party library support so that we can focus on implementing the core algorithms of our approach.

In conclusion, C++ fulfills all of these requirements. It is a compiled language in which we can explicitly work with pointer arithmetic and references to avoid unnecessary data copying. We can also manage the memory as needed and disable runtime checks, such as those for array out-of-bounds access, when they are unnecessary. And, C++ has an established standard library and many additional third-party libraries. These features allow for focusing on the core algorithms, while also allowing for maximization of the computational performance of our implementation.

The runtime performance issue of the interpreted Python language compared to the compiled C++ language is also well described in Pereira et al. (2017), which has already been cited over 400 times. In their study, the authors compared 27 different programming languages regarding their runtime performance, memory performance, and energy efficiency on various algorithms. Python ranked second to last in runtime performance, being ≈ 71.90 times slower than the programming language C. In contrast, the modern programming language C++ is said to be only ≈ 1.56 times slower than C, which is somewhat surprising given that C++ should be equally fast with similar code. Upon further examination of the study, we

discovered that in the complete result set on the websites of the authors²⁴, the ‘regex-redux’ test is the only one in which C++ is significantly slower than C by a factor of ≈ 7 . Using the provided code, we discovered that a different regular expression (regex) implementation was used in the C++ code, so the results are not really comparable. This is the reason why C++ allegedly performs so poorly compared to C. In other words, disregarding the ‘regex-redux’ test, C++ is among the fastest existing programming languages and is generally not slower than C when similar algorithms are used, as shown by all other benchmarks in the result set.

For all these reasons, we decided to reimplement our approach from scratch in September 2020, which became our reference open source software implementation Map Matching 2¹⁹. Using C++ enabled us to implement the necessary optimizations to make our routing fast enough to lay the groundwork for our candidate adoption feature. In our opinion, candidate adoption would not have worked well in Python, especially due to the lack of pointer functionality in Python, which candidate adoption utilizes to reuse the incoming and outgoing sub-partial lines in order to share them among candidates with the same road positions.

By switching from Python to a C++ solution, we gained access to new software libraries for representing geometric, network, and graph algorithms. The Boost²⁵ libraries, a stable and popular collection of C++ libraries, provide a ‘Geometry’ and ‘Graph’ library, among others, and are frequently used in our implementation. For working with OpenStreetMap data, we rely on the libosmium²⁶ library. Other software dependencies are documented in our open source software code repository¹⁹.

The first major version ‘0.1’, which also marked our initial open source software publication, was evaluated in a small test case in Wöltche (2021). With many minor improvements, version ‘0.3’ was then evaluated in a large-scale benchmark against various third-party solutions in Wöltche (2023).

²⁴<https://sites.google.com/view/energy-efficiency-languages>

²⁵<https://www.boost.org/>

²⁶<https://osmcode.org/libosmium/>

4.1.3 Spatial Scaling

One of the requirements for providing a generally accessible open source software map matching solution is to allow for importing and matching on large road networks. The use cases requiring such spatial scaling are online map matching and providing offline map matching as an online web-service. In both cases, the spatial location of a track to be matched is unknown upon starting the software. As such, very large road networks need to be available before the matching starts, e.g., whole countries, such as Germany, when providing the service for German road networks, or even larger road networks, such as whole continents. Nevertheless, also regular offline map matching benefits from spatial scaling optimizations. For example, with our older version ‘0.3’, we needed > 100 GiB of main memory and around > 2 min of startup time to import and match on whole Germany in our test cases, which makes the usage impractical. All other map matching specific open source software implementations that we evaluate in this work also have this limitation, e.g., Barefoot and Fast Map Matching. Barefoot is designed to work only with regional extracts because it also reads the entire road network graph into main memory during startup. Fast Map Matching is designed to work only with singular cities or other small road networks, since it precomputes all routes within specific bounds and reads that data into main memory during startup, along with the road network. The ST-Matching mode of Fast Map Matching omits the precomputation and works with slightly larger road networks, but still reads the entire road network into main memory during startup. Only the general routing engine implementations with map matching on top do not have this limitation, e.g., GraphHopper, OSRM, and Valhalla. These implementations store the road network in files on disk and only read the necessary data into main memory to fulfill the current request. As such, they can work with entire countries and even larger networks. Still, they are not particularly designed for map matching, which causes further issues down the line, as we will describe and evaluate later in [chapter 5](#). Thus, although supporting large road networks is a technical issue, we see it limiting the versatility of many implementations, especially those tailored for map matching.

To address this issue in our map matching approach, we reworked our Map Matching 2 software to support memory mapping, a technology for swapping main memory in and out of disk-based files. This technology has the advantage of transparently representing data objects ‘as if’ they were in main memory, even though they are located in a file on disk. The operating system contains algorithms for automatically loading only the necessary parts of data that is accessed and discarding parts that are no longer needed, depending on available system memory. Unfortunately, the Boost ‘Graph’ library is designed in a way that renders it unusable with memory mapping as provided by the Boost ‘Interprocess’ library, because Boost ‘Graph’ does not support so-called ‘stateful allocators’. For this reason, we implemented our own graph in the current version of Map Matching 2 that supports ‘stateful allocators’, thus removing the Boost ‘Graph’ dependency. We could not find another C++ graph library that natively supports memory mapping or ‘stateful allocators’. Furthermore, the third-party routing engines OSRM and Valhalla, which are also implemented in C++, also use their own software-specific graph implementations. It appears that the lack of graph implementations compatible with memory mapping in C++ has persisted for some time.

For map matching, we need a specific set of graph features, such as directed graphs for routing. In our current Map Matching 2 version ‘1.0’, which is also evaluated in Wöltche (2025), we implemented the necessary features for graphs with memory mapping support, and we optimized the employed shortest path routing algorithms for map matching. As a result, our Map Matching 2 software is the first map matching specific open source software that supports very large road networks. As such, it provides the basis for online map matching and online-service offline map matching when the track location is unknown beforehand. Unlike the previous version ‘0.3’, since ‘1.0’, the road network graph does not need to be loaded into main memory during startup anymore. Instead, the graph is opened (i.e., referenced or ‘mapped’ into main memory) and its contents are transparently loaded and unloaded by the operating system on demand. This improves the speed of matching individual tracks, as only the necessary road network data is loaded.

Implementation Benchmarks Setup

A note on all benchmark tables that follow from now on. Unless otherwise stated, all benchmarks in this chapter are run on the region of ‘Upper Franconia, Bavaria, Germany’, outlined by the **green** line in the right panel of **Figure 4.2**. The tracks were taken from the Hof data set, which is described in **section 5.4**. All of the tracks are located in the city of ‘Hof, Bavaria, Germany’, outlined by the **red** line in the same figure. However, rather than using the city-level road network provided by the Hof data set, the larger region of ‘Upper Franconia’ was used to provide more distinct results compared to a very small region. ‘Upper Franconia’ is located in ‘Bavaria, Germany’, as outlined by the **blue** line in the left panel of the figure. The metadata for all regions is given in **Table 4.1**.

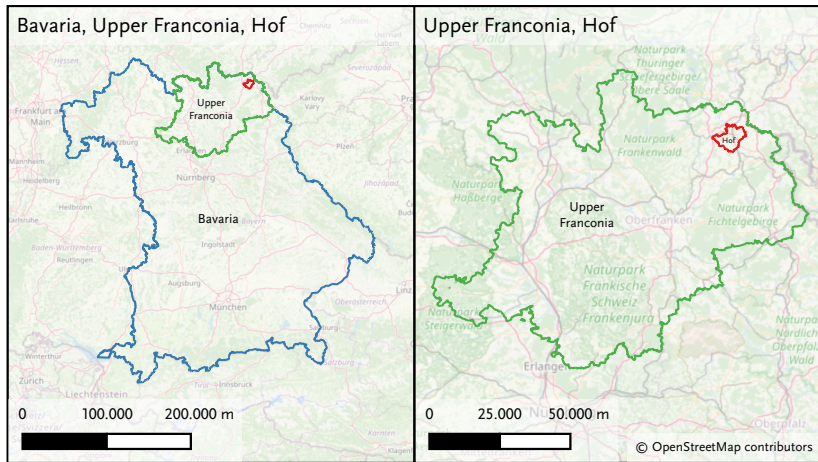


Figure 4.2: *Map administrative boundaries* – The left panel shows the regions of ‘Bavaria, Germany’ with **blue** outline, ‘Upper Franconia, Bavaria, Germany’ with **green** outline, and ‘Hof, Bavaria, Germany’ with **red** outline. The right panel shows a zoom-in of the ‘Upper Franconia’ region. The metadata for the road networks are given in **Table 4.1**.

Regarding the benchmark result tables, the default mode of operation of our software, as used in this work, is always marked with a **sky-blue** background color. The times and the CPU resources are measured in

Table 4.1: *Road network metadata* – Overview of the map and road network metrics of the regions of ‘Bavaria, Germany’, ‘Upper Franconia, Bavaria, Germany’ (U. F.), and ‘Hof, Bavaria, Germany’, all after road network topology simplification (as illustrated in [Figure 4.8](#)). The length is the sum of all edge lengths, i.e., the length of the road network.

Region	Area	Nodes	Edges	Length
Bavaria	$\approx 70\,542\text{ km}^2$	2 694 565	6 529 118	$\approx 910\,475\text{ km}$
U. F.	$\approx 7231\text{ km}^2$	234 292	584 693	$\approx 98\,482\text{ km}$
Hof	$\approx 58\text{ km}^2$	4411	10 708	$\approx 1102\text{ km}$

seconds, and the main memory consumption is measured in Mebibytes (MiB). Most benchmarks were run with single-threading, while larger ones were run with multi-threading on up to 256 threads for saving time. The hardware that was used is the same as that described later in [chapter 5](#). The accuracy is always the ‘weighted mean correct fraction’ that our comparison algorithm in [subsection 3.1.4](#) calculates after [Equation 2.2](#).

Memory Mapping Benchmark

Table 4.2: *Memory mapping prepare benchmark* – Results for preparing and starting up the region of ‘Upper Franconia, Bavaria, Germany’, with and without memory mapping (MMAP). Time and CPU are in seconds, maximum total resident set size main memory (TMEM), anonymous process memory (PMEM), and file-mapped memory (FMEM) are in MiB.

Mode	MMAP	Time	CPU	TMEM	PMEM	FMEM
Prepare	Full	26.35	28.39	2338	208	2130
Prepare	Output	12.28	14.27	1450	1442	502
Prepare	None	13.80	15.66	1439	1431	10
Startup	With	0.07	0.16	18	10	8
Startup	Without	2.71	2.82	392	382	10
Matching	With	9.09	9.25	128	31	97
Matching	Without	11.52	11.66	416	403	13

In [Table 4.2](#) we compare memory mapping against no memory mapping (i.e., main memory only) in our Map Matching 2 software. In the ‘Prepare’ phase, the road network is imported as a graph, and the spatial indices are built. The ‘Startup’ phase represents loading the graph and spatial indices, and the ‘Matching’ phase represents the full map matching processing of the Hof data set, which is described in more detail later in [subsection 5.4.5](#). We distinguish between ‘Full’ and ‘Output’ memory mapping. In the ‘Full’ variant, all road network data is written to memory mapping files on disk during the import, so that only the most necessary data remains in main memory. This allows for importing road networks that would otherwise require more main memory than is available during import, but it takes more time due to the overhead of writing everything to and from disk. Therefore, the ‘Output’ mode, which is the default mode used in our tests in this work, only writes the final output to memory mapping files and uses the main memory for all intermediate computations. This mode is the fastest in comparison, even faster than memory-only preparation without memory mapping, because in the latter case, the final memory-only data has to be serialized into files to be loaded again for the matching phase. Consequently, the startup and matching without memory mapping take longer because the memory-only binary files have to be deserialized into main memory before the matching can begin. This also results in greater main memory consumption because the entire road network is loaded. With memory mapping, only the necessary data is lazily loaded from the files as needed. However, since the ‘Output’ mode requires more main memory during the preparation phase, the ‘Full’ mode is the software default for end-users when using the software outside the scope of this work to minimize the likelihood of encountering memory limitations.

Preparation Times Benchmark

As shown in [Table 4.3](#), the individual times of the preparation and matching phases reveal the major impact of memory mapping. With ‘Full’ memory mapping, the importing of the road network takes significantly longer due to the necessary intermediate transformation of the OpenStreetMap road network data carried out on disk. Additionally, the road network simpli-

Table 4.3: *Preparation times benchmark* for memory mapping (MMAP) and no memory mapping, i.e., in-memory, in seconds – The individual steps are ‘Importing’, ‘Simplifying’, ‘Compressing’, ‘Indexing’, ‘Serializing’, ‘Deserializing’, and ‘Matching’ with ‘Full MMAP’ as profiled in Figure 4.3, ‘Output MMAP’ without intermediate memory mapping files as profiled in Figure 4.4, and in-memory without memory mapping, i.e., ‘None MMAP’, as profiled in Figure 4.5 and Figure 4.7. Additionally, the raw ‘Matching’ times without the opening of the memory mapping files, as profiled in Figure 4.6, and the deserializing of the binary in-memory dump files, as profiled in Figure 4.7, are shown.

MMAP	Import	Simp.	Com.	Index	Seri.	Des.	Match
Full	11.168	7.204	3.210	3.449	0.000	0.000	9.464
Output	5.238	1.188	3.116	3.256	0.000	0.000	9.464
None	5.209	1.114	2.250	2.695	3.208	2.178	8.823

fication takes significantly longer because of the random access patterns that the simplification algorithm carries out on disk. The compression phase transforms the road network from its adjacency list format into the final compressed sparse-row format, as described in more detail later in subsection 4.2.1. With ‘Full’ memory mapping, the data needs to be read from the adjacency list disk file while the transformed data is written to the compressed sparse-row graph output file simultaneously. This explains why the ‘Output’ only memory mapping mode is faster when the input data is transferred from main memory. The same is true for writing the spatial indices. In the ‘Output’ only memory mapping mode, the final files are written during the last two phases, so no further serialization is needed, in contrast to the in-memory mode without memory mapping. As expected, the matching phase alone is faster without memory mapping because, after deserialization, all the data is already in main memory and does not need to be loaded lazily during matching. However, the deserialization takes a lot of time, which the memory mapping case does not require.

Memory Profiling Benchmark

In the presented memory profiling plots, the preparation and matching phases are compared with and without memory mapping. We compare

the total main memory (i.e., resident set size), the actually used process memory, and the file-mapped (i.e., memory mapping) memory over time. In [Figure 4.3](#) we show the ‘Full MMAP’ preparation, in [Figure 4.4](#) the ‘Output MMAP’ preparation without intermediate memory mapping files, and in [Figure 4.5](#) the ‘None MMAP’ in-memory preparation memory consumption over time. In [Figure 4.6](#) we show map matching with memory mapping, and in [Figure 4.7](#) we show in-memory map matching without memory mapping. As can be seen, there are trade-offs between memory mapping and in-memory map matching, but memory mapping enables matching on large road networks without requiring much available main memory by swapping the data to and from disk.

On a side note, since version ‘1.1’, we have improved our support for map matching large floating car data sets quickly, which is an essential advantage for mobility analysis based on recorded movements.

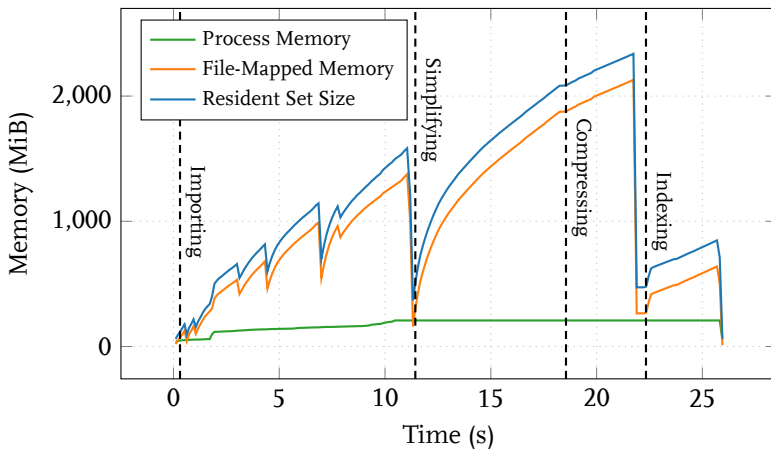


Figure 4.3: *Full memory mapping prepare memory profiling* – During importing, the memory mapping files need to be closed, grown, and reopened multiple times, because their final size is unknown beforehand. Simplifying on disk temporarily suspends deletion operations to increase performance by using more memory. After compressing the road network, unnecessary data is unloaded and deleted before indexing.

4 Implementation

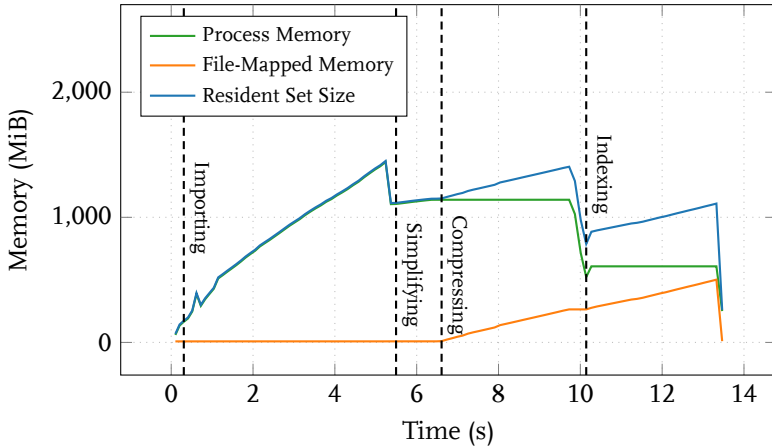


Figure 4.4: *Output-only memory mapping prepare memory profiling* – The importing and simplification are done completely in-memory and reuse previously allocated memory. Compressing and indexing write directly to memory mapping files from the current main memory. Unnecessary data is unloaded at the end of the compression phase.

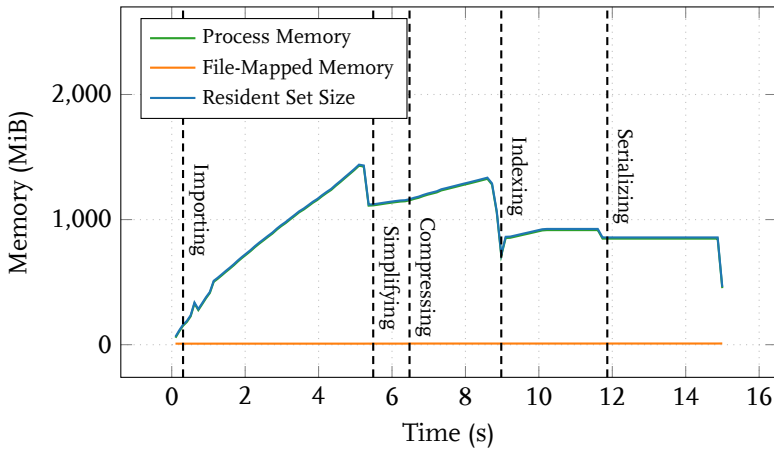


Figure 4.5: *In-memory prepare memory profiling without memory mapping* – No memory mapping files are created, but the main memory needs to be serialized, i.e., the compressed road network and the indices need to be written to disk files.

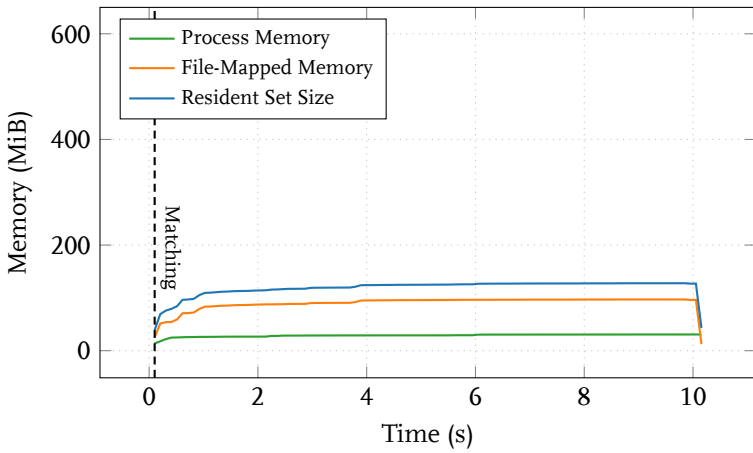


Figure 4.6: *Memory mapping matching memory profiling* – With memory mapping the matching can start immediately and read only the needed road network data.

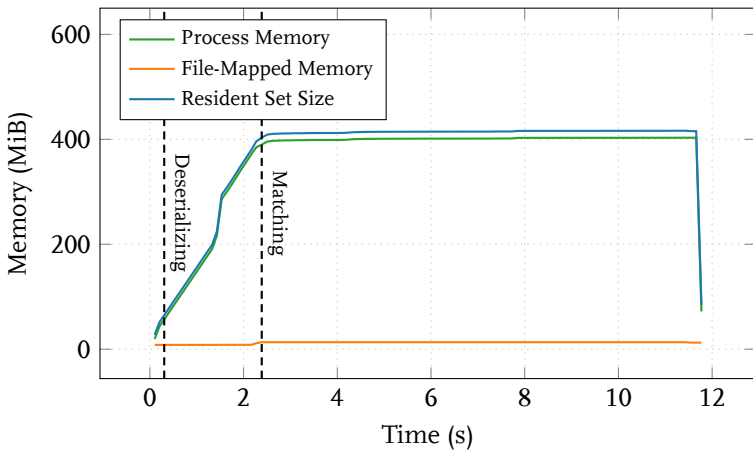


Figure 4.7: *In-memory matching memory profiling without memory mapping* – Without memory mapping, the road network first needs to be deserialized and read into main memory before the matching can start.

4.2 Essential Software Design Principles

In our Map Matching 2 reference implementation, we employ additional essential software design principles among those previously described in [subsection 4.1.1](#), concerning the candidate search, the graph routing, and the memory mapping graph data structures. We recall that the main focus of our work is to make the stochastic process more robust against outliers by providing it with more information. This ultimately leads to increased data processing computations that need to be addressed with specific algorithms and data structures. This ensures that the computational speed and resource usage remain within reasonable boundaries, making our novel concepts practically usable. In this section, we will focus on the key aspects of these algorithm and data structure customizations.

4.2.1 Graph Models and Algorithms

We already described our graph support for memory mapping, which makes large road networks available with low main memory consumption. Nevertheless, map matching can benefit from further optimizations regarding the road network graph and the corresponding algorithms. One of the main factors influencing the computational speed of map matching is routing. We recall that a route is computed for every pair of candidates from adjacent candidate sets, i.e., a cross-product. For example, for the candidates shown in [Figure 3.6](#), already 128 routes have to be computed. In practice, with higher k setting in the k -nearest neighbor search, larger search radii in the circular search, more GNSS track points, and especially when candidate adoption is used, we often have to compute millions of routes for a single track. As such, computational routing performance is crucial for efficient map matching.

During the development of our novel map matching solution, we identified several key aspects impacting computational routing performance. For computing an optimal route in a graph (i.e., a shortest path between two nodes), the graph needs to be traversed from each node over all outgoing

edges to their respective target nodes until all nodes are exhausted. Additionally, the weights of the edges need to be computed along all edges, and the actual route needs to be returned from the list of nodes found composing the shortest path. These are the essential aspects of shortest path routing to determine a route between two nodes.

Distance Limit Benchmark

The first rather obvious issue is that ‘all nodes and edges’ are traversed, which can be expensive for large road networks. This issue has already been addressed in Yang and Gidófalvi (2018) by specifying an ‘upper bound’ distance limit for Dijkstra’s algorithm. The key idea here is that the routes computed in map matching are generally very short, lying only within the close range of the currently evaluated track points. Therefore, routes that go to destinations far away from this location or routes that are much longer than the direct flight line connection between the two GNSS measurements are unlikely. By defining an ‘upper bound’, Dijkstra’s shortest path algorithm essentially cannot find optimal routes anymore, but since the computed routes are only an intermediate result before the stochastic optimization evaluates all of them against each other, inferring partially sub-optimal intermediate results for connections that are eventually discarded does not matter as long as optimal results for relevant connections still exist. For example, as long as for one track segment, one route is found for the candidate sets cross-product that turns out to be optimal towards a global optimum in the stochastic optimizer, the other non-optimal routes are irrelevant. An ‘upper bound’ dramatically limits the search range of Dijkstra’s algorithm, keeping it within the spatial bounds of the track. Furthermore, it allows using very large road networks without further impacting the routing speed. We hypothesize that, without an ‘upper bound’, using a much larger road network than the spatial track location would result in much slower routing speeds. As such, we also added an ‘upper bound’ in our implementation of Dijkstra’s shortest path algorithm to limit its search range.

Definition 4.1 (Distance Limit Factor): A distance limit factor is a factor that is based on the maximum linear distance between the current start candidate of the source candidate set and the farthest goal candidate from the target candidate set. For example, a distance limit factor of 5 is equal to five times the linear distance between the current start candidate and the farthest goal candidate.

Table 4.4: *Routing distance limit benchmarks* – For Dijkstra’s shortest path algorithm in ‘Upper Franconia, Bavaria, Germany’ run on a single thread. Enabled distance limit with a default distance limit factor of 5 against disabled distance limit. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Distance Limit	Time	CPU	MEM	Accuracy
Enabled	9.05	9.22	116	0.99651
Disabled	88.94	89.05	191	0.99656

As we can see in Table 4.4, having a distance limit allows for much faster map matching because the search radius for the shortest path routing is limited within the network. The road network is ‘Upper Franconia’ and the tracks are all located in the city of ‘Hof’, as shown in Figure 4.2. When the matching area is much smaller than the road network size, limiting the shortest path routing with a distance limit drastically improves the runtime performance. However, it can also slightly decrease the accuracy, especially when the distance limit is too tight, because, technically, Dijkstra’s shortest path algorithm cannot find optimal solutions under these conditions. We will evaluate this again in more detail shortly.

Shortest Path Memory Layout Benchmark

The design decision to use an ‘upper bound’ distance limit results in another small yet impactful optimization. Typically, Dijkstra’s shortest path algorithm internally uses several memory blocks, each the size of the number of all nodes. These memory blocks are used for, e.g., the predecessor, distance, and color maps. In Table 4.1 is the metadata for the road networks that are outlined in Figure 4.2, such as the number of nodes. When using

an ‘upper bound’, especially in combination with a very large road network, we hypothesize that it is advisable to use a data structure that allocates only the necessary memory instead of the default memory blocks, which reserve memory for all nodes of a graph. Although such a flexible data structure requires additional bookkeeping, it should still be faster overall than reserving the default whole memory blocks because the ‘upper bound’ limits the visited nodes to a tiny fraction of all graph nodes.

Table 4.5: *Shortest path memory benchmark* – Regions of ‘Hof’, ‘Upper Franconia’ (U. F.), and ‘Bavaria’, as shown in [Figure 4.2](#), run on a single thread. Evaluated are flat maps, and arrays (i.e., pre-reserved C++ vectors) with and without reusing. The ‘Bavaria’ region was retrospectively extracted with a different tool version and also may therefore contain a slight difference in the timestamp and data version of the road network. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Region	Memory	Time	CPU	MEM	Accuracy
Hof	Flat Map	8.53	8.68	42	0.99651
Hof	New Arrays	7.77	7.91	40	0.99651
Hof	Reused Arrays	7.65	7.80	42	0.99651
U. F.	Flat Map	9.05	9.22	116	0.99651
U. F.	New Arrays	16.05	16.21	120	0.99651
U. F.	Reused Arrays	8.25	8.41	122	0.99651
Bavaria	Flat Map	9.44	9.60	230	0.99654
Bavaria	New Arrays	208.35	208.40	268	0.99654
Bavaria	Reused Arrays	18.02	18.17	284	0.99654

As shown in [Table 4.5](#), which was run on the three different road networks from [Figure 4.2](#), using a flexible data structure with additional bookkeeping (i.e., flat maps in our case) is especially beneficial on very large road networks when an ‘upper bound’ distance limit is used. Using memory blocks (i.e., arrays or reserved vectors in C++), actually is faster when the road network is small because they require less additional bookkeeping. However, this changes as the road network grows because the allocation of the large memory blocks becomes the bottleneck. This suggests reusing the memory blocks instead of recreating them each time a shortest path is computed. In this case, additional bookkeeping is needed to clean the memory blocks before reuse, but since only a fraction of the memory is

written due to the limited distance, this is much faster than using new memory blocks in each routing call. As a side note, the ‘Bavaria, Germany’ road network was created retrospectively for this experiment with the same target timestamp, but was found to contain minor differences in practice. Thus, the resulting accuracy differs slightly on that road network.

Shortest Path Early Stopping Benchmark

Dijkstra’s shortest path algorithm is a one-to-many algorithm because it computes the shortest paths from one source node to all nodes in the graph. This can be limited by providing a set of target nodes, which stops the algorithm once the shortest paths to all given target nodes are found. However, this should still be combined with an ‘upper bound’ distance limit because, otherwise, if there is no route to a given target node, Dijkstra’s algorithm would have to perform an exhaustive search unless stopped by the ‘upper limit’.

Table 4.6: *Shortest path early stopping benchmark* – Aborting the shortest path search once all goals have been found, against an exhaustive search up to the default distance limit on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Early Stopping	Time	CPU	MEM	Accuracy
Enabled	9.05	9.22	116	0.99651
Disabled	21.98	22.13	130	0.99651

In [Table 4.6](#), we can see that, as expected, aborting the shortest path search after all goals have been found is faster and uses less memory than performing an exhaustive search up to the distance limit.

Shortest Path Dijkstra’s vs. A-star Algorithm Benchmark

Due to the one-to-many nature of Dijkstra’s algorithm, several lists of node sequences are returned for a shortest path search from a source node to a

given set of target nodes. The advantage of this approach over a one-to-one algorithm, such as the A-star shortest path algorithm, is that fewer shortest path routing calls are necessary for computing all routes for the candidate set cross-products. Caching the results from one call allows subsequent calls for routes from the same source node to another target node to be fulfilled from the cache rather than having to recompute the entire shortest path tree. For example, in [Figure 3.6](#), the A-star algorithm requires 128 calls, whereas Dijkstra’s algorithm requires only 30 calls to compute and cache all 128 routes. Especially in combination with candidate adoption, which drastically increases the density of the candidate sets, we hypothesize that using a one-to-many routing algorithm n times scales better than using a one-to-one algorithm $n \times m$ times.

Table 4.7: *Dijkstra and A-star algorithm benchmark* – All with default distance limit. Dijkstra and A-star with spatial reference system (SRS)-native geographic distance computations, and A-star with SRS-native heuristic distance computations against approximate heuristic distance computations based on Euclidean calculations on spherical computations. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Routing Algorithm	Time	CPU	MEM	Accuracy
Dijkstra	9.12	9.27	112	0.99651
A-star	8.72	8.89	108	0.99505
A-star Eucl.	8.41	8.57	104	0.99505

Contrary to our hypothesis, we can see in [Table 4.7](#) that Dijkstra’s algorithm, although called less often than the A-star algorithm, is in fact slower, especially when A-star uses fast Euclidean heuristic distance computations. However, the A-star algorithm yields significantly worse results. We hypothesize that the distance limit causes the A-star algorithm to stop searching before the local road network has been searched exhaustively enough to find an optimal route. The next experiment will provide further insight into this.

Table 4.8: *Dijkstra and A-star distance limit benchmark* – Default distance limit factor (DLF) is 5. The benchmarks were run with multi-threading on 256 threads. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Routing Alg.	DLF	Time	CPU	MEM	Accuracy
Dijkstra	1	0.51	10.99	372	0.98700
Dijkstra	2	0.53	11.78	394	0.99554
Dijkstra	5	0.53	12.28	416	0.99651
Dijkstra	10	0.61	12.78	468	0.99656
Dijkstra	20	1.32	16.85	536	0.99656
Dijkstra	50	5.00	42.25	807	0.99656
Dijkstra	100	7.85	78.76	1110	0.99656
Dijkstra	Off	13.20	118.39	1209	0.99656
A-star	1	0.50	10.34	506	0.56200
A-star	2	0.52	11.36	504	0.99425
A-star	5	0.53	12.52	510	0.99505
A-star	10	0.53	13.19	537	0.99512
A-star	20	0.82	14.07	622	0.99595
A-star	50	2.19	20.12	848	0.99605
A-star	100	3.91	37.66	1253	0.99656
A-star	Off	9.13	99.27	2076	0.99656
A-star Eucl.	1	0.43	9.32	312	0.56200
A-star Eucl.	2	0.48	11.20	352	0.99425
A-star Eucl.	5	0.50	11.13	384	0.99505
A-star Eucl.	10	0.52	11.46	416	0.99512
A-star Eucl.	20	0.66	12.53	458	0.99595
A-star Eucl.	50	1.80	18.73	610	0.99605
A-star Eucl.	100	3.34	32.15	837	0.99656
A-star Eucl.	Off	7.48	84.03	1198	0.99656

Shortest Path Distance Limit Factor Benchmark

For further evaluating our previous hypothesis, we can see in Table 4.8 that, in fact, the A-star algorithm performs worse with the default distance limit factor of 5 than Dijkstra's algorithm. Dijkstra's algorithm becomes stable

with a distance limit factor ≥ 10 , whereas A-star needs a distance limit factor of ≥ 100 for reaching the same accuracy. This means that the A-star algorithm requires significantly more resources for achieving the same level of accuracy as Dijkstra's algorithm in map matching with a distance limit. As such, Dijkstra's algorithm is actually more suited for map matching with an 'upper bound' distance limits than the A-star algorithm. Only without a distance limit does the A-star algorithm outperform Dijkstra's algorithm due to its more goal-directed design.

Route Referencing vs. Route Copying Benchmark

The final output of Dijkstra's shortest path algorithm (i.e., the several lists of node sequences), needs to be converted into the routes along the edges because the metrics of the routes are necessary for the stochastic optimization. When this conversion is performed naively, it results in the creation of copies of the edges along the node sequences into a new geometric line. Since we have to compute millions of routes, and since we have to concatenate the graph routes between the nodes with the partial lines from the candidate road positions to create the final routes between the candidates, the naive approach of copying the data is computationally and memory-wise very expensive. Therefore, we created a custom 'referencing' route data type that stores lists of references to existing lines while also being able to contain its own lines to which it self-references. For example, this is useful when a new line has to be extracted between two candidates on the same edge, as previously discussed. In the regular case, a route consists of a sequence of references to the already computed partial lines from the candidate position to the next graph node, the edges following the node sequence of the shortest path, and the partial line from the last graph node to the final candidate position. We hypothesize that this data type design significantly accelerates the generation of the final result of the shortest path algorithm.

As we can see in [Table 4.9](#), referencing existing road network graph edges and candidate sub-lines, as shown in [Figure 4.1](#), is much faster than copying the data, although it requires additional bookkeeping for maintaining the

Table 4.9: *Route referencing and copying benchmark* – Referencing existing road network edges and candidate sub-lines as shown in [Figure 4.1](#) in route objects against copying them on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Route Objects	Time	CPU	MEM	Accuracy
Referencing	9.05	9.22	116	0.99651
Copying	11.70	11.86	122	0.99651

references when concatenating multiple route objects into longer routes, which is necessary for the final match output.

Shortest Path Edge Weight Precomputation Benchmark

The impact of references over copies is further emphasized by the topic of weight computation in the shortest path algorithm. In our case, the edge weights are all positive because they are computed from the geometric length of the edges, i.e., the length is the weight. A naive approach would be to compute the lengths on demand while a route is computed. In a Cartesian coordinate system, these calculations are rather inexpensive, but in a geographic coordinate system, such as WGS-84, they are computed using iterative approximation algorithms, such as Andoyer’s formula (Andoyer 1932) or Vincenty’s formula (Vincenty 1975). Thus, repeated computations of edge lengths, which is performed when computing the candidate set cross-product routes, is computationally comparably expensive in geographic coordinate systems. Similarly complex are the calculations of the azimuths for the segments for computing the direction changes according to our metrics. Therefore, we hypothesize that precomputing the lengths and azimuths and storing them along the edges would mitigate the impact of these expensive calculations. With this approach, the weights for the shortest path routing and the additional metadata needed for the metrics in the stochastic optimization can simply be accessed rather than having to be calculated on demand.

Table 4.10: *Edge weight precomputation and live computation benchmark* – Using the precomputed edge weights against live computation in the shortest path algorithm on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Edge Weight	Time	CPU	MEM	Accuracy
Use Precomputation	9.05	9.22	116	0.99651
Live Computation	17.41	17.56	172	0.99651

In Table 4.10, our hypothesis is confirmed. Using precomputed edge weights is much faster than computing the weights during shortest path routing at runtime. In practice, the edge weights are precomputed during the ‘Compressing’ phase of our preparation process. As such, they are ready to be used as soon as the memory mapping files are opened for map matching.

Spatial Reference Systems Benchmark

Table 4.11: *Spatial reference system benchmark* – Comparison of Cartesian, spherical, and geographic spatial reference systems (SRSs) preparation and matching on a single thread. Cartesian uses Euclidean computations, spherical uses Haversine computations, and geographic uses Andoyer’s formula. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	SRS	Time	CPU	MEM	Accuracy
Prepare	Cartesian	9.91	12.04	1414	
Prepare	Spherical	10.98	12.97	1415	
Prepare	Geographic	12.13	14.20	1414	
Match	Cartesian	6.40	6.54	112	0.99490
Match	Spherical	7.07	7.22	118	0.99587
Match	Geographic	9.09	9.25	116	0.99651

The previous experiment also raises the question of how much geographic spatial reference system computations impact the speed and accuracy of the preparation and matching phases. We hypothesize that, even when using precomputation, the geographic computations are slower than Cartesian

computations. In [Table 4.11](#), we compare Cartesian, spherical, and geographic computations. As expected, the geographic computations are the slowest. However, they also produce the most accurate results. Spherical computations based on the Haversine formula are faster but less accurate. Cartesian computations based on Euclidean computations are the fastest, but the least accurate. Despite their complexity, we use geographic computations as the default because they produce the best results with only a moderate impact on runtime performance in our approach.

Road Network Topology Simplification Benchmark

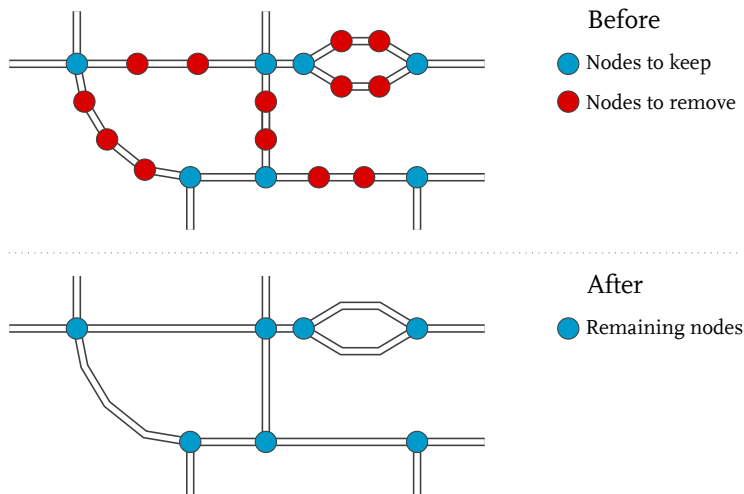


Figure 4.8: *Road network topology simplification process* – Red nodes represent graph nodes that are removed during simplification, while blue nodes represent intersections and endpoints that are preserved. This reduces the number of nodes while maintaining the topological structure of the network. Ultimately, only the blue nodes remain.

Definition 4.2 (Road Network Topology Simplification): A road network topology simplification algorithm removes graph nodes that are unnecessary for shortest path routing. This process is lossless in that the geometry

of the road network remains unchanged. This is accomplished by combining the geometric information of the edges adjacent to the node to be removed into a new edge connecting the remaining sibling nodes. As a result, the number of nodes and edges decreases while the geometric information remains.

This brings us to another aspect of the graph representation optimization. A road network topology simplification algorithm reduces the number of nodes and edges in the graph without altering its geometric representation. For example, [Figure 4.8](#) shows the nodes to remove, the nodes to keep, and the resulting simplified road network. Reducing the number of nodes and edges positively impacts the computational speed of any shortest path routing algorithm. However, it also prevents turning around within the roads for the shortest path routing algorithms, since the nodes within the roads have been removed. Nevertheless, we hypothesize that this could be advantageous in practice, as it could reduce the likelihood of unnecessary U-turns in noisy situations, especially since turning around within a road is probably less common than turning around at an intersection. Especially when combined with candidate adoption and our metrics that penalize direction changes, specifically for turnarounds and U-turns, disallowing turns within the edges by removing the nodes should be beneficial. Nevertheless, it is still technically possible to turn around at candidate road positions, if necessary, since concatenating the sub-lines of the candidates is performed after the shortest path routing algorithm is called, which allows turning around at every candidate.

Table 4.12: *Road network topology simplification benchmark* – Comparison of the preparation and map matching phases with and without road network topology simplification, as shown in [Figure 4.8](#) on a single thread on the region of ‘Upper Franconia’. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	Simplification	Time	CPU	MEM	Accuracy
Prepare	Enabled	12.13	14.20	1414	
Prepare	Disabled	17.43	19.50	2006	
Match	Enabled	9.09	9.25	116	0.99651
Match	Disabled	49.74	49.88	210	0.99329

We can see in [Table 4.12](#) that, as expected, disabling road network topology simplification slows down both the preparation and map matching phases. Although the simplification step is omitted during preparation, compressing the graph with all precomputations and building the spatial indices takes significantly longer because much more nodes and edges have to be processed. Additionally, matching on a denser graph with more (i.e., unnecessary) nodes is much slower than matching on a simplified graph. As expected, due to different routing behavior when the nodes are kept within the edges, the accuracy decreases slightly because U-turns within roads become feasible, which are less likely in the simplified graph version.

Table 4.13: *Road network topology simplification metadata* – Overview of the road network metadata before and after road network simplification, as shown in [Figure 4.8](#), for the regions of ‘Bavaria, Germany’, ‘Upper Franconia, Bavaria, Germany’ (U. F.), and ‘Hof, Bavaria, Germany’. The area and lengths, as shown in [Table 4.1](#), remain the same.

Region	Simplification	Nodes	Edges
Bavaria	Before	13 458 382	27 737 061
Bavaria	After	2 694 565	6 529 118
U. F.	Before	1 447 898	2 978 383
U. F.	After	234 292	584 693
Hof	Before	19 846	39 872
Hof	After	4411	10 708

The metadata in [Table 4.13](#) illustrates how the number of nodes and edges decreases for the evaluated regions. As can be seen, the numbers with simplification are much lower, which explains why the routing becomes correspondingly faster.

Graph Data Layout Benchmark

If road network topology simplification is desired, the road network graph needs to be represented as a mutable data structure, such as an adjacency

list, because the simplification algorithm has to modify the graph. With a bidirectional adjacency list, the road network topology simplification can be carried out in linear time without additional memory when the merging edges are replaced with the merged edge. However, for Dijkstra's shortest path routing, a directional, static (i.e., non-mutable) graph is sufficient and, in our hypothesis, more efficient to route on. A non-mutable directional graph data structure that meets these criteria is, for example, the compressed sparse-row (CSR), which has a smaller memory and data footprint than an adjacency list. Due to their smaller memory footprint and a higher data locality, which benefits memory caching in modern computers, compressed sparse-row graphs are preferred over adjacency lists when the graph no longer needs to change. Compressed sparse-row graphs must be built from an existing graph data structure in one pass because they cannot be altered, i.e., built iteratively. Thus, the graph must already exist in the correct representation (e.g., as adjacency list) and cannot be imported directly into a compressed sparse-row graph, if the nodes and edges are not finalized. This is the case with OpenStreetMap because the data is not organized as a graph data structure internally and requires a conversion step. For all these reasons, in our reference implementation, we first import a road network as a bidirectional adjacency list (i.e., the 'Importing' phase), and we don't yet compute the lengths and azimuths of the edges since they are not final during the import. Then, once the graph has finished importing and has optionally been simplified, we convert it to the compressed sparse-row graph format. During this step, we also precompute the lengths and azimuths of the edges and segments of the edges (i.e., the 'Compressing' phase). This optimizes the graph for fast shortest path routing with precomputed weights, and supports route-referencing of edges with precomputed metadata.

In [Table 4.14](#), we can see that the preparation phase with the additional conversion step into the compressed sparse-row graph format is slightly faster than using an adjacency list directly, although the latter does not require the conversion step. The reason for this is that storing the adjacency list in a memory mapping file takes much longer because it is much larger than the compressed sparse-row graph. Eventually, the adjacency list also performs slower in map matching and requires more memory to use than the compressed sparse-row graph, as expected.

Table 4.14: *Adjacency list and compressed sparse-row graph representation benchmark* – Comparison of the preparation and map matching phases on a single thread with a mutable adjacency list against a static compressed sparse-row (CSR). Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	Graph	Time	CPU	MEM	Accuracy
Prepare	CSR	12.05	14.09	1418	
Prepare	Adjacency List	12.18	14.12	1416	
Match	CSR	9.05	9.22	116	0.99651
Match	Adjacency List	9.45	9.60	196	0.99651

Spatial Index Benchmark

Finally, it is also notable that we create our spatial indices on the road network graph using an R-tree packing method (García R et al. 1998; Leutenegger et al. 1997) on the final compressed sparse-row graph, because this method has been shown²⁷ to create indices with faster query times than iteratively created R-trees. We hypothesize that this can be confirmed in our use case with map matching.

Table 4.15: *R-tree packing and R-star insertion benchmark* – Comparison of the preparation and map matching phases on a single thread with a bulk-packed R-tree against an R-star R-tree where the data is inserted iteratively. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	R-tree Loading	Time	CPU	MEM	Accuracy
Prepare	Bulk	12.05	14.09	1418	
Prepare	R-star	221.10	222.96	1417	
Match	Bulk	9.05	9.22	116	0.99651
Match	R-star	9.60	9.77	140	0.99651

Our evaluation in Table 4.15 confirms our theory. As expected, the R-star insertion method is much slower than the bulk packing loading algorithm, and the resulting R-tree is slower to query, slowing down the map matching phase.

²⁷https://www.boost.org/doc/libs/latest/libs/geometry/doc/html/geometry/spatial_indexes/introduction.html

The key takeaway from the latest results is that, when the road network graph and spatial indices have to be mutable, for example, when road network updates have to be applied regularly, the runtime performance and memory consumption of map matching will be worse than when using data structures containing optimizations based on immutability characteristics.

In conclusion, although the preparatory steps for the road network graph are complex, they significantly accelerate the computation of millions of routes, enabling our candidate adoption feature to perform efficiently and reliably in practice.

4.2.2 Candidate Search Algorithms

As previously explained, candidate search can be performed using a k -nearest neighbor search, a circular search, or a combination of both. In [Table 4.16](#), we measured the performance of different settings of k for k -nearest neighbor candidate search, different circular search radii for circular candidate search, and different k for our combined candidate search with the radius representing the distance limit for a candidate.

We use a default of $k = 16$ in our k -nearest neighbor and combined candidate search algorithm for two reasons. First, a regular junction in our graph has eight edges along the junction node, four incoming and four outgoing directed edges. With a value of 16, we can reference all of these edges and have additional space for referencing candidates from other close edges. Second, our candidate adoption feature enriches the candidate set with farther-away candidates from adjacent and nearby candidate sets after candidate search has been performed. Using a high default value for k grows the initial candidate sets too large, which would negate some of the benefits of the candidate adoption algorithm using a more intelligent approach to adapt farther-away candidates.

For the adaptive circular radius described in [subsection 3.1.2](#), which is used in our circular and combined candidate search algorithm, we use a default lower limit of either 200 m or the configured default radius, whichever is

lower. Thus, when using a larger radius > 200 m (e.g., 500 m), the dynamic radius adaptation significantly improves the runtime performance to stay within reasonable bounds by lowering the actual radius down to a maximum of ≥ 200 m, if the candidate situation permits. The lower default of 200 m is intentionally high to account for higher GNSS error and regular outliers. These occur especially within city boundaries when smartphones are used for measurements, as discussed in [subsection 2.1.2](#).

Furthermore, the upper radius limit for limiting the doubling of the search radius when no candidates were found is 10 km. This distance is intentionally high because its only purpose is to limit the radius doubling. This limit is rarely reached because radius doubling stops as soon as at least one candidate is found. If no candidates have been found within 10 km, we can safely assume that there are no reasonable candidates to map the point to. Such points are eventually skipped dynamically with our MDP design, as explained in [subsection 3.1.6](#).

We hypothesize that increasing the circular search radius, or k , will improve the accuracy of the results by enlarging the number of candidates. However, having more candidates is also expected to decrease performance. Therefore, we expect to find a balance between increased accuracy and decreased performance.

In [Table 4.16](#), we can see how each candidate search mode performs with different parameter settings. As expected, all modes have in common that the matching slows down as the parameter settings increase and more candidates are found. We can also see sweep spots where the improvement in accuracy is not proportional to the increase in computational resources. However, as the parameter settings increase further, the accuracy decreases again, which is unexpected at first. This outcome is explained by our stochastic outlier handling, which is based on our candidate adoption feature in combination with our metrics. With very large candidate search circles or k , the start and end of a track tend to be cut off by collapsing the points to later or earlier candidates. This occurs specifically when the start and end of the track contain large turns. In other words, in such situations, the start and end tend to be detected as outliers instead of being mapped as is, when too many possibilities are given to the stochastic optimizer.

Table 4.16: *Candidate search modes and radius benchmark* – Comparison of the k -nearest neighbor (k -nearest), adaptive circular (A. Circle), one regular circular without adaptive radius (*), and combined candidate search (CS) with multi-threading. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

CS	k	Radius	Time	CPU	MEM	Accuracy
k -nearest	1		0.27	3.62	106	0.78564
k -nearest	2		0.35	4.57	118	0.96255
k -nearest	4		0.34	5.16	138	0.98446
k -nearest	8		0.40	6.78	210	0.99309
k -nearest	16		0.52	11.67	432	0.99651
k -nearest	32		0.97	28.25	828	0.99732
k -nearest	64		2.23	82.02	1906	0.99737
k -nearest	128		6.03	270.99	4756	0.99675
k -nearest	256		25.27	1058.85	13361	0.99650
A. Circle		1	0.35	4.27	132	0.95895
A. Circle		2	0.36	4.35	118	0.96025
A. Circle		5	0.30	4.20	134	0.96761
A. Circle		10	0.33	4.58	142	0.98022
A. Circle		20	0.39	6.57	242	0.99061
A. Circle		50	0.70	16.99	608	0.99523
A. Circle		100	1.89	81.71	2034	0.99703
A. Circle		200	10.68	708.72	10258	0.99660
A. Circle		500	13.72	882.37	12373	0.99660
Circle *		500	604.41	24256.62	163975	0.98754
Combined	1	200	0.31	3.69	90	0.78564
Combined	2	200	0.32	4.43	116	0.96255
Combined	4	200	0.36	5.25	148	0.98446
Combined	8	200	0.39	6.98	216	0.99309
Combined	16	200	0.53	11.89	424	0.99651
Combined	32	200	0.96	29.73	840	0.99732
Combined	64	200	2.20	80.96	1890	0.99737
Combined	128	200	5.24	248.59	4506	0.99685
Combined	256	200	8.84	591.64	8915	0.99667

Consequently, the accuracy decreases, as some results end up shorter than the correct solution. We visually confirmed this in the data. Whether the outlier collapsing is too aggressive with larger candidate search settings is open to discussion. Nevertheless, it seems to be very efficient and accurate with the smaller, default candidate search settings.

Furthermore, we can see that combining k -nearest neighbor and circular candidate search has a higher performance in terms of runtime and accuracy than the corresponding scenarios because the circular distance limit reduces the number of k candidates, especially with larger k . For this reason, combined candidate search is the default mode in our Map Matching 2 application. We also evaluated the impact of disabling the adaptive circle (in the scenario marked with *). However, having way too many candidates dramatically impacts the performance in terms of computational speed, resource usage, and accuracy. Therefore, we can conclude that a good candidate search is crucial for high map matching performance.

4.2.3 Stochastic Process Performance

In general, MDPs are computationally slower than HMMs when the same number of states are traversed. There are multiple reasons for this. First, when building a HMM, all hidden states and observations have to be known beforehand. Then, cache-aware memory blocks can be used to represent these states and their respective observations and transition probabilities. The Viterbi algorithm also only requires one sweep through the model to compute an optimal path. Therefore, moving away from such a computationally optimized solution introduces additional challenges.

Definition 4.3 (Index-Mapping): Index mapping is a technique in which data is stored at memory array index positions for direct memory access. These indices are then used to reference the data. This improves runtime performance compared to hash lookup tables, which also have a constant access time but require a hash computation for each lookup.

In reinforcement learning, states are usually more complex and can consist of multiple attributes, as is the case in our model design, as illustrated in [Figure 3.11](#). Our states consist of up to three values, two track points, and one source candidate. This represents the current position of the agent and the knowledge of the next goal. States need to be recognized when they are revisited. Thus, the more complex the state model, the more computationally expensive the state-recognition becomes. In tabular Value Iteration and Q-learning, the state values or state-action values for a state need to be stored in a way that allows them to be accessed again when the same state is reached during the iterations in Value Iteration or the episodes in Q-learning. Traditionally, hashing algorithms are used for this state and state-action recognition. Although these algorithms have constant time complexity, they are much slower than direct memory access, which also has constant time complexity, because no further computations are required in that case. For this reason, we use a technique called ‘index-mapping’ to recognize states that have already been visited. Each state, action, and state-action pair is assigned a directly accessible memory index upon generation. This makes comparing and recognizing states, actions, and state-action pairs much faster than using complex object comparison and hashing methods. This technique also facilitates storing already computed rewards in a cache memory under the same index, allowing actions carried out more than once in a state to simply return the previously computed reward. These re-taken actions in states occur in both the iteration sweeps of Value Iteration and the episodes in Q-learning. Furthermore, we use the ‘in-place’ version of Value Iteration with beneficial preordering of the states by traversing them in the reverse order of the forward generation to reduce the number of sweeps needed for convergence. With these optimizations, our MDP is nearly as fast as the HMM and only slower by a few percent, as we will see in more detail in the benchmarks in [chapter 5](#).

In [Table 4.17](#) we evaluated the impact of disabled beneficial preordering and in [Table 4.18](#) the impact of disabled index-mapping on our MDP. We can see that disabling either of them reduces the runtime performance of the matching phase significantly. Generating the MDP states in an order that benefits iterative optimization and using index mapping instead of key-value store lookup both increase the runtime performance of solving the MDP, despite the additional bookkeeping required. Interestingly, index

Table 4.17: *MDP beneficial preordering benchmark* – Comparison of disabled beneficial preordering of the states of a Markov decision process (MDP), and the default, which is enabled, all on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	Time	CPU	MEM	Accuracy
Default	9.05	9.22	116	0.99651
No Beneficial-Preordering	10.19	10.34	116	0.99651

Table 4.18: *MDP index-mapping benchmark* – Comparison of disabled index-mapping of the states of a Markov decision process (MDP), and the default, which is enabled, all on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	Time	CPU	MEM	Accuracy
Default	9.05	9.22	116	0.99651
No Index-Mapping	10.18	10.34	122	0.99651

mapping with MDPs also improves memory consumption, despite the additional bookkeeping. This is because the main state-cache uses large keys with hash lookup tables as opposed to array lookup with index mapping, in which the key is already the position.

4.3 Memory Access Optimizations

As hinted at in previous sections, another performance optimization we use is employing different caching strategies throughout the matching process. Specifically, we use four different caching levels to address the most frequent computational repetitions.

Edge Data Precomputation Benchmark

First, as previously mentioned, we precompute and cache all edge lengths and azimuths, as well as all of their segment lengths and azimuths in the graph. This data is eventually reused by the partial candidate lines and the referencing routes, which minimizes the need for additional length and azimuth computations. In practice, therefore, only the total azimuth between the first and last points of a concatenated route, and the data for new partial segments being created at the candidate road position need to be recomputed. Any other segments and their data are reused. Additionally, when a track is imported and optionally simplified, it is enriched by precomputing all lengths and azimuths, similar to the edges. This ensures that also the track is fully prepared with all metadata during the matching process.

Table 4.19: *Edge precomputation benchmark* – Comparison of precomputing all edge lengths and azimuths in the graph and completely disabling the precomputation, all on a single thread. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	Precomputation	Time	CPU	MEM	Accuracy
Prepare	Edges	12.05	14.09	1418	
Prepare	None	10.41	12.49	1415	
Match	Edges	9.05	9.22	116	0.99651
Match	None	22.36	22.52	129	0.99651

As we can see in [Table 4.19](#), disabling the edge length and azimuth precomputation in the graph slightly decreases the preparation time, but dramatically reduces the matching runtime performance. This is because, while these computations are omitted during the preparation phase, they have to be carried out repeatedly without precomputation during the matching phase when multiple tracks traverse the same road edges. This benchmark differs from the weight precomputation benchmark in [Table 4.10](#) in that any usage of precomputed lengths and azimuths is disabled throughout all data models. For this reason, this benchmark produces worse runtime performance results than the earlier benchmark, in which only the usage of precomputed weights during routing was disabled.

Cache Layers Benchmark

Second, as previously explained, every state-action reward is cached so that repeated action calls in states, as is done in reinforcement learning algorithms, reuse the cached reward values instead of recomputing them.

Third, as previously explained, we use a cache for the shortest path routing results. This means that one call of Dijkstra's shortest path algorithm computes all routes from one source candidate to all next candidates and stores the results in the cache. This generates more data than the caller immediately needs, but it precomputes the data for subsequent calls. In Value Iteration, all precomputed routes are guaranteed to be called eventually. However, in Q-learning, not all precomputed routes may be called due to the random action selection. As long as Q-learning explores a large portion of the state-action space, this trade-off may be acceptable.

The fourth cache that we have not yet mentioned is an additional cache for geometric length and azimuth computations between any two points, i.e., for any segments. This cache is used for any length or azimuth computation that has not been precomputed before. This ensures that the expensive iterative approximation algorithms in geographic spatial reference systems are only performed once per matching. There are various situations in which repeated computations between any two points occur in practice. For example, this happens during candidate adoption when track points and candidates are shared and the distances need to be computed. It also happens when different routes have the same start and end points and the azimuths need to be computed.

As shown in [Table 4.20](#) we can see the impact of disabling individual or all of the cache layers. As expected, disabling the additional distance and azimuth cache only slightly impacts the performance because most lengths and azimuths are already precomputed in the graph, except for those previously mentioned. Disabling the MDP reward cache has a much higher impact because the iterations in the optimization algorithm have to recompute the rewards for already visited states each time. However, disabling the routing cache has the highest individual impact, because

Table 4.20: *Cache layers benchmark* – This shows the difference between having all cache levels enabled, disabling them individually, and disabling them altogether, i.e., having no cache enabled. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB. Abbreviation: Markov decision process (MDP)

Caches	Time	CPU	MEM	Accuracy
All	9.05	9.22	116	0.99651
No Distance & Azimuth	9.49	9.65	114	0.99651
No MDP Rewards	17.94	18.10	110	0.99651
No Routing	80.40	80.54	110	0.99648
None	367.15	367.20	106	0.99648

Dijkstra’s shortest path algorithm has to be called $n \times m$ times instead of n times, since the shortest paths to other goals needed later are not stored for future querying. It has to be noted that disabling one cache layer can still lead to other cache layers intercepting the queries. For example, when the routing cache is disabled but the MDP cache is still enabled, the cached rewards prevent further routing calls down the line.

Interestingly, disabling the routing cache slightly decreases the accuracy. This is due to the ‘upper bound’ distance limit. Although the MDP requests a route for a single source-and-destination node-pair, routes for the other destination nodes from the target candidate set are also computed and populated into the routing cache. When this is done for a source and target candidate set covering a larger area, the routing cache can contain routes that would not be computed for a source and target candidate set covering a smaller area in the same region, because the routing distance limit depends on the node distances in the candidate pairs. Thus, the routing cache can sometimes already contain routes for node-pairs from earlier routing calls that would not be computed in subsequent calls. For this reason, disabling the routing cache with an ‘upper bound’ distance limit can lead to slightly worse results.

Therefore, disabling all cache layers similarly reduces the accuracy but also dramatically worsens the runtime performance. While the main memory consumption is lower without caches, the computational impact is not worth the gained memory. In conclusion, an appropriate cache layer de-

sign that mitigates repeated computations can significantly improve map matching performance.

All of our caches, except for the precomputed edge data that is baked into the graph, are thread-local. This means that the state-action cache, the routing cache, and the geometric length and azimuth caches are only used once per track and are not shared among matching different tracks. There are two reasons for this. First, different tracks generally do not have the same shape, so the state-action cache is only valid for a specific MDP instance. The routing cache could be shared between different tracks, but this is only useful if the distinct tracks share the same geospatial area. Otherwise, the shared routing cache is useless and bloated with unusable data, which slows down the access. The same goes for the length and azimuth caches. Second, with our current software and road network design, we can match multiple tracks in parallel using multi-threading. Since our graph is static and unaltered, we can read from it with multiple threads in a parallel, lock-free manner. Each matching of a track has its own thread-local storage, which allows this solution to scale well with many CPU cores. However, sharing the routing or distance and azimuth caches between the tracks would introduce a synchronization bottleneck. This would also no longer allow discarding the caches to free memory after a match is completed because it would interfere with the data requirements of the parallel matches. Ultimately, this would lead to a memory leak that would require additional housekeeping, such as time-limited caching, which would negatively impact the performance again. For these reasons, the cache layers are thread-local to each matching of a track.

Multithreading and Memory Allocation Benchmark

One final notable performance improvement addresses global memory allocation. As previously explained, our reference implementation works with many small data fragments in various caches in combination with multi-threading. The challenge is that memory allocation is a thread-synchronized, blocking call to the operating system. Therefore, allocating many small memory fragments with many threads takes significantly more

time than allocating fewer large memory fragments with one thread. For this reason, we also use the public domain memory allocation manager `rpmalloc`²⁸ to transparently intercept memory allocations to the operating system. This way, larger memory blocks are prefetched by `rpmalloc` and provided thread-local and lock-free to the querying threads. The hypothesis is that the run-time performance improves at the expense of increased memory usage.

As shown in Table 4.21, our Map Matching 2 software scales well with the number of threads. We use a thread-pool concept with a buffered task-queue that is filled during importing. The queue sends the tasks to the thread-pool when threads finish their previous task and become available for the next one. As we can see, increasing the number of threads up to the number of available system threads decreases the time for the matching to finish. However, the CPU resources increase with more threads because further threads need additional bookkeeping, and because the CPU frequency decreases as more threads are queried, i.e., the usual turbo frequency boost only occurs with a single or very few threads. Additionally, we can see that the main memory consumption increases with the number of threads because each thread has its own thread-local storage. This storage adds up, resulting in increased total memory consumption. Increasing the number of threads beyond the number of available system threads is not useful with a thread-pool approach, as the operating system then has to schedule the worker threads, which end up competing over the available physical system threads. As a result, the CPU resources of the application decrease because the operating system requires them for the scheduling. In addition, the 2-socket simultaneous multithreading (SMT) CPU system that we used for our benchmarks causes performance interference between ≤ 128 and ≥ 256 threads due to CPU thread-pinning. For example, ≤ 128 threads can be distributed unevenly across both CPUs whereas ≥ 256 threads can use all physical CPU threads evenly.

Furthermore, as expected, the results also show that `rpmalloc`²⁸ significantly improves the computational speed by trading memory usage for reduced CPU resources. Interestingly, during preparation, memory usage is higher

²⁸<https://github.com/mjansson/rpmalloc>

Table 4.21: *Memory allocation and multi-threading benchmark* – This comparison demonstrates the impact of multiple threads and the memory allocator library `rpmalloc`²⁸ on the computational resources with different numbers of threads. The benchmark system has 128 cores distributed on two CPU sockets with 256 threads in total. The data set contains 263 tracks of various sizes. Time and CPU are in seconds, maximum resident set size main memory (MEM) is in MiB.

Mode	rpmalloc	Threads	Time	CPU	MEM	Accuracy
Prepare	Enabled	1	12.10	14.15	1422	
Prepare	Disabled	1	14.03	16.00	1697	
Match	Enabled	1	9.11	9.26	118	0.99651
Match	Enabled	2	4.60	9.22	130	0.99651
Match	Enabled	4	2.39	9.29	150	0.99651
Match	Enabled	8	1.29	9.34	184	0.99651
Match	Enabled	16	0.74	9.41	224	0.99651
Match	Enabled	32	0.50	9.58	298	0.99651
Match	Enabled	64	0.42	10.82	358	0.99651
Match	Enabled	128	0.44	13.40	392	0.99651
Match	Enabled	256	0.51	12.56	440	0.99651
Match	Enabled	512	0.62	11.98	500	0.99651
Match	Enabled	1024	1.02	12.40	542	0.99651
Match	Disabled	1	9.86	10.03	106	0.99651
Match	Disabled	2	5.01	10.04	112	0.99651
Match	Disabled	4	2.58	10.08	118	0.99651
Match	Disabled	8	1.38	10.15	128	0.99651
Match	Disabled	16	0.77	10.24	142	0.99651
Match	Disabled	32	0.47	10.38	162	0.99651
Match	Disabled	64	0.37	12.60	201	0.99651
Match	Disabled	128	0.37	17.98	246	0.99651
Match	Disabled	256	0.45	16.59	260	0.99651
Match	Disabled	512	0.52	13.86	277	0.99651
Match	Disabled	1024	0.66	14.77	306	0.99651

without `rpmalloc`. Since `rpmalloc` keeps track of different memory block sizes internally and reuses previously prefetched memory before returning it to the operating system, `rpmalloc` appears to provide better memory management for the allocation pattern used in the preparation phase than the

operating system itself can provide. During the matching phase, especially with many threads, `rpmalloc` allocates more memory while consistently improving computational speed. As expected, with many threads, the total elapsed time is slightly higher, since more memory has been allocated and needs to be returned to the operating system after matching is complete. However, the benchmark size of 263 tracks is comparably small, with fewer tracks than the number of system threads used with > 256 threads. With larger data sets, the difference is more apparent, as we empirically tested during development. In conclusion, specifically due to our candidate adoption process and MDP design, which lead to many small data fragments in memory, using a more sophisticated memory management allocator than the default one significantly improves the runtime performance of our map matching solution.

4.4 Code Organization and Transferability

Due to the complex, customized data structures and algorithms that we developed for our reference implementation Map Matching 2, some aspects of our novel approach may be more challenging to fully adapt to new or existing implementations. However, the code for the major innovations on top of the customizations and optimizations is not very complex. Here, we will provide tips on where to find these parts of the code.

We recap that the main contributions of our novel approach are the individual trajectory simplification algorithms, the candidate adoption feature, the MDP model, the new metrics with direction changes, and the comparison algorithm. While these features can be transferred to new implementations without the complex code foundation, there may be a significant performance impact, particularly with regard to the candidate adoption feature.

Our open source software reference implementation Map Matching 2¹⁹ is organized into two parts, a library containing the map matching algorithms and data structures, and an application interface that builds upon the

library. The code of the library is organized into different packages, e.g., geometry, graph, input-output, matching, comparison, and the models with optimization algorithms. The code within these packages is organized into different modules depending on the use cases.

In the current version of the software, that was recent to the date of writing this thesis, the essential algorithms for our novel approach can be found in the following modules of our Map Matching 2 reference implementation: The trajectory simplification algorithms are located in the ‘geometry/algorithm/simplify’ module for the customized Douglas-Peucker algorithm, and in the ‘geometry/algorithm/median_merge’ module for the ‘median-merge’ algorithm. The ‘duplicates-removal’ and ‘warp-detection’ algorithms are implemented in the ‘matching/algorithm/detect’ module. The comparison algorithm is defined in the ‘geometry/algorithm/compare’ module. The HMM and MDP, including the metrics as probabilities for HMMs and rewards for MDPs, are defined in the ‘environment’ package. The ‘environment/environments/hmm’ module is for the HMM with the Viterbi algorithm, which is defined in the ‘learning/algorithms/viterbi’ module. The MDP, in its current design for single tracks not transferring learned behavior to multiple tracks, is defined in the ‘environment/environments/single’ module, with the Value Iteration algorithm defined in the ‘learning/algorithms/value_iteration’ module, and the Q-learning algorithm defined in the ‘learning/algorithms/q_learning’ module. The candidate search algorithms, including the candidate adoption methods, are defined in the ‘matching/algorithm/candidate_search’ module. The algorithms for processing the shortest path routes with the sub-partial lines between the candidates and for extracting the final match from the optimized policies are defined in the ‘matching/algorithm/route’ module. The rest of the reference implementation consists of supporting data structures and algorithms, such as the graph model implementations, the shortest path routing algorithms, the road network import algorithms, the customized route and edge data models, and other algorithms and data structures that can theoretically be exchanged or replaced with different implementations in different programming languages.

Furthermore, many insights from the implementation comparison benchmarks can be applied to other implementations. [Table 4.22](#) provides an

Table 4.22: *General code applicability and transferability* of the evaluated optimizations and technologies – The table provides an overview of the presented technologies and their hypothetical applicability to other domains in the context of developing different implementations. The checkmark ✓ stands for yes, this technology can be useful in this domain. The crossmark × stands for no, this technology is probably not useful in this domain. Abbreviations are: Map Matching (MM), Routing Engines (RE), Reinforcement Learning (RL), General Software (GS)

Technology	MM	RE	RL	GS
Graph Preparation	✓	✓	×	×
General Distance Limit	✓	✓	×	×
Distance Limit Factor	✓	✓	×	×
Shortest Path Memory Layout	✓	✓	×	×
Shortest Path Early Stopping	✓	✓	×	×
Dijkstra and A-star	✓	✓	×	×
Edge Weight Precomputation	✓	✓	×	×
Edge Data Precomputation	✓	✓	×	×
Spatial Reference Systems	✓	✓	×	×
Network Topology Simplification	✓	✓	×	×
Graph Data Layout	✓	✓	×	×
Spatial Indices	✓	✓	×	×
Trajectory Simplification	✓	×	×	×
Candidate Search Algorithms	✓	×	×	×
Candidate Adoption	✓	×	×	×
Map Matching Metrics	✓	×	×	×
Beneficial Preordering	✓	×	✓	✓
Index-Mapping	✓	×	✓	✓
Memory-Mapping	✓	✓	✓	✓
Referencing vs. Copying	✓	✓	✓	✓
Cache Layers	✓	✓	✓	✓
Multithreading	✓	✓	✓	✓
rpmalloc	✓	✓	✓	✓

overview of which optimization technologies are useful for other implementations. For example, the memory mapping concept and graph data layout are widely applicable to many tools using large graphs. Similarly, many

performance optimizations for the shortest path routing can be applied to various routing engine and map matching problems. Additionally, the multiple cache layers and precomputation concepts can benefit various problems involving repeated calculations. Moreover, thread-local memory allocation for multi-threading applications and employing a more sophisticated memory allocator are generally applicable concepts. In conclusion, our novel map matching algorithms and evaluated implementation concepts can support existing and future developments in map matching research, general routing engines, reinforcement learning, and other related fields.

5 Evaluation

To evaluate the performance of our novel map matching approach, we conducted an extensive benchmark of different operating modes of our solution. In this section, we compare its performance against the existing third-party open source software solutions presented in [Table 2.1](#). The benchmark suite itself is referenced in [section 1.4](#). It also contains the code for the artificial ground truth generation method.

All benchmarks were run on a high-performance dedicated server on Ubuntu 22.04 LTS with $2\times$ AMD EPYC 7742 64-Core Processor (Dual-Socket) and 1 TB DDR4-2400 SDRAM main memory (NUMA-aware) on a local 1 TB NVMe M.2 PCIe 3.0 SSD.

5.1 Default Settings and Algorithms

Although all existing third-party solutions use HMMs and the Viterbi algorithm, they differ greatly in their surrounding algorithms. For example, they have different road network preparation algorithms, candidate search algorithms, and metrics, as well as different spatial reference systems and geometric calculation methods. Therefore, identifying common parameters is futile, and all solutions have different default settings.

This gave us two options: We could try to find settings that are as close as possible to each other, to see how well the tools work with comparable settings, or we could use the default settings for each tool and argue that the authors knew best which settings should be used for their solution.

We opted for the latter concept, since it is also more practical, given that end-users generally start with the default settings. Thus, using the default settings gives a practical impression of the tools, but it does not demonstrate the impact of alternative settings.

Table 5.1: *Tool versions and coordinate systems* of all evaluated open source software map matching tools – The versions are the version tag or a date if no version was provided. The only supported spatial reference system (SRS) for all tools except Fast Map Matching (FMM) and Map Matching 2 (MM2) is WGS-84. FMM ignores the SRS, and MM2 supports various SRSs, such as geographic, spherical, and Cartesian coordinate systems and calculations. Abbreviations: Open Source Routing Machine (OSRM), World Geodetic System (WGS)

Software	Version	SRS	Coordinate System
Barefoot ¹⁴	2021-08-18	WGS-84	Geographic
FMM ¹⁵	2024-04-22	None	Cartesian
GraphHopper ¹⁶	10.2	WGS-84	Spherical
OSRM ¹⁷	v6.0.0	WGS-84	Spherical
Valhalla ¹⁸	3.5.1	WGS-84	Spherical
MM2 ¹⁹	v1.1.7	Flexible	All above

In [Table 5.1](#) we provide an overview of all open source software map matching tools that we evaluate in this work, including their version number or version date. Additionally, we provide a comparison of the spatial reference systems and coordinate systems. Of all the solutions compared, only our Map Matching 2 solution supports all three spatial reference systems and coordinate system computations.

Concerning trajectory simplification, candidate adoption, and the used metrics, [Table 5.2](#) provides an overview. Only OSRM supports a simple trajectory simplification algorithm, but the authors disabled it by default, so we kept this setting. Most tools use the timestamp of the GNSS coordinates to sort them. However, only Barefoot and the ST-Matching algorithm in Fast Map Matching also use them in their metrics for temporal computations.

In [Table 5.3](#), we compare the candidate search methods and their default parameters for all tools. Most tools use a circular radius, while only Fast

Table 5.2: *Tool metrics and advanced functionality* – Comparison of the metric terms, candidate adoption (CA), and trajectory simplification (TS), on all evaluated open source software map matching tools. All tool metrics rely on distances and lengths according to [subsection 3.1.7](#). Barefoot and Fast Map Matching (FMM) in its ST-Matching (STM) mode also use temporal metrics. Our Map Matching 2 (MM2) implementation also uses azimuths, direction changes, and turns, but no temporal metrics. Abbreviations: Open Source Routing Machine (OSRM), Hidden Markov model (HMM)

Software	TS	CA	Metrics
Barefoot	No	No	Distance, Length, Temporal
FMM / HMM	No	No	Distance, Length
FMM / STM	No	No	Distance, Length, Temporal
GraphHopper	No	No	Distance, Length
OSRM	Off	No	Distance, Length
Valhalla	No	No	Distance, Length
MM2	On	On	Distance, Length, Azimuth, Directions, Turns

Table 5.3: *Tool candidate search methods and default metrics* of all evaluated open source software map matching tools – All tools filter candidates by distance. Fast Map Matching (FMM) and Map Matching 2 (MM2) also use k -nearest neighbor (KNN) search (i.e., k) to limit the candidates. MM2 also supports KNN-only or ‘BBox and Distance’-only (i.e., circular radius) modes. The radius of MM2 is dynamic, depending on the track density. Abbreviations: Bounding Box (BBox), Open Source Routing Machine (OSRM)

Software	Candidate Search	k	Radius
Barefoot	BBox, Distance	/	200 m
FMM	KNN, BBox, Distance	8	300 m
GraphHopper	BBox, Distance	/	20 m
OSRM	Nearest, Distance	/	15 m
Valhalla	Tile Bins, Distance	/	50 m – 100 m
MM2	KNN, BBox, Distance	16	200 m – 10 km

Map Matching and our Map Matching 2 solution use a combined approach. Valhalla and our Map Matching 2 solution are the only ones that allow for a dynamic adjustment of the search radius. Further default parameters can be found in the corresponding software documentations.

5.2 Default Metrics Weights

Our metric after [Equation 3.3](#) contains weights for the individual terms because they have different impacts on achieving good results. However, the individual impact of each term is unknown, which is why we employed a hyperparamter tuning for finding plausible default values. We used Bayesian optimization with Gaussian processes on an older version of the Hof data set, which is described in more detail in [subsection 5.4.5](#), and selected the default weights after reviewing various optimization runs.

Table 5.4: *Optimized metric weights* for our map matching software – These values result from rounding the results of 1000 runs of our software on the Hof data set with Bayesian optimization with Gaussian processes. Abbreviations: Markov decision process (MDP), Hidden Markov model (HMM)

Method	Weight				
	Distance	Length	Azimuth	Directions	Turn
MDP	0.8	0.3	0.5	0.6	0.5
HMM	0.8	0.1	0.2	0.8	0.6

As shown in [Table 5.4](#), the hypertuning considers distances and direction changes to be the most relevant factors for achieving good results. This outcome is to be expected, because GNSS measurements are close to the ground truth GNSS position in practice, and fewer direction changes more closely resemble actual movement patterns, as displayed by the original floating car data track shapes. Furthermore, the direction changes allow for collapsing track points in our candidate adoption feature since they penalize round-trips and detours. This seems favorable for them to play a more relevant role in our metrics. The azimuth and turn are also relevant

because the heading of the GNSS track is expected to resemble the heading of the ground truth route. However, in comparison to the distance and direction changes term, they are less relevant for finding good matches. This makes sense because the azimuth comparisons are more impacted by noise, and the turns are only heuristically approached. Interestingly, according to the hypertuning, the length is the least relevant term, although it is generally the only other major term in the other third-party open source software map matching solutions that we cover in this work. Nevertheless, it is reasonable that the length is the least relevant term because GNSS distances tend to overestimate actual traveled distances. Therefore, the GNSS noise has a significant impact and distorts the lengths. Finally, we note that, due to the difference between MDPs using absolute rewards and HMMs using relative probabilities, the weights are not exactly equal though they have a similar rank.

Even though we used hundreds of tracks to compensate for individual tendencies, our hyperparameter tuning may contain a bias with respect to the data set on which we trained it. However, we hypothesize that the default weights we selected are generally well-suited. Thus, we verify them by using them unaltered in all real and artificial benchmark data sets. This will demonstrate that our hypertuned default weights perform well on data sets that they were not trained on, similar to selected default weights for a navigational routing applications. Consequently, this will confirm that our default weights are generally a reasonable choice.

5.3 Experiment Framework Conditions

To evaluate the performance of our new map matching approach, we conducted an extensive comparison benchmark on the real-world data sets introduced in [subsection 2.4.1](#), the Hof data set that we introduce in [subsection 5.4.5](#), and on several artificial data sets generated with SUMO and our artificial GNSS noise generation process, as described in [section 3.2](#).

To compare the accuracy after [Equation 2.2](#), calculated by our comparison algorithm as defined in [subsection 3.1.4](#), and the computational resource usage, measured by the `/usr/bin/time` command, concerning the elapsed time in seconds, the total CPU time in seconds (i.e., the sum of ‘user’ and ‘sys’ CPU times), and the main memory usage (i.e., RAM consumption) as resident set size in MiB, we ran an extensive, repeatable benchmark suite on all real-world and artificial map matching data sets, with all third-party open source software map matching tools from [subsection 2.4.3](#) and our novel Map Matching 2 approach. All benchmarks in this chapter use multi-threading with up to 256 available threads on 128 physical cores on two CPU sockets (see the hardware description in [chapter 5](#)).

We measured the preparation, the startup, and the matching phases for all tools. The preparation phase (i.e., importing the road network) plays an important role in the total resource consumption of matching a data set, especially when matching a small number of tracks, since the preparation phase can take much longer than the matching phase. The startup phase (i.e., booting the software and loading the prepared road network) allows for determining how long it takes to become ready for accepting matching requests. We measured it by matching a singular empty track on the respective road network of a data set, after which we immediately killed the software. Therefore, if only the resource usage required for the matching itself is of interest, the startup phase resources can be subtracted from the matching phase resources (i.e., the matching phase is the total resource usage including the startup phase). For these reasons, we measured all phases to provide a complete picture of the performance of all evaluated map matching tools on all evaluated data sets.

Barefoot uses an additional intermediate PostgreSQL database (DB) during the preparation phase for importing the OpenStreetMap road network data. The database is no longer necessary during the startup and matching phases, but its resource usage is essential to the complete picture. First, a client-side script imports the OpenStreetMap road network into the database, and then the Barefoot application reads the data from the database into its own custom road network representation format. Thus, for each step during the preparation phases of Barefoot, two tools run simultaneously. Regarding the total resource usage of Barefoot, the total elapsed time is the sum of the

‘Prepare Import DB’ and ‘Prepare Startup DB’ times, the total CPU usage is the sum of all four ‘Prepare’ runs, and the main memory limit is the higher of the sum of the two ‘Prepare Import’ or the two ‘Prepare Startup’ runs.

Fast Map Matching requires an initial conversion of the OpenStreetMap road network data into a Shapefile data format that Fast Map Matching can read in both HMM and ST-Matching modes. In the HMM mode, an upper bounded origin-destination table (UBODT) must be generated in advance from the Shapefile. The UBODT is then read during the startup of the software in the HMM mode, in addition to the Shapefile. For the ST-Matching mode, only the Shapefile is read. The total resource consumption for Fast Map Matching is the sum of the individual resource consumption values from both ‘Prepare Shapefiles’ and ‘Prepare UBODT’.

GraphHopper converts the road network into its own format during the first startup and reads the prepared data in the subsequent startups. OSRM, Valhalla, and our own Map Matching 2 software have extra preparation steps to convert the road network data into the application-specific formats before starting the software in the actual matching phases.

For our Map Matching 2 software, we always use the ‘Output’ memory mapping preparation mode, as evaluated in [Table 4.2](#), and we always compare our novel MDP model with and without candidate adoption against our HMM implementation, which represents the historical approach. All other parameters are left at their default settings, as described in [Table 5.2](#) and [Table 5.3](#). In an additional experiment, we also evaluate trajectory simplification in combination with candidate adoption and examine how our trajectory simplification algorithm functions with the third-party map matching tools. We hypothesize that our novel Map Matching 2 method will excel on all benchmarks due to its robust and careful design of the methods, metrics, algorithms, and models, which should work accurately and robustly in most, if not all, evaluated scenarios.

One challenge with the third-party map matching data sets introduced in [subsection 2.4.1](#) is the fact that they do not provide the road network data in the OpenStreetMap data format, but rather in custom text-based files.

However, Barefoot, GraphHopper, OSRM, and Valhalla can only import OpenStreetMap road network data. Fast Map Matching, on the other hand, can only import Shapefiles. Still, the authors provided information on how to convert OpenStreetMap data into the Shapefile format that Fast Map Matching needs. Therefore, we have to convert the third-party data set road networks into the OpenStreetMap data format so that all tools can import them. Thus, we have implemented an export routine in our Map Matching 2 software that is able to output any imported road network in the OpenStreetMap native data format. In combination with native import routines for every third-party road network data format evaluated in this work, we can convert all map matching data sets so that all map matching tools can use them.

Of course, the exported OpenStreetMap data lacks tags that regular OpenStreetMap data contains, because the custom text-based format does not include this information. For example, the ‘highway’ road tag type with different values is missing, such as ‘motorway’ for freeways, or ‘primary’, ‘secondary’, or ‘tertiary’ for different classes of road importance. However, most tools require a ‘highway’ tag for being able to import the provided OpenStreetMap data as a road network, to distinguish geographic lines representing roads from other lines, such as walls, borders, or hedges. Therefore, we add an artificial ‘highway’ tag with the ‘unclassified’ value to every exported road. This allows for all evaluated open source software map matching solutions to import all road networks from all evaluated map matching data sets.

The preparation benchmarks were repeated 5 times, except for the Fast Map Matching Shapefile and UBODT benchmarks, which were repeated only 3 times. The startup and matching phases were repeated 10 times. For all third-party data sets, the UBODT distance limit for Fast Map Matching was set to 1000 m since the default of 3000 m led to extremely high resource consumption. However, for our own real-world and artificial data sets, we were able to keep the default value of 3000 m. From all benchmark runs, we used the best times, CPU resources, and main memory values, since these values are closer to the optimum performance that each tool can theoretical achieve if no other processes of the running operating system would interfere. Concerning the Java applications, we want to add that the

fluctuation between the best and worst runtime performance results may be higher in practice than for the C++ applications, because Java applications are just-in-time compiled and contain a runtime garbage collection that interferes with execution.

5.4 Real-World Data Sets

A key aspect of map matching data sets is that they provide the GNSS tracks alongside the corresponding correct solutions and the road network from which the correct solutions are derived. This leads to challenges when comparing the results from different map matching tools. First, each map matching tool expects the input tracks in a different format, often requiring additional conversion steps if the map matching tool cannot parse the input data directly. Second, the road network is processed differently depending on the tool, e.g., different OpenStreetMap tags may be used. This results in different road network graphs, and differences in the matches may already emerge from this factor alone. For example, in the artificially generated data sets, SUMO uses a different internal road network representation for generating the artificial ground truth than our Map Matching 2 software uses for matching the generated tracks, even though both applications import the same original OpenStreetMap network. Nevertheless, these issues are realistic and will occur similarly in actual use cases. Therefore, for a valid practical comparison of different map matching implementations, these differences are left in place.

In [Table 5.5](#), we provide an overview of the real-world map matching data sets that we introduced in the literature review in [subsection 2.4.1](#), and our new Hof data set, which we introduce shortly after. Of these data sets, only two are considered real ‘ground truth’, the rest are hand-corrected. Only our own Hof data set contains exact geometric lines, the other data sets only contain lists of edge IDs, which are converted into geometric lines for our comparison algorithm. The main issue with the correct solution provided as list of edge IDs is that the start and stop positions of the provided solution can only start and end at the beginning and ending of an edge.

Table 5.5: *Map matching data sets* – Overview of the ground truth (GT) Seattle data set (DS-A) (Newson and Krumm 2009) and the Melbourne data set (DS-B) (Li et al. 2014), and the map matching GIS CUP data set (DS-C) (Mohamed et al. 2012), the Worldwide data set (DS-D) (Kubicka et al. 2015), and the Hof data set (DS-E) (Wöltche 2023). Abbreviations: Washington (WA), Identifier (ID), Is ground truth (GT?)

Set	Region	Tracks	GT Format	GT?
DS-A	Seattle, WA, USA	1	List of Edge IDs	Yes
DS-B	Melbourne, Australia	1	List of Edge IDs	Yes
DS-C	Seattle, WA, USA	10	List of Edge IDs	No
DS-D	Worldwide	100	List of Edge IDs	No
DS-E	Hof, Bavaria, Germany	263	Geometric lines	No

However, a track typically starts alongside an edge and not within a junction. Consequently, there is generally a discrepancy between the matches and the provided solutions. Although it is possible to return the complete edges in the matches, which would result in a higher computed accuracy towards the provided solutions, the actual accuracy is lower, because the match was artificially extended beyond the spatial region covered by the input track. However, since this discrepancy affects all map matching algorithms as long as they export a match as a geometric line that is not stretched to the beginning and ending of an edge, as is the case with the evaluated tools, this issue is neglected for a fair comparison, and the provided solutions are assumed to be correct.

The Seattle data set from Newson and Krumm (2009) provides one track with an ≈ 2 h drive through Seattle, Washington, USA, with corresponding ground truth. The network is provided as a custom text file with geometric lines for the edges. The ground truth data was collected by the drivers themselves, which makes it very accurate. However, the ground truth match is provided as a list of edge IDs of the network file, even though the measured track starts and ends within an edge. In this specific case, there is a discrepancy of ≈ 200 m between the provided ground truth and the actual movement.

The Melbourne data set from Li et al. (2014) also contains one track with corresponding ground truth. Similarly, the ground truth is provided as a list of edge IDs with a custom road network text file. Additionally, the original OpenStreetMap data is provided, from which the custom road network text file was derived. However, the OpenStreetMap road network contains more roads than the derived custom export. Therefore, matching with the OpenStreetMap road network may produce different results than matching with the custom road network extract.

The GIS CUP data set from Mohamed et al. (2012) contains 10 tracks in the same area as the Seattle data set, in Seattle, Washington, USA. The hand-corrected solutions are again provided as lists of edge IDs, and the road network is provided as custom text files. Unfortunately, the solutions for the tracks with the IDs 2 and 7 contain very large errors and need to be corrected as follows: In ‘output_02.txt’, the lines from IDs 701 to 760 with the incorrect edge IDs 1263704 between the lines with the correct edge IDs 180859 need to be removed. In ‘output_07.txt’, the lines from IDs 873 to 932 with the incorrect edge IDs 1263704 between the lines with the correct edge IDs 180859 need to be removed. There are more minor errors in the data set, but this correction is the most significant for achieving good accuracy.

The Worldwide data set from Kubicka et al. (2015) contains 100 tracks from around the world that were matched using a custom algorithm and then hand-corrected by the authors. The authors did not record the tracks themselves, but selected them from the OpenStreetMap GPX (GPS Exchange Format) project. This allowed to create a diverse, worldwide data set, but it also means that, unlike the Seattle data set and the Melbourne data set, this data set is also not ‘ground truth’ because the correct match in the road network remains unknown. Due to the hand-correction, the supposedly correct matches that are also provided as a list of edge IDs contain some errors, i.e., oversights in the hand-correction. Since the tracks are spread all over the world, a separate road network is provided as a custom text file for each track. As such, no two tracks share the same road network, though there are some overlaps.

Since all of these map matching data sets have in common that the ground truth is provided as lists of edge IDs, which leads to the previously described discrepancies between the real movements and the provided solutions, we also created our own new Hof data set for map matching in Wöltche (2023). This data set was also hand-corrected after an initial match with an earlier, not yet hypertuned version of our map matching solution and as such, it is also not ‘ground truth’. However, the supposedly correct matches are not provided as lists of edge IDs, but rather as geometric lines. Since our comparison algorithm from [subsection 3.1.4](#) is able to compare geometric matches directly, we can provide solutions that are more spatially accurate to the real movement than those based on lists of edge IDs. The GNSS tracks are anonymized with written permission from a floating car data set that we received from Map and Route GmbH & Co. KG²⁹. The anonymization was performed by removing all directly connected points within a 300 m circle around the original start and end position. Thus, the true source and destination of the movements are blurred and unrecognizable, ensuring that the data set complies with data protection laws.

We created the Hof data set by selecting only tracks that lie entirely within the city boundary of Hof, Bavaria, Germany, where our research institute is located. This enabled us to infer local knowledge of the road network to create plausible solutions for the data set. From the 1300 retained tracks, we randomly selected $\approx 20\%$, resulting in 263 tracks, for which we hand-corrected the corresponding matches. We also provide the OpenStreetMap road network from which our matches are derived. Thus, the Hof data set is directly compatible with all third-party map matching solutions that support OpenStreetMap road network data import.

The metadata for the road networks of all data sets is presented in [Table 5.6](#). The numbers of nodes and edges directly impact the preparation times, i.e., the time for processing the road network into the application-specific road network graph formats. Furthermore, the metadata of the tracks of all data sets is presented in [Table 5.7](#). Based on the (mean) point count, the segment lengths, and the timestamp differences, we can see that the tracks in the Hof data set are sparser than those in the other data sets. Additionally,

²⁹<https://www.mapandroute.de/>

Table 5.6: *Map matching data set road network metadata* – Overview of the map and road network metrics of the data set regions. The area is a rough estimate computed from the convex hull area around the nodes. The numbers of nodes and edges come from the original data as provided, without road network simplification. The length is the total road network length, i.e., the sum of all edge lengths. Abbreviations: Seattle data set (DS-A), Melbourne data set (DS-B), GIS CUP data set (DS-C), Worldwide data set (DS-D), Hof data set (DS-E)

Set	Area	Nodes	Edges	Length
DS-A	$\approx 7140 \text{ km}^2$	133 460	306 182	$\approx 43\,624 \text{ km}$
DS-B	$\approx 567 \text{ km}^2$	107 315	208 632	$\approx 10\,496 \text{ km}$
DS-C	$\approx 207\,753 \text{ km}^2$	535 452	1 283 540	$\approx 442\,549 \text{ km}$
DS-D	$\approx 29\,712 \text{ km}^2$	2 193 006	4 400 473	$\approx 194\,342 \text{ km}$
DS-E	$\approx 88 \text{ km}^2$	19 846	39 872	$\approx 1102 \text{ km}$

all tracks have zigzag movements with 180° direction changes between adjacent segments. Nevertheless, most segments are straightforward, with the Melbourne data set having more jitter than the others. Furthermore, we can see that only the Melbourne data set has a constant time step of 1 s, while the other data sets have more or less jitter. In fact, the segment lengths and timestamp differences show that the Seattle data set, the GIS CUP data set, the Worldwide data set, and the Hof data set all contain significant outliers. Finally, the min segment length shows that all data sets contain spatially duplicate points.

5.4.1 Seattle Data Set

In the first experiment, we benchmarked the Seattle data set from Newson and Krumm (2009), available at *Microsoft Research*⁸. In their original work, the authors evaluated their own solution by adding artificial noise and removing points to make the tracks more sparse. However, our work focuses on comparing multiple tools on different data sets. Nevertheless, we will also conduct an experiment involving artificial noise and increasing sparseness later in [section 5.5](#).

Table 5.7: *Map matching data set metadata* in comparison for each data set (DS) – The range of values is shown from min to max, with median (med), mean, standard deviation (std), and count representing the number of values. The compared metadata are points counts (P_c), track lengths in km (T_l), segment lengths in m (S_l), time differences in s (T_d), and direction changes in $^\circ$ (D_c). Data sets (DS): Seattle data set (A), Melbourne data set (B), GIS CUP data set (C), Worldwide data set (D), Hof data set (E)

Metric	DS	min	med	mean	max	std	count
P_c	A						7531
	B						2512
	C	884	1248	1444	2368	530	14 436
	D	262	1440	2473	19 313	3054	247 251
	E	2	41	56	343	51	14 651
T_l km	A				119.52		1
	B				22.84		1
	C	14.04	33.44	34.69	52.57	11.98	10
	D	7.49	22.22	40.79	153.96	37.52	100
	E	0.14	4.23	4.73	20.53	3.30	263
S_l m	A	0.0	13.75	15.87	1404.65	25.58	7530
	B	0.0	7.92	9.09	45.21	8.21	2511
	C	0.0	24.25	24.05	1534.74	21.26	14 426
	D	0.0	10.88	16.50	1250.74	18.55	247 151
	E	0.0	63.02	86.51	1639.35	103.30	14 388
T_d s	A	1.0	1.0	1.01	35.0	0.54	7530
	B	1.0	1.0	1.0	1.0	0.0	2511
	C	1.0	1.0	1.0	39	0.4	14 426
	D	0.0	1.0	2.0	153 064	325.8	247 151
	E	0.0	5.0	7.3	506.0	12.4	14 388
D_c $^\circ$	A	-180	0.0	0.24	180	32.69	7529
	B	-180	-0.09	-0.97	180	78.33	2510
	C	-180	0.0	-0.15	180	28.63	14 416
	D	-180	0.0	-0.07	180	30.69	247 051
	E	-180	0.0	0.15	180	65.40	14 125

Table 5.8: *Seattle data set (DS-A) prepare and startup benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the preparation and startup phases with multi-threading on the Seattle data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Database (DB), Upper bounded origin-destination table (UBODT), ST-Matching (STM)

Software	Mode	Time	CPU	MEM
Barefoot	Prepare Import	27.63	22.30	873
Barefoot	Prepare Import DB	29.52	13.23	203
Barefoot	Prepare Startup	3.90	47.00	445
Barefoot	Prepare Startup DB	5.48	0.54	36
FMM	Prepare Shapefiles	84.90	99.87	2536
FMM	Prepare UBODT	2.04	105.19	141
GraphHopper	Prepare Startup	10.29	699.38	1314
MM2	Prepare	2.91	2.96	525
OSRM	Prepare	6.58	346.17	355
Valhalla	Prepare	15.49	26.00	531
Barefoot	Startup	2.87	26.19	246
FMM	Startup FMM	3.18	7.62	316
FMM	Startup STM	1.19	5.31	136
GraphHopper	Startup	2.10	8.48	242
MM2	Startup	0.07	0.15	22
OSRM	Startup	0.13	0.13	100
Valhalla	Startup	0.12	0.11	42

In [Table 5.8](#), we can see the preparation and startup times of all evaluated tools on the Seattle data set. Due to its use of an additional PostgreSQL database (DB) server during preparation, the preparation of Barefoot itself is delayed for additional time until the database is ready for writing or reading. The main reason why the conversion of the OpenStreetMap data to Shapefiles for Fast Map Matching needs so many resources is that the provided conversion script relies on Python. As discussed in [subsection 4.1.2](#), Python is one of the slowest and most resource-hungry programming languages in terms of runtime performance. GraphHopper also needs many resources in comparison because it is based on Java. The UBODT generation in Fast Map Matching is comparably fast, but only

because the distance limit is set to 1000 m instead of the default 3000 m. However, even this smaller UBODT file causes Fast Map Matching to have the longest startup time. OSRM is one of the faster applications, but uses more CPU resources for preparing the data than Valhalla, which is also a C++ application. Our Map Matching 2 application is the fastest for preparation, though it does not use the least amount of memory, which OSRM does. The Java applications Barefoot and GraphHopper take much longer to start up than the C++ applications OSRM, Valhalla, and Map Matching 2. Employing memory mapping, which OSRM, Valhalla, and Map Matching 2 use, leads to near-immediate startup times.

Table 5.9: *Seattle data set (DS-A) match benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the Seattle data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA), Trajectory simplification (TS)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	55.38	4303.92	1082	0.99719
FMM	HMM	4.00	9.19	324	0.00000
FMM	STM	3.89	9.09	161	0.86652
GraphHopper	HMM	4.00	22.57	376	0.86477
MM2	MDP	1.24	1.35	100	0.99616
MM2	MDP (CA)	0.73	0.78	66	0.98871
MM2	HMM	1.17	1.28	86	0.97334
MM2	HMM (CA)	0.73	0.82	68	0.97054
OSRM	HMM	0.98	0.21	102	0.00000
Valhalla	HMM	2.40	20.44	5472	0.79591
Barefoot	HMM (TS)	8.53	349.52	846	0.98798
FMM	HMM (TS)	3.24	8.21	318	0.00000
FMM	STM (TS)	1.33	6.53	140	0.96035
GraphHopper	HMM (TS)	3.58	17.72	329	0.96152
OSRM	HMM (TS)	0.97	0.24	104	0.00446
Valhalla	HMM (TS)	2.28	20.41	5458	0.48241

In [Table 5.9](#), we can see the matching results on the Seattle data set. For this singular track, Barefoot achieves the highest accuracy, but with a tremen-

dous CPU resource consumption. Our Map Matching 2 application with our novel MDP model with candidate adoption enabled is only slightly less accurate, but requires only a tiny fraction of the CPU resources. In terms of accuracy, it has the best runtime performance, and it uses the least amount of memory. OSRM may require fewer resources, but since it fails to match the track and seemingly aborts the matching process, the results are not actually comparable.

At this point, we want to note that comparing a singular track is not necessarily representative of the quality of all map matching tools. For example, Barefoot may have been specifically tuned for this particular publicly available track during development. Since the authors of Barefoot cite the work of Newson and Krumm (2009) on their code repository, this hypothesis is reasonable. Furthermore, Fast Map Matching in the HMM mode and OSRM completely fail to match the track, which offers no insight into how well these tools generally work. Regarding the remaining results, GraphHopper and Valhalla produce suboptimal results. The memory consumption of Valhalla is especially extreme on this track. Additionally, disabling our candidate adoption feature or relying on the older HMM model decreases the accuracy of our Map Matching 2 application. While disabling candidate adoption reduces the resource consumption, it also decreases the accuracy for this track.

In an additional experiment, we applied our trajectory simplification algorithm to the track and fed it into all third-party tools, except for our own Map Matching 2 application, which already uses trajectory simplification by default. The results were mixed, Fast Map Matching with the ST-Matching algorithm and GraphHopper increased significantly in accuracy, Barefoot decreased slightly and Valhalla decreased substantially. OSRM can now match a tiny fraction of the track, but Fast Map Matching with HMM still fails. Again, since this is only a singular track, these results have to be treated with caution.

5.4.2 Melbourne Data Set

In the next experiment, we evaluated the Melbourne data set from Li et al. (2014), available at the *Internet Archive*⁹ (originally available at the *University of Melbourne*). As with the previous experiment, these results should be taken with caution as they are again based on a singular track.

Table 5.10: *Melbourne data set (DS-B) prepare and startup benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the preparation and startup phases with multi-threading on the Melbourne data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Database (DB), Upper bounded origin-destination table (UBODT), ST-Matching (STM)

Software	Mode	Time	CPU	MEM
Barefoot	Prepare Import	17.66	18.84	442
Barefoot	Prepare Import DB	19.46	7.74	202
Barefoot	Prepare Startup	3.18	36.64	232
Barefoot	Prepare Startup DB	4.78	0.44	32
FMM	Prepare Shapefiles	29.34	44.39	871
FMM	Prepare UBODT	1.16	85.82	79
GraphHopper	Prepare Startup	9.47	751.55	904
MM2	Prepare	0.95	0.97	179
OSRM	Prepare	4.06	184.72	189
Valhalla	Prepare	24.66	36.52	462
Barefoot	Startup	2.66	30.22	213
FMM	Startup FMM	2.36	6.91	236
FMM	Startup STM	0.50	4.79	77
GraphHopper	Startup	2.05	8.44	247
MM2	Startup	0.07	0.15	24
OSRM	Startup	0.08	0.07	36
Valhalla	Startup	0.12	0.12	40

The preparation results in Table 5.10 for the Melbourne data set are similar to those for the Seattle data set. Once again, our Map Matching 2 preparation and startup are the fastest.

Table 5.11: *Melbourne data set (DS-B) match benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the Melbourne data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA), Trajectory simplification (TS)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	23.54	1537.03	829	0.92290
FMM	HMM	2.65	7.81	241	0.57617
FMM	STM	1.78	6.82	87	0.73403
GraphHopper	HMM	3.63	18.86	319	0.93519
MM2	MDP	0.36	0.42	68	0.99316
MM2	MDP (CA)	0.26	0.32	64	0.99355
MM2	HMM	0.35	0.42	66	0.99355
MM2	HMM (CA)	0.26	0.34	62	0.96273
OSRM	HMM	1.17	0.35	44	0.00039
Valhalla	HMM	2.23	19.91	5454	0.99427
Barefoot	HMM (TS)	6.95	188.62	779	0.96275
FMM	HMM (TS)	2.39	7.56	238	0.99428
FMM	STM (TS)	0.56	5.53	77	0.99428
GraphHopper	HMM (TS)	3.45	17.34	290	0.99427
OSRM	HMM (TS)	0.97	0.17	38	0.12072
Valhalla	HMM (TS)	2.18	20.36	5430	0.96272

The matching results in Table 5.11 for the Melbourne data set are similar to the previous results from the Seattle data set. However, Valhalla is the most accurate solution this time, with our Map Matching 2 application in second place again. OSRM nearly fails to match the track again, but Fast Map Matching gives a better picture here, although not a good one. Interestingly, disabling candidate adoption on the MDP or using the old HMM model is slightly more accurate on this particular track. This reinforces our statement that matching results for singular tracks have to be taken with caution, as this result was unexpected. For individual tracks, tuning the metric weights specifically can greatly influence the results. Nevertheless, we can see that our tool is one of the most accurate ones while also needing very few

resources. For example, Valhalla again needs extreme memory resources for achieving this level of accuracy, whereas our Map Matching 2 solution only needs a tiny fraction for achieving similar accuracy.

Using trajectory simplification on this track dramatically improves the accuracy of all tools, except for Valhalla. Similarly to how enabling trajectory simplification decreased the accuracy of Barefoot on the Seattle data set, it decreases the accuracy of Valhalla on the Melbourne data set. These results support the hypothesis that Barefoot could have been tuned on the Seattle data set and Valhalla on the Melbourne data set during development. While the results suggest this, there is no clear evidence. Although the Valhalla authors also cite the work of Newson and Krumm (2009) on their code repository, they did not achieve a good accuracy on the Seattle data set, but here on the Melbourne data set. Nevertheless, the results also support our statement that the matching results for singular tracks have to be taken with caution, as hidden biases may exist within the data or the tools.

As we will see with the following data sets, which all consist of multiple tracks, these specific odds will not be repeated. Nevertheless, the Seattle data set and the Melbourne data set are the only real ground truth data sets and are therefore valuable to review.

5.4.3 GIS CUP Data Set

In this experiment, we evaluate the GIS CUP data set from Mohamed et al. (2012), available at the *Internet Archive*¹⁰ (originally available at the *University of Washington*). As explained in section 5.4, we manually corrected a data-set-specific error in the provided solutions.

The preparation benchmarks in Table 5.12 again show a similar result, although the Shapefiles preparation for Fast Map Matching becomes a significant limiting factor. Road networks larger than this may be unfeasible if they are not already provided in the Shapefile format. The UBODT with the 1000 m limit still computes in a reasonable amount of time but leads to a very long startup time of Fast Map Matching. When we compare the

Table 5.12: *GIS CUP data set (DS-C) prepare and startup benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the preparation and startup phases with multi-threading on the GIS CUP data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Database (DB), Upper bounded origin-destination table (UBODT), ST-Matching (STM)

Software	Mode	Time	CPU	MEM
Barefoot	Prepare Import	144.77	93.45	2520
Barefoot	Prepare Import DB	146.71	76.10	248
Barefoot	Prepare Startup	14.49	68.52	1814
Barefoot	Prepare Startup DB	16.10	2.01	22
FMM	Prepare Shapefiles	520.53	534.36	15260
FMM	Prepare UBODT	8.82	377.46	488
GraphHopper	Prepare Startup	25.38	645.37	4568
MM2	Prepare	18.80	19.09	2613
OSRM	Prepare	25.77	1487.30	1670
Valhalla	Prepare	49.45	156.48	4918
Barefoot	Startup	9.69	45.24	903
FMM	Startup FMM	12.71	17.31	1116
FMM	Startup STM	5.45	9.77	486
GraphHopper	Startup	2.07	8.42	335
MM2	Startup	0.07	0.11	24
OSRM	Startup	0.43	0.43	436
Valhalla	Startup	0.12	0.12	42

startup time of Fast Map Matching against the matching time below, we can see that Fast Map Matching uses most of the time in the matching phase for reading the precomputed UBODT and the Shapefiles, i.e., the actual matching of the 10 tracks is relatively fast. Overall, our Map Matching 2 software is the fastest for the entire preparation and startup process, though OSRM uses the least memory while simultaneously taking much more CPU time. It becomes apparent at this point that trading memory for computing power can be beneficial in some cases, such as this one.

Since the GIS CUP data set is the first data set in comparison consisting of more than one track, the results in [Table 5.13](#) for a ranking of the matching

Table 5.13: *GIS CUP data set (DS-C) match benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the GIS CUP data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA), Trajectory simplification (TS)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	33.63	3043.60	4633	0.97636
FMM	HMM	13.36	21.24	1118	0.83617
FMM	STM	6.23	15.47	513	0.38312
GraphHopper	HMM	5.19	36.23	898	0.71669
MM2	MDP	0.59	3.66	204	0.97947
MM2	MDP (CA)	0.42	2.53	168	0.97852
MM2	HMM	0.58	3.61	178	0.97947
MM2	HMM (CA)	0.42	2.54	154	0.93745
OSRM	HMM	2.14	0.84	494	0.25544
Valhalla	HMM	2.42	21.53	5644	0.94265
Barefoot	HMM (TS)	15.22	507.05	1653	0.97740
FMM	HMM (TS)	12.85	18.34	1114	0.40875
FMM	STM (TS)	5.62	11.20	486	0.81676
GraphHopper	HMM (TS)	3.81	23.32	469	0.87433
OSRM	HMM (TS)	2.06	0.61	458	0.15490
Valhalla	HMM (TS)	2.31	21.65	5574	0.66712

performance are more trustworthy. As we can see here, our novel Map Matching 2 application with candidate adoption on our MDP achieves the highest accuracy. Interestingly, the old HMM in comparison with candidate adoption enabled also achieves similar accuracy, but degrades much more without candidate adoption. Barefoot is very close in second place this time, but it needs a significant amount of resources to achieve similar accuracy. Fast Map Matching with HMM is much more accurate than ST-Matching. However, when we compare the resource usage with the startup times for Fast Map Matching, we can see that most of the time is spent reading the Shapefiles and the UBODT. Valhalla achieves good accuracy but again has the worst memory usage. GraphHopper has mixed results, and OSRM is again far behind in terms of accuracy.

Applying trajectory simplification improves Barefoot, GraphHopper, and ST-Matching, whereas Fast Map Matching with HMM, OSRM, and Valhalla degrade. Barefoot is especially interesting because its accuracy improves dramatically at the same time as its resource consumption. This also happened on the Melbourne data set, but on a less apparent scale. This reveals some hidden optimization potential in Barefoot.

5.4.4 Worldwide Data Set

The Worldwide data set from Kubicka et al. (2016), available under creative commons CC BY-SA 4.0¹¹ license at *Zenodo*¹² is the first spatially diverse map matching data set in comparison, because it consists of 100 tracks distributed worldwide. Therefore, the provided road networks consist of 100 individual ones. For this experiment, we merged all road networks into one large road network consisting of isolated islands. This makes the preparation and startup benchmark results comparable to those of the other data sets also consisting of a singular road network.

The results in Table 5.14 show that the worldwide road network significantly impacts the preparation results. Barefoot and Fast Map Matching already need half a minute just to start up, and they use a large amount of main memory. This illustrates why memory mapping becomes so important as the road network grows. As usual, our Map Matching 2 preparation is the fastest, and the memory consumption during preparation is second to OSRM.

The matching results in Table 5.15 for the 100 tracks show that our Map Matching 2 application is the most accurate solution again. Now, since the sample size is larger, we can also see the expected ranking of our novel MDP against the old HMM design. Furthermore, disabling candidate adoption again decreases the accuracy while improving the computational resources because fewer candidates have to be evaluated. Barefoot is close in second place again, but the amount of resources, especially the memory usage, is impractical. Fast Map Matching is the slowest application with the ST-Matching model, and the Fast Map Matching HMM mode needs

Table 5.14: *Worldwide data set (DS-D) prepare and startup benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the preparation and startup phases with multi-threading on the Worldwide data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Database (DB), Upper bounded origin-destination table (UBODT), ST-Matching (STM)

Software	Mode	Time	CPU	MEM
Barefoot	Prepare Import	288.88	209.91	3774
Barefoot	Prepare Import DB	290.82	133.08	243
Barefoot	Prepare Startup	36.18	140.29	3694
Barefoot	Prepare Startup DB	37.83	6.56	22
FMM	Prepare Shapefiles	435.04	448.76	14190
FMM	Prepare UBODT	16.07	1147.81	581
GraphHopper	Prepare Startup	55.15	1439.37	9096
MM2	Prepare	18.12	18.92	2612
OSRM	Prepare	26.58	1797.29	1732
Valhalla	Prepare	128.56	498.22	3979
Barefoot	Startup	29.03	103.02	2183
FMM	Startup FMM	33.34	37.66	2828
FMM	Startup STM	6.87	11.16	579
GraphHopper	Startup	2.23	9.20	392
MM2	Startup	0.07	0.11	24
OSRM	Startup	0.36	0.35	344
Valhalla	Startup	0.12	0.12	40

over four-fifths of its matching time just to read the precomputed UBODT. GraphHopper is also comparably slow and has poor accuracy. OSRM is the fastest tool, but its accuracy is on the lower end. Valhalla needs a lot of memory and provides only average accuracy. In fact, of all the tools tested on this data set, only Barefoot and our Map Matching 2 application have acceptable accuracy, with our Map Matching 2 solution also having practical resource consumption.

Enabling trajectory simplification does not improve the accuracy for Barefoot, but it does reduce the resource usage quite a bit, except for the memory, which remains extremely high. Valhalla again decreases in accuracy, but all

Table 5.15: *Worldwide data set (DS-D) match benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the Worldwide data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA), Trajectory simplification (TS)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	318.23	35025.15	27231	0.96750
FMM	HMM	40.98	55.28	2936	0.22204
FMM	STM	707.05	850.99	1064	0.08893
GraphHopper	HMM	133.44	811.18	4651	0.17797
MM2	MDP	6.99	54.61	1320	0.98750
MM2	MDP (CA)	3.16	29.50	984	0.98630
MM2	HMM	6.81	52.63	1160	0.97051
MM2	HMM (CA)	3.09	29.59	939	0.93540
OSRM	HMM	2.19	4.15	831	0.21996
Valhalla	HMM	4.89	42.12	7908	0.64180
Barefoot	HMM (TS)	51.63	2986.47	22583	0.96375
FMM	HMM (TS)	35.59	41.46	2833	0.32349
FMM	STM (TS)	7.90	17.34	594	0.53190
GraphHopper	HMM (TS)	15.09	164.88	2023	0.50130
OSRM	HMM (TS)	2.74	1.25	526	0.26182
Valhalla	HMM (TS)	3.63	35.04	7156	0.46380

other tools improve with our trajectory simplification applied, some drastically, such as the Fast Map Matching ST-Matching model, or GraphHopper.

5.4.5 Hof Data Set

In the final experiment on real-world data, we evaluated the Hof data set from our previous work in Wöltche (2023). For the experiments in this work, we corrected some solutions with large U-turns. In our previous work, we only corrected the matches according to the geometric shape.

However, when taking the time information into account as well, some corrections have to be made differently. These corrections were revealed by the ‘Warp detection’ feature described in [subsection 3.1.1](#) that was not yet developed in our previous work.

For this data set, we were able to use the default UBODT limit of 3000 m for Fast Map Matching since the road network is very small in comparison and has the same bounding box as the tracks have.

Table 5.16: *Hof data set (DS-E) prepare and startup benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the preparation and startup phases with multi-threading on the Hof data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Database (DB), Upper bounded origin-destination table (UBODT), ST-Matching (STM)

Software	Mode	Time	CPU	MEM
Barefoot	Prepare Import	5.41	9.79	307
Barefoot	Prepare Import DB	7.05	2.69	204
Barefoot	Prepare Startup	1.13	55.03	100
Barefoot	Prepare Startup DB	2.68	0.11	22
FMM	Prepare Shapefiles	17.31	32.36	726
FMM	Prepare UBODT	4.68	154.62	64
GraphHopper	Prepare Startup	5.48	484.38	395
MM2	Prepare	0.19	0.24	48
OSRM	Prepare	1.49	25.33	46
Valhalla	Prepare	8.67	8.17	295
Barefoot	Startup	0.85	28.81	74
FMM	Startup FMM	15.54	20.10	1416
FMM	Startup STM	0.39	4.80	66
GraphHopper	Startup	2.03	8.02	244
MM2	Startup	0.07	0.16	24
OSRM	Startup	0.05	0.04	10
Valhalla	Startup	0.12	0.12	42

As we can see in [Table 5.16](#), preparing a small road network still has a significant impact on many preparation steps. GraphHopper and the UBODT generation, in particular, need significant CPU resources. The startup time

of Fast Map Matching is exceptionally slow with the default UBODT distance limit. It should now be clear why we used lower distance limits for all other data sets. Otherwise, Fast Map Matching would have dominated the resource usage just for the startup. Our Map Matching 2 solution is very fast at preparing small road networks. As we can see, our Map Matching 2 solution is the fastest and most CPU-friendly solution for every data set so far, with only slightly higher memory consumption than OSRM during preparation.

Table 5.17: *Hof data set (DS-E) match benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the Hof data set. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA), Trajectory simplification (TS)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	18.90	2072.31	18065	0.94970
FMM	HMM	16.50	23.53	1418	0.70584
FMM	STM	1.29	14.80	66	0.49416
GraphHopper	HMM	4.76	127.33	826	0.69176
MM2	MDP	0.52	11.35	354	0.99651
MM2	MDP (CA)	0.41	6.71	142	0.97808
MM2	HMM	0.49	11.25	258	0.98488
MM2	HMM (CA)	0.38	6.17	138	0.95688
OSRM	HMM	2.79	0.99	12	0.83975
Valhalla	HMM	4.36	33.66	6656	0.72640
Barefoot	HMM (TS)	13.90	1483.60	16398	0.95777
FMM	HMM (TS)	16.94	24.07	1418	0.83964
FMM	STM (TS)	0.74	12.91	66	0.69660
GraphHopper	HMM (TS)	5.27	103.13	599	0.80471
OSRM	HMM (TS)	2.74	0.81	12	0.75482
Valhalla	HMM (TS)	4.40	34.32	6656	0.88978

In Table 5.17, we have the accuracy results for the Hof data set. As we recall from Table 5.7, our data set has longer segment lengths and greater timestamp differences than all previous data sets, while also having shorter tracks. The previous data sets were acquired using more professional GNSS

measuring methods, as represented by the median 1 s time difference. In contrast, our Hof data set consists of tracks recorded with smartphones, with a median time difference of 5 s and thus larger segments. As we can see in the matching results of the Hof data set, some tools that had significant issues with the previous high-sampled data sets perform much better with this more sparse one.

For example, OSRM achieves a comparably high level of accuracy while consuming very few CPU and memory resources. This time, it even outperforms GraphHopper and Valhalla. Fast Map Matching also has a comparably high accuracy, but needs a lot of time to read the UBODT again. Clearly, precomputing all possible routes in advance is not ideal for map matching, especially with data sets consisting of relatively few tracks. Barefoot is again in second place, but, as before, its resource consumption is impractical. Our Map Matching 2 solution again provides fast, competitive results and outperforms all other tools in terms of accuracy, especially with candidate adoption enabled.

When we use our trajectory simplification algorithm, Barefoot, Fast Map Matching, GraphHopper, and also Valhalla this time all improve in terms of accuracy and mostly also resource consumption, with only OSRM slightly degrading.

Candidate Adoption, Trajectory Simplification, and Q-learning

We conducted an additional benchmark on the Hof data set with our Map Matching 2 solution to further evaluate the impact of trajectory simplification and candidate adoption in combination. We also evaluated applying Q-learning on the MDP. Since Q-learning is an explorative learning algorithm, it takes longer and provides less accurate results. In our case, Q-learning uses a termination criterion that stops after a specific number of episodes with no improvement. However, Q-learning has the advantage of providing suboptimal results in a limited amount of time, for example, in real-time situations. This experiment demonstrates that our MDP design works with actual reinforcement learning algorithms, such as Q-learning.

Table 5.18: *Hof data set (DS-E) Map Matching 2 (MM2) matching mode comparison benchmark results* – Comparison of Markov decision process (MDP) and hidden Markov model (HMM) with and without candidate adoption (CA) and trajectory simplification (TS) for the matching modes of MM2 with multi-threading on the Hof data set. In addition, Q-learning (QL) is evaluated on the MDP. The elapsed time and CPU are in seconds, the main memory (MEM) is in MiB.

Software	Mode	Time	CPU	MEM	Accuracy
MM2	MDP	0.54	12.43	434	0.99651
MM2	MDP (TS)	1.62	30.24	742	0.98685
MM2	MDP (CA)	0.37	6.72	172	0.97803
MM2	MDP (CA, TS)	0.54	12.21	292	0.94238
MM2	HMM	0.49	11.64	302	0.98487
MM2	HMM (TS)	1.38	26.51	484	0.95925
MM2	HMM (CA)	0.38	6.56	154	0.95685
MM2	HMM (CA, TS)	0.52	11.68	248	0.87586
MM2	MDP (QL)	0.69	14.22	468	0.98714
MM2	MDP (QL, TS)	5.46	74.44	1082	0.97852
MM2	MDP (QL, CA)	0.39	7.10	170	0.97123
MM2	MDP (QL, CA, TS)	0.77	15.67	314	0.93572

As the results in Table 5.18 show, trajectory simplification improves both accuracy and resource consumption for our map matching solution in all scenarios. Additionally, we can observe the robustness of our MDP design compared to the HMM design, as the MDP design deteriorates less when candidate adoption and trajectory simplification are disabled. Furthermore, enabling candidate adoption while disabling trajectory simplification leads to a significant increase in computational resources. This is to be expected, as trajectory simplification reduces the point density of the track while candidate adoption increases the candidate density, especially in situations with high point-density. This illustrates how trajectory simplification and candidate adoption work in conjunction. When both are disabled, the time and CPU resources are similar to when both are enabled. However, as expected, the memory consumption is slightly higher when both are enabled due to the higher candidate density that candidate adoption generates. Combining trajectory simplification and candidate adoption results in a

much higher matching accuracy while having only a minor impact on the resource usage, especially concerning the CPU resources.

Table 5.19: *Hof data set (DS-E) Map Matching 2 (MM2) matching mode comparison metadata* – Comparison of Markov decision process (MDP) with Value Iteration and Q-Learning (QL) and hidden Markov model (HMM) with Viterbi Algorithm with and without candidate adoption (CA) and trajectory simplification (TS) for the matching modes of MM2 with multi-threading on the Hof data set. The trajectory simplification algorithm reduces the number of points from 14 651 to 5600. Therefore, fewer candidates are found with trajectory simplification enabled. The candidates are the sum of all possible found road positions for all points, the combinations are the number of shortest path routes and metrics that were computed between adjacent candidate pairs. For Q-learning, the average combinations of three runs was used.

Software	Mode	Candidates	Combinations
MM2	MDP	131 181	3 589 816
MM2	MDP (TS)	397 712	15 021 766
MM2	MDP (CA)	44 397	455 789
MM2	MDP (CA, TS)	123 048	1 408 940
MM2	HMM	131 181	3 254 383
MM2	HMM (TS)	397 712	13 956 790
MM2	HMM (CA)	44 397	354 205
MM2	HMM (CA, TS)	123 048	1 100 908
MM2	MDP (QL)	131 181	3 586 586
MM2	MDP (QL, TS)	397 712	15 020 487
MM2	MDP (QL, CA)	44 397	455 148
MM2	MDP (QL, CA, TS)	123 048	1 408 180

As shown in [Table 5.19](#), the number of candidates and combinations changes with the modes of trajectory simplification and candidate adoption on or off. Disabling trajectory simplification but leaving candidate adoption on dramatically increases the number of candidates and combinations, which explains the increase in computational resource usage. However, due to various performance optimizations, such as our multiple cache layers, the resource consumption does not increase proportionally by the same factor as the number of combinations that have to be evaluated. The older

HMM uses fewer combinations because the MDP design allows for ‘skips’ as explained in [subsection 3.1.6](#). This allows for more possibilities to match the track, slightly increasing the number of combinations. Additionally, we observe that our Q-learning algorithm is tuned to explore most, but not all, of the state space.

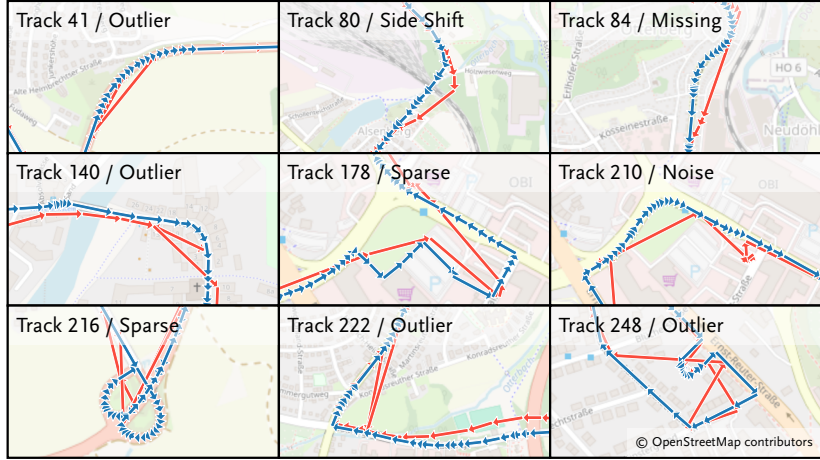


Figure 5.1: *Good map matching results* – All panels show in **red** the track and in **blue** the match from our Map Matching 2 (MM2) software. Each panel illustrates a challenging situation that our new approach successfully resolved. From left to right, top to bottom, we have the following situations: 1. large outlier; 2. shift to east; 3. unclear track shape with shift through missing data; 4. shift and outlier; 5. sparse track shape with cross movement; 6. noisy track shape with U-turn and cross movement; 7. sparse cross movement; 8. large outlier; 9. shifted track part with a large outlier.

To illustrate how our novel approach handles difficult situations, we present in [Figure 5.1](#) some examples from the Hof data set with the results from our Map Matching 2 application. As can be seen, our new approach is capable of finding robust solutions even in the presence of outliers, unclear track shapes, and track shifts.

As such, we can conclude that our novel map matching approach, using MDPs with Value Iteration, trajectory simplification, and candidate adoption in our Map Matching 2 reference implementation, considerably

outperforms existing approaches and implementations on different real-world data sets in various experiments, particularly with respect to achieved accuracy and resource usage.

5.5 Artificial Ground Truth Data Scenarios

The previous benchmark results on real-world data sets revealed several issues during the evaluation process. First, most data sets and tracks were hand-corrected rather than ground truth, which undermines the trustworthiness of the accuracy results. Additionally, the metadata of the data sets appears to impact most tools, e.g., the time differences, as well as the noise and outlier distributions.

We attempt to address and evaluate these issues by generating artificial ground truth map matching data sets according to our process described in [section 3.2](#). For each following experiment, we generate 1000 ground truth routes and their corresponding tracks with SUMO, all based on the road network from the Hof data set.

In [Table 5.20](#), we define multiple scenarios for applying artificial GNSS noise on the tracks. We evaluate a hypertuned scenario with parameters that resemble real-world floating car data, and we evaluate the impact of individual parameters by defining scenarios with only regular noise, additional outliers, sparse tracks, and missing points. With these specific scenarios, we can compare all evaluated map matching solutions in different situations.

5.5.1 Hypertuned Scenario

In [Figure 5.2](#), we can see the tracks of our hypertuned scenario in the same location as the real-world floating car data that we presented in [Figure 1.1](#). Although the individual tracks are different because they are randomly

Table 5.20: *Parameter definitions for our benchmark experiments* – We define as B the baseline experiment without applied error, as H the hypertuned scenario with a sampling interval distribution that is meant to resemble real floating car data, as E the typical error scenarios with L = low, M = mid, and H = high error, as O the outlier scenarios, and as S the sparse scenarios with 10 s to 30 s interval between two points. We define the error standard deviation e_σ , the smoothing window size s_w , the outlier probability o_p , the outlier standard deviation error o_σ , the missing points percentage m_p , and the time step delta δ .

Scenario		e_σ	s_w	o_p	o_σ	m_p	δ
Baseline	(B)	0.0	1				
Hypertuned	(H)	1.0	2				4 s
			2	2%	50 m	3%	5 s
			1				15 s
Error Low	(E_L)	1.0	10				1 s
Error Mid	(E_M)	2.0	20				1 s
Error High	(E_H)	3.0	30				1 s
Outlier Low	(O_L)	1.0	10	1%	30 m		1 s
Outlier Mid	(O_M)	2.0	20	2%	40 m		1 s
Outlier High	(O_H)	3.0	30	3%	50 m		1 s
Sparse Low	(S_{10})	1.0	1				10 s
Sparse Mid	(S_{20})	2.0	1				20 s
Sparse High	(S_{30})	3.0	1				30 s
Missing Low	(M_L)	1.0	2			10%	5 s
Missing Mid	(M_M)	2.0	2			20%	5 s
Missing High	(M_H)	3.0	2			30%	5 s

generated, the noise distribution around the roads, some outliers, and some map-crossing track segments resemble the real floating car data. After map matching the artificial tracks, a float map of the simulated floating car data can be generated, similar to the float map of the real floating car data. This demonstrates that artificially generated floating car data can resemble real floating car data to some extent.

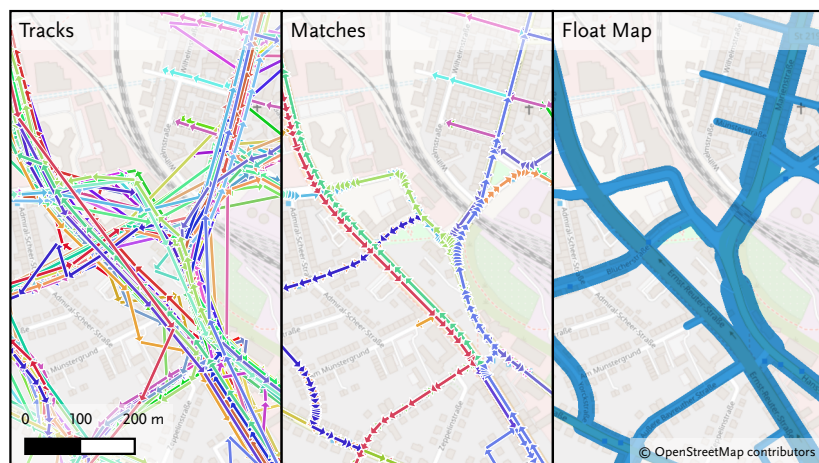


Figure 5.2: *Artificial hypertuned floating car data float map* – Similar to [Figure 1.1](#), the left panel shows the original tracks, in this case, with artificial noise applied to the artificially generated ground truth routes, the middle panel shows the matches of our Map Matching 2 (MM2) software, and the right panel shows a float map generated from these matches.

In [Table 5.21](#), we can see the results of the hypertuned scenario. When comparing these results to the real-world reference in [Table 5.17](#) (i.e., the Hof data set), we can see that the ranking of the tools is quite similar, although not exactly the same. Again, our Map Matching 2 solution is the most accurate in comparison, with Barefoot closely following. Barefoot has even higher accuracy than in the real-world reference, but again uses a huge amount of resources. Fast Map Matching has a very similar level of accuracy in both modes. GraphHopper, OSRM, and Valhalla all produce slightly better results in the hypertuned scenario. GraphHopper and Valhalla swapped their rank compared to the real-world reference, although they are still close. Since the results of the hypertuned scenario are so remarkably similar to the real-world results of the Hof data set, we can conclude that it seems, in fact, possible, to a certain extent, to use artificially generated map matching data for evaluating map matching algorithms reasonably.

Table 5.21: *Artificial hypertuned (H) matching benchmark results* – Comparison of the elapsed time in seconds, CPU seconds, and main memory (MEM) MiB resource usage for the matching phase with multi-threading on the hypertuned (H) scenario. Abbreviations: Fast Map Matching (FMM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM), Hidden Markov model (HMM), ST-Matching (STM), Markov decision process (MDP), Candidate adoption (CA)

Software	Mode	Time	CPU	MEM	Accuracy
Barefoot	HMM	34.07	3769.41	21625	0.97023
FMM	HMM	16.54	28.18	1418	0.71873
FMM	STM	0.66	24.69	66	0.50862
GraphHopper	HMM	9.56	270.17	1569	0.78698
MM2	MDP	1.16	83.42	1417	0.99296
MM2	MDP (CA)	0.86	34.77	535	0.97532
MM2	HMM	1.09	73.57	879	0.98513
MM2	HMM (CA)	0.83	34.36	491	0.94523
OSRM	HMM	6.51	3.61	12	0.86597
Valhalla	HMM	9.32	72.78	7406	0.77597

We now use this method to evaluate the performance of the different map matching applications when specific noise parameters are simulated. We hypothesize that these experiments will reveal some strengths and weaknesses of the different tools, depending on the noise characteristics.

5.5.2 Baseline Tracks

In contrast to the hypertuned scenario above, which is meant to be compared to the real-world results from the Hof data set, all of the following scenarios are meant to be compared against an artificially simulated baseline. This baseline is created by sampling the tracks from the ground truth at 1 s time intervals without applying any noise. Consequently, all points from the baseline tracks lie exactly on the corresponding ground truth routes. By map matching the baseline scenario, we can determine the maximum accuracy each software can achieve with a 1 s track when no noise has been applied.

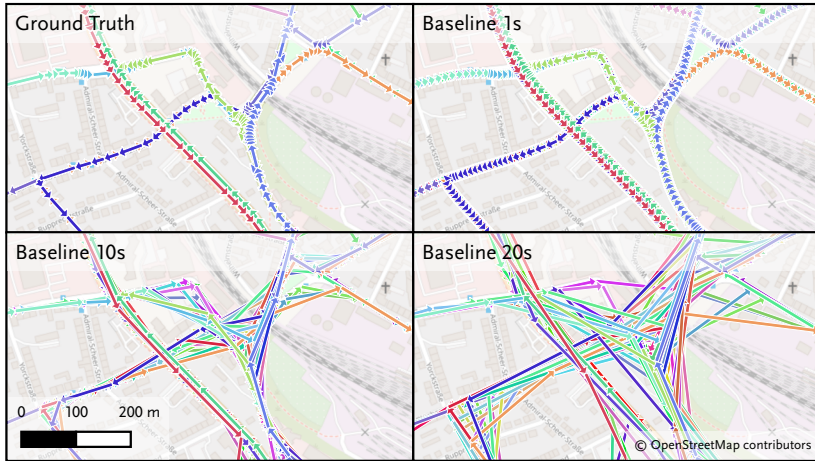


Figure 5.3: *Artificial noise baseline (B) scenarios* – The top left panel shows the ground truth. The top right panel shows the 1 s interval baselines (B_1), the bottom left panel shows the 10 s interval baselines (B_{10}), and the bottom right panel shows the 20 s baselines (B_{20}). The additional 30 s baselines (B_{30}) have been omitted from this visualization due to space restrictions.

As we observed when evaluating the real-world data sets in [section 5.4](#), the time interval significantly impacts the accuracy of some map matching tools. This is unfortunate because the sampling interval while recording GNSS tracks can usually be configured relatively freely, and can even be adjusted dynamically by the recording application, for example, depending on speed. As seen in [Table 5.7](#), real-world data sets actually contain different sampling intervals. For this reason, we also evaluate artificially simulated sparse baseline scenarios with time intervals of 10 s, 20 s, and 30 s. This allows us to gain insight into how the maximum achievable accuracy of each software changes with track sparseness.

In [Figure 5.3](#), we can see the simulated ground truth tracks and the baselines B_1 with 1 s interval, B_{10} with 10 s interval, and B_{20} with 20 s interval. Due to space restrictions, the B_{30} baselines with 30 s interval have been omitted from this figure. The ground truth uses the same road edge segments as the underlying road network, which is why curves are approximated with more segments than straight roads. As can be seen in the baselines, all *points*,

i.e., the endpoints of the arrows, lie exactly on the ground truth routes. When connecting these points with segment lines, they may visually cross the roads, especially when a track is more sparse. This shows that B_1 is closer to the original ground truth than the sparser baselines. It can also be seen that the B_{10} baselines have only every 10th point of the B_1 baselines, and the B_{20} baselines have only every 2nd point of the B_{10} baselines. The last point of the sparse baselines is always kept, even when the actual time interval is below the configured sparse time interval. For example, if the last point is 5 s away from the previous point in B_{10} , both points are kept. This ensures that the track does not cut off early. We will explain this in more detail shortly.

Table 5.22: *Artificial sparse baseline (B) matching benchmark results* – Comparison of the accuracy of the matching phase on the baseline scenarios (B) with the default baselines B_1 with 1 s time interval, the sparse baselines B_{10} with 10 s time interval, the sparse baselines B_{20} with 20 s time interval, and the sparse baselines B_{30} with 30 s time interval. Abbreviations: Fast Map Matching (FMM), ST-Matching (STM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM)

Software	B_1	B_{10}	B_{20}	B_{30}
Barefoot	0.99612	0.99603	0.99067	0.98092
FMM	0.99705	0.99191	0.98090	0.96075
STM	0.58659	0.95838	0.98129	0.95932
GraphHopper	0.97592	0.97959	0.97890	0.97525
MM2	0.99668	0.99501	0.98789	0.97599
OSRM	0.85473	0.95200	0.94923	0.95388
Valhalla	0.90783	0.96603	0.97377	0.96968

After matching the baselines, we can see in Table 5.22 that Barefoot, Fast Map Matching, GraphHopper, and our Map Matching 2 software are relatively robust, regardless of the sampling interval. However, they degrade as the track becomes sparser, since fewer points contain less information about the exact movement. In contrast, Fast Map Matching in its ST-Matching mode, OSRM, and Valhalla show significant improvement with sparser baseline tracks, especially in B_{20} (with OSRM peaking in B_{30}), compared to the dense B_1 baselines.

There are several reasons why no application can achieve 100% accuracy. First, discrepancies may already emerge from the different road network representations in different applications. For example, SUMO may use different OpenStreetMap tags than the individual open source software map matching tools when importing the road network. Consequently, the ground truth routes may run along edges that are not imported into the road network representation of another map matching application, even though the same original road network data file is used for all tools.

Furthermore, the last point of the ground truth route and the last point of the tracks are not identical. The floating car data tracks are sampled from the actually simulated movements at 1 s intervals, whereas the ground truth is extracted from the planned routes. Consequently, when the simulated movement does not end on a whole second, the remainder between the most recently sampled position and the reaching of the final position in the planned route is discarded because the movement ends before another position is sampled. As a result, every track affected by this circumstance has a discrepancy between the last sampled point and the final ground truth route point. Accordingly, since the baseline and all noise scenario tracks are based on the sampled floating car data, they inherit this limitation. The same is true for the sparse baseline tracks, which keep the last sampled track point, even if it is reached below the configured sampling interval. Without this exception, the accuracy of map matching the sparse baseline tracks would degrade even further, simply from the final segments not ending where the sampled tracks end. Keeping this final point ensures that all baseline tracks start and end in the same position. Nevertheless, all baseline and noise scenario tracks retain the discrepancy to the ground truth, although the discrepancy is always < 1 s in sum.

For all these reasons, it is virtually impossible to achieve 100% accuracy. However, since these discrepancies are realistic, we keep them. In practice, real-world floating car data can also originate from a road network different from the one a map matching application later imports. Additionally, the floating car data sampling can stop before the actual movement of the device stops. Therefore, it is reasonable to keep these limitations and discrepancies with our artificial map matching data generation process in order to gather realistic insights.

5.5.3 Error Scenario

After having introduced the baselines, we proceed with the actual scenarios with different noise characteristics. In the first scenario, the error scenario, we simulated autocorrelated Ornstein-Uhlenbeck process noise on high-sampled tracks to evaluate how shifting the track positions away from the roads impacts the map matching accuracy.



Figure 5.4: *Artificial noise error (E) scenarios* – The top left panel shows the ground truth. The top right panel shows low error noise applied (E_L), the bottom left panel shows middle error noise applied (E_M), and the bottom right panel shows high error noise applied (E_H).

In [Figure 5.4](#), we show how the ground truth compares to the tracks after low (E_L), mid (E_M), and high (E_H) artificial noise is applied, according to the definitions in [Table 5.20](#). As we can see, the low error noise scenario looks more realistic than the mid and high error noise scenarios. We intentionally apply strong noise error to stress the map matching implementations with increasingly difficult scenarios. The hypothesis is that this allows for a clearer ranking and degradation regression analysis of the evaluated map matching tools.

Table 5.23: *Artificial noise error (E) matching benchmark results* – Comparison of the accuracy for the matching phase on the noise error scenarios (E) with low (E_L), mid (E_M), and high (E_H) error in comparison to the baseline scenario B_1 . Abbreviations: Fast Map Matching (FMM), ST-Matching (STM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM)

Software	B_1	E_L	E_M	E_H
Barefoot	0.99612	0.99519	0.97607	0.92400
FMM	0.99705	0.53235	0.36836	0.29577
STM	0.58659	0.27324	0.16981	0.13235
GraphHopper	0.97592	0.62869	0.31098	0.22977
MM2	0.99668	0.99608	0.99336	0.98246
OSRM	0.85473	0.73002	0.29241	0.15727
Valhalla	0.90783	0.90636	0.89536	0.83038

As expected, applying artificial Ornstein-Uhlenbeck noise leads to a decrease in accuracy in all map matching tools, as we can see in [Table 5.23](#). Even with a low error, Fast Map Matching with HMMs, Fast Map Matching with ST-Matching, GraphHopper, and OSRM degrade drastically, which worsens with larger noise. Valhalla does not degrade as much with applied noise, but struggles with the high sampling interval. Among the third-party tools, Barefoot is the only one that achieves reasonably good accuracy. However, our own Map Matching 2 approach is much more robust than Barefoot with larger errors.

5.5.4 Outlier Scenario

In [Figure 5.5](#), we added different amounts of outliers to the previously added noise error, ranging from a few small outliers to many large outliers. The outliers were taken from an additional Gaussian noise distribution and are intended to stress the map matching algorithms. The outliers are clustered using a closeness parameter to simulate the behavior of satellite opacity occurring in batches in practice. In the real-world floating car data in [Figure 1.1](#), we can see similar outliers in the tracks. Our hypothesis is

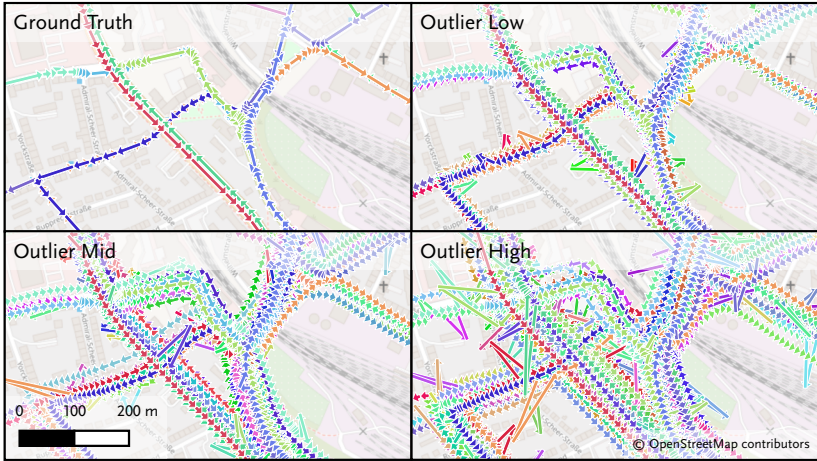


Figure 5.5: *Artificial outlier error (O) scenarios* – The top left panel shows the ground truth. The top right panel shows low error noise with a few small outliers applied (O_L), the bottom left panel shows middle error noise with average outliers applied (O_M), and the bottom right panel shows high error noise with many large outliers applied (O_H).

Table 5.24: *Artificial outlier error (O) matching benchmark results* – Comparison of the accuracy for the matching phase on the noise error scenarios (O) with low noise and few outliers (O_L), mid noise and average outliers (O_M), and high noise with large outliers (O_H) in comparison to the baseline scenario B_1 . Abbreviations: Fast Map Matching (FMM), ST-Matching (STM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM)

Software	B_1	O_L	O_M	O_H
Barefoot	0.99612	0.97226	0.90387	0.78587
FMM	0.99705	0.45913	0.28989	0.21938
STM	0.58659	0.21981	0.12209	0.08453
GraphHopper	0.97592	0.51291	0.22941	0.17626
MM2	0.99668	0.99599	0.98516	0.95021
OSRM	0.85473	0.59476	0.14829	0.07484
Valhalla	0.90783	0.57459	0.25904	0.13643

that our novel Map Matching 2 approach is more robust to such outliers, which was one of our main goals in developing our solution.

The results in Table 5.24 show that most existing map matching solutions are unable to handle outliers. Even a few small outliers dramatically degrade the accuracy of all tools, except for Barefoot and our own Map Matching 2 approach. With more and larger outliers, even Barefoot struggles, while our Map Matching 2 approach is the only in comparison that remains comparatively accurate, even in very difficult situations with heavily degraded tracks.

5.5.5 Sparse Scenario

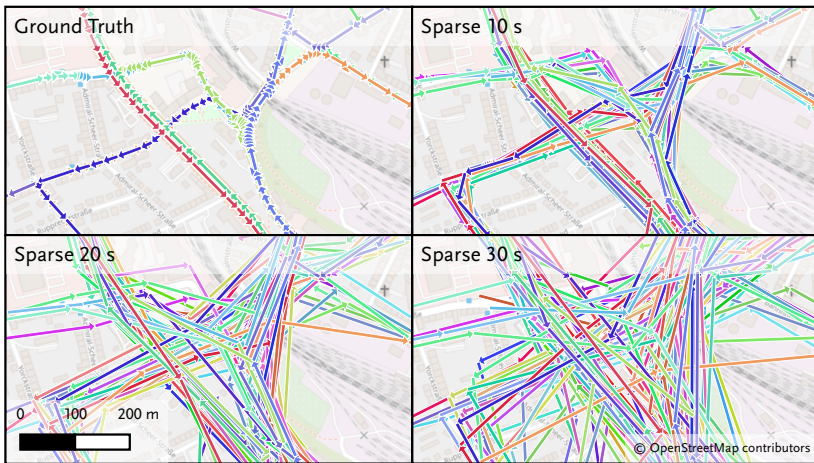


Figure 5.6: *Artificial sparse error (S) scenarios* – The top left panel shows the ground truth. The top right panel shows 10 s time difference applied (S_{10}), the bottom left panel shows 20 s time difference applied (S_{20}), and the bottom right panel shows 30 s time difference applied (S_{30}).

Similar to the previously described sparse baseline tracks, we generated the sparse noise tracks shown in Figure 5.6, however, with additional noise error applied, as in the error scenario shown in Figure 5.4. As with the baselines with 20 s, the 20 s and especially the 30 s sampling intervals appear erratic on the map. However, this is once again merely a visual artifact resulting from the long distances between successive segment points crossing the

map. The sparse points of the tracks remain close to the original ground truth, similar to the noise error scenarios in [Figure 5.4](#).

Table 5.25: *Artificial sparse (S) matching benchmark results* – Comparison of the accuracy for the matching phase on the sparse scenarios (S) with 10 s time difference (S_{10}), 20 s time difference (S_{20}), and 30 s time difference (S_{30}) in comparison to the best baseline scenario B^{max} from the best results of B_1 , B_{10} , B_{20} , or B_{30} , as shown in [Table 5.22](#). Abbreviations: Fast Map Matching (FMM), ST-Matching (STM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM)

Software	B^{max}	S_{10}	S_{20}	S_{30}
Barefoot	0.99612	0.99261	0.94376	0.86661
FMM	0.99191	0.82629	0.71922	0.64823
STM	0.98129	0.68049	0.60150	0.53910
GraphHopper	0.97959	0.88709	0.74959	0.72207
MM2	0.99668	0.99352	0.98117	0.95215
OSRM	0.95388	0.93134	0.71786	0.47783
Valhalla	0.97377	0.96209	0.96728	0.90995

As expected, the sparse scenarios produced interesting results. As shown in [Table 5.25](#), the tools that performed poorly in the high sampling scenarios now perform much better with a lower sampling rate, despite the noise error remaining the same as in [Table 5.23](#). For this reason, we use the best baseline accuracy from [Table 5.22](#) in B^{max} as the upper accuracy limit for comparing the sparse scenarios. Of course, it is also possible to directly compare the sparse scenarios with the corresponding baselines in [Table 5.22](#). Interestingly, Valhalla even outperforms the previously more robust Barefoot with lower sampling intervals of 20 s and 30 s. However, as expected, our Map Matching 2 approach is again the most robust in all situations, staying closest to the corresponding baselines from [Table 5.22](#).

5.5.6 Missing Points Scenario

In the last experiment, we evaluated the impact of missing points on the matching accuracy. In the real-world floating car data in [Figure 1.1](#), we can see some segments crisscrossing the map. This occurs when one or



Figure 5.7: *Artificial missing points error (M) scenarios* – The top left panel shows the ground truth. The top right panel shows few missing points applied (M_L), the bottom left panel shows average missing points applied (M_M), and the bottom right panel shows many missing points applied (M_H).

more points are missing, e.g., due to failure of locking a GNSS position. In these cases, the sampling interval between successive points is larger than on average. For this experiment, we use a default sampling interval of 5 s so that successive missing points have a greater impact. Missing points are also clustered with a closeness parameter similar to that used for outliers, for the same reasons. Since this approach leads to random time intervals, we again use B^{max} for comparison. In [Figure 5.7](#), we can see the crisscrossing segments that result from removing points from the tracks, i.e., applying missing points.

In [Table 5.26](#), we can see how missing points impact the accuracy of the different map matching tools. Although a lower sampling interval improves the accuracy of some tools, sporadically missing points negatively impact the accuracy much more dramatically than a fixed, lower sampling interval. Barefoot and Valhalla still produce good results, even with many missing points, although Barefoot handles this better than Valhalla. However, only our Map Matching 2 solution remains robust in all scenarios.

Table 5.26: *Artificial missing points (M) matching benchmark results* – Comparison of the accuracy for the matching phase on the missing points scenarios (M) with a 5 s sampling interval, with few (M_L), average (E_M), and many (M_H) missing points, compared to the best baseline scenario B^{max} . Abbreviations: Fast Map Matching (FMM), ST-Matching (STM), Map Matching 2 (MM2), Open Source Routing Machine (OSRM)

Software	B^{max}	M_L	M_M	M_H
Barefoot	0.99612	0.99301	0.96130	0.90516
FMM	0.99191	0.76809	0.58727	0.49056
STM	0.98129	0.58153	0.39016	0.33304
GraphHopper	0.97959	0.84306	0.55155	0.49332
MM2	0.99668	0.99317	0.98540	0.97517
OSRM	0.95388	0.85556	0.34464	0.15649
Valhalla	0.97377	0.93135	0.92986	0.85655

In conclusion, we can confirm that our novel Map Matching 2 application is highly accurate and robust in all evaluated scenarios, most often achieving first place, especially with larger and more difficult errors. Barefoot is second overall, but it uses many resources for achieving this level of accuracy, even without candidate adoption. All other map matching implementations have their own strengths and weaknesses, succeeding in one scenario but failing in another. The different scenarios provide more insight into these situations. For example, GraphHopper works well on tracks with few errors, OSRM works well on sparse tracks with few errors, Valhalla works well on tracks with more errors and missing points, but not well on tracks with outliers, and Fast Map Matching works best on sparse tracks with few errors. When the noise characteristics of the data to be matched are uncertain, our novel Map Matching 2 solution is robust and accurate in all evaluated scenarios. Therefore, it may be the most practical solution.

6 Discussion

In this work, we presented our novel map matching approach with our novel Markov decision process (MDP) replacing the traditional hidden Markov model (HMM), based on our newly developed metric, our improved trajectory simplification algorithm, and our new candidate adoption feature. Our new approach provides a more robust and accurate solution to the offline map matching problem than existing methods.

To support this statement, we conducted an extensive benchmark on five real-world map matching data sets and newly generated, artificial map matching data sets with various noise characteristics. We compared our own reference implementation Map Matching 2 with five existing open source software map matching implementations.

To measure the accuracy of all solutions, we developed a new comparison algorithm implementing existing metrics that can extract the similarities and differences between any two routes within the same road network, even with slight rounding differences in the coordinates.

The evaluation of the benchmark results showed that our new approach generally outperforms existing solutions and is more robust in difficult map matching situations. We provide our new solution as ready-to-use open source software application. It contains many algorithms and optimizations not only specific to map matching, but also serves as a resource for interested developers and future researchers.

6.1 Answers to Research Questions

In this work, we sought answers to the research questions defined in [section 1.2](#). The following are our findings:

1. The MDP, compared to the traditional HMM, the trajectory simplification algorithm, and the candidate adoption feature, all individually and especially in combination, make map matching more robust in challenging situations, especially with noise and outliers.
2. The stochastic process (i.e., HMM or MDP) can be made more robust by defining a metric that penalizes U-turns and round trips. We have demonstrated that a metric incorporating direction changes of the computed routes, combined with candidate adoption, can leverage the stochastic process to collapse outliers, thus making the process more robust.
3. With trajectory simplification, there are algorithms that remove noise within defined boundaries. Thus, there is a deterministic method for resolving noise algorithmically as a preprocessing step before the stochastic process. However, not all map matching algorithms benefit equally from trajectory simplification.
4. Detours can be stochastically penalized by decreasing the value of routes with many direction changes. This approach differs from fixed U-turn penalties, which require a hard limit to distinguish U-turns from regular turns. By scoring every direction change, U-turns are penalized more than regular turns, without the need for hard limits.
5. To some extent, it is possible to generate artificial map matching ground truth data. While real-world global navigation satellite system (GNSS) noise is difficult to simulate due to real-world physics, assumptions about the error distance distributions and autocorrelation allow for the Ornstein-Uhlenbeck process to generate artificial noise with similar characteristics. When combined with additional methods that simulate outliers and failed or missing GNSS measurements, comparatively realistic artificial map matching ground truth data can be generated. This method can also generate scenarios with errors beyond regular GNSS errors for evaluation and stress-testing purposes.

6.2 General Applicability of Solutions

In this work, we provide an improved, comprehensive solution to the offline map matching research topic. We have also developed many modular algorithms, models, and optimizations that can be used not only in map matching, but also in related fields, as described in [section 4.4](#). For example, our evaluations of different algorithms can inform decisions regarding the selection of routing algorithms, routing distance limits, road network topology simplification, or candidate search algorithms for map matching. Thus, our evaluations offer valuable insights on how to improve existing and new map matching solutions.

Furthermore, our artificial ground truth map matching data generation process, as described in [section 3.2](#), can create various GNSS noise scenarios for further evaluation. Additionally, our Map Matching 2¹⁹ application can be directly used in new research, such as in map matching comparison reviews, map matching use case analyses, or other related geographic information science research domains. There are many more ideas and findings from this work that can be adapted to existing or new map matching solutions and research. We will present some of them in the next section.

With the improved accuracy of our novel map matching solution, several offline map matching use cases, as discussed in [section 1.7](#), become more reliable. For example, identifying road network errors or analyzing road network usage (i.e., float maps) for planning road maintenance or public transportation is much more reliable with highly accurate map matching. There may be additional use cases not covered in this work that could benefit from highly accurate, robust map matching as well.

6.3 Limitations and Future Research

Of course, many research topics remain for future work. For example, we only discussed the offline map matching approach, but online map matching can also benefit from more accurate positioning. However, online map matching typically has access to additional sensors, such as momentary speed or acceleration, which would require the inclusion of additional terms in our metrics. Nevertheless, our very fast map matching algorithm may already be well suited for real-time use cases when the historical amount of live-tracking points and corresponding candidates is limited. Further tuning of the MDP, such as using smaller discount factors in Value Iteration to adjust how much later states influence earlier states while reusing the state space when new points are measured, and applying real-time limits to the Q-learning algorithm, has a lot of potential for achieving very fast and accurate real-time online map matching. If map matching should be performed via web service, our Map Matching 2 reference implementation is already prepared for this purpose. Internally, imported tracks are passed to a processing queue, and the matches are returned asynchronously. This internal processing design facilitates the direct attachment of a web- or network service.

Moreover, the current reinforcement learning algorithms that we use are rather basic. We mainly use ‘Value Iteration’, although we have also implemented ‘Policy Iteration’ (which produces equal accuracy in similar runtime) and we evaluated ‘Q-learning’. One idea is to use more sophisticated algorithms, such as n-step planning algorithms or SARSA- λ algorithms, to guide the learning agent to visit more promising states instead of random ones. This could further improve the performance of the learning algorithms compared to the dynamic programming algorithms that visit all states. It can also allow for larger candidate and combination counts to find even more accurate results while maintaining low runtime performance by visiting only promising states.

Furthermore, our MDP design currently only works on single tracks. The advantage of this design is that our map matching solutions works on individual tracks without requiring a training phase. However, this also means

that learned behavior cannot be transferred to other tracks. Reinforcement learning generally facilitates the transfer of learned behavior, since the learning algorithms create a knowledge-base of the MDP outside its scope. With a MDP design that works on multiple tracks, learned behavior can be transferred between tracks. There are various ideas on how a MDP can be designed to support this behavior. One approach is to design the MDP to contain spatial locations, such as in a tiled grid system or with spatial radial basis functions. If segments spanning different parts of the map are consistently matched in a specific way, this information can be applied to tracks with missing data or outliers to infer knowledge from previously matched tracks. Such a general MDP may also be combined with a specific MDP, so that there exists a global learned behavior that exchanges its knowledge with individual tracks.

Another limitation of our work is our focus on car movements, i.e., vehicular GNSS tracks. The main reason for this is that all existing map matching data sets, as well as floating car data, originate from individual automobile movements. Of course, map matching can also be performed for bicycles, pedestrians, public transportation, and other means of transportation on road, river, or train networks. Although our map matching approach focuses on car movements, our Map Matching 2 application supports arbitrary road networks by allowing manual specification of the desired OpenStreetMap tags to import. Combined with specifying different weights for our metrics, our solution can technically match any GNSS recorded track because GNSS errors generally originate from the same physical principles, regardless of the means of transportation. Nevertheless, different means of transportation can present specific challenges. For example, public transportation matching can benefit from the time schedules for estimating the candidate positions, and bicycles usually take routes that avoid mountains. However, such specifics play a greater role when the sampling rate is low, or when there are large outliers or long periods of missing data in the tracks. As such, our map matching solution may already provide reasonably accurate results in these use cases.

Another research topic that we briefly discussed in [section 1.6](#) is map matching when the map is wrong. In such situations, our solution does not introduce new roads because we argue that the maps are generally more

accurate and complete than the recorded GNSS tracks. However, since our MDP computes the rewards for all states, it is technically possible to detect track segments with poor ‘outlier’ rewards and make corrections, such as discarding the corresponding route and replacing it with a direct connection. This could also be incorporated into the metrics so that it is based on a last-resort penalty rather than a hard limit, similar to the ‘skips’ that are currently implemented. Additionally, there is the use case of matching in a shared space without explicit roads, such as a plaza or park intended for pedestrian use. In this case, incorporating a spatial line aggregation algorithm or a heat map functionality could provide insight into larger shared spaces for determining routes.

Another area for improvement is incorporating time information into our map matching metrics. Currently, our metrics are based solely on geometric computations. The advantage of this approach is that any geospatial track can be matched, even when no time information is provided. For example, this can be useful when a track is provided as well-known text (WKT) linestring. However, this approach limits the incorporation of speed metrics into the stochastic process that depend on the timestamps of the GNSS points. Our Map Matching 2 solution already keeps the timestamps throughout the entire process, even with trajectory simplification, but this information is only used in preparatory steps, such as the ‘warp detection’ feature, as described in [subsection 3.1.1](#). Similarly to the ‘warp detection’ feature, a ‘staypoint detection’ feature could be implemented, so that staypoints collapsed by the median-merge algorithm receive a marker depending on the number of points merged. Having a higher weight for a staypoint position could make it more likely to be traversed than if there were no weights for any points. Furthermore, incorporating time information into the metrics could also further improve the matching accuracy. For example, since OpenStreetMap data often contains road speeds, and our Map Matching 2 reference implementation already processes these tags (i.e., they are available during the metrics computation), it is technically possible to add speed computations to our metrics, though this would require re-tuning the metric weights.

Although our map matching approach is already highly accurate, we want to demonstrate some edge cases in which our approach did not produce

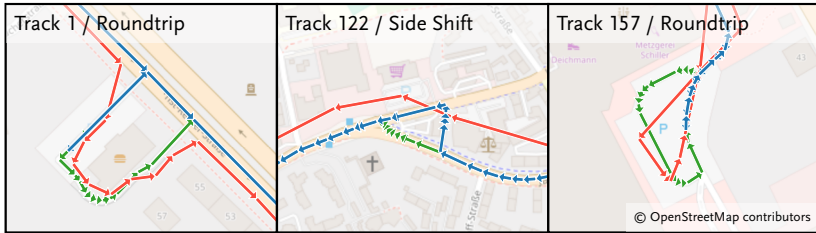


Figure 6.1: *Bad matches* – All panels show in red the track, in blue the match from our Map Matching 2 software, and in green the correct solution for the wrong part. In the left panel, our software does not follow the track around a place, but instead makes a U-turn. In the middle panel, the track is strongly shifted to the north, causing our software to take a suboptimal detour instead of staying on the main road. And in the right panel, although the track is greatly shortened, our software again takes a U-turn instead of following the detour around the parking lot.

optimal results. In [Figure 6.1](#) we show three cases of non-optimal matching behavior. While it is difficult to estimate the correct solution, since the provided solutions are not ground truth, the human eye would likely match the tracks differently in these situations. Nevertheless, such errors are rare, and our approach generally matches many difficult situations with remarkable accuracy, as was shown in [Figure 5.1](#).

6.4 Conclusion

In conclusion, we made significant progress in the field of map matching research by exploring innovative, novel solutions to the ongoing challenge of achieving 100% accuracy. We introduced a novel map matching approach, conducted an extensive benchmark of our new solution against existing ones, reviewed many optimizations for improving map matching performance, and developed a system for generating artificial map matching ground truth data, so that future research can evaluate and develop more sophisticated solutions.

Despite our advances in this domain, there is still optimization potential and room for improvement in map matching, to achieve even higher accuracy and computational performance than we have today. We hope our findings and groundwork will serve as a foundation for many future advances in map matching to follow.

Bibliography

- Akiba, Takuya et al. (July 2019). “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’19. New York, NY, USA: ACM, pp. 2623–2631. doi: [10.1145/3292500.3330701](https://doi.org/10.1145/3292500.3330701) (cit. on p. 77).
- Alt, Helmut et al. (Nov. 2003). “Matching planar maps”. In: *Journal of Algorithms* 49.2, pp. 262–283. doi: [10.1016/s0196-6774\(03\)00085-3](https://doi.org/10.1016/s0196-6774(03)00085-3) (cit. on p. 26).
- Andoyer, Henri (Apr. 1932). “Formule Donnant la Longueur de la Géodésique Joignant 2 Points de L’ellipsoïde Donnés Par Leurs Coordonnées Géographiques”. In: *Notices Scientifiques* 34.1, pp. 77–81. doi: [10.1007/bf03030136](https://doi.org/10.1007/bf03030136) (cit. on p. 106).
- Ayfantopoulou, Georgia et al. (Oct. 2022). “Improving Map Matching of Floating Car Data with Artificial Intelligence Techniques”. In: *Information* 13.11, p. 508. doi: [10.3390/info13110508](https://doi.org/10.3390/info13110508) (cit. on pp. 30, 38).
- Baza-Solares, Nelson et al. (Apr. 2022). “Traffic Simulation with Open-Source and Commercial Traffic Microsimulators: A Case Study”. In: *Communications - Scientific letters of the University of Zilina* 24.2, E49–E62. doi: [10.26552/com.c.2022.2.e49-e62](https://doi.org/10.26552/com.c.2022.2.e49-e62) (cit. on p. 43).
- Bernstein, David and Kornhauser, Alain L. (Aug. 1996). “An Introduction to Map Matching for Personal Navigation Assistants”. In: *US Transportation Collection*. URL: <https://rosap.ntl.bts.gov/view/dot/38257> (cit. on pp. 26, 31).
- Boeing, Geoff (Sept. 2017). “OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks”. In: *Computers, Environment and Urban Systems* 65, pp. 126–139. doi: [10.1016/j.compenvurbsys.2017.05.004](https://doi.org/10.1016/j.compenvurbsys.2017.05.004) (cit. on p. 33).

- Brakatsoulas, Sotiris et al. (Aug. 2005). “On Map-Matching Vehicle Tracking Data”. In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, pp. 853–864. URL: <https://dl.acm.org/doi/10.5555/1083592.1083691> (cit. on p. 26).
- CartONG (Mar. 2019). *Assessment of the GPS accuracy of mobile devices*. Tech. rep. United Nations High Commissioner For Refugees (UNHCR), p. 15. URL: <https://alnap.org/help-library/resources/assessment-of-the-gps-accuracy-of-mobile-devices/> (cit. on p. 22).
- Chao, Pingfu et al. (Jan. 2020). “A Survey on Map-Matching Algorithms”. In: *Databases Theory and Applications*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 121–133. ISBN: 978-3-030-39469-1. DOI: [10.1007/978-3-030-39469-1_10](https://doi.org/10.1007/978-3-030-39469-1_10) (cit. on pp. 28, 30, 44).
- Charanaraj, N. et al. (July 2019). “Web Information System for GPS Vehicle Tracking, Telematics, Communication and Advanced Fleet Management System”. In: *International Journal of Research in Engineering, Science and Management* 2.7, pp. 173–175. URL: https://www.ijresm.com/Vol.2_2019/Vol2_Iss7_July19/IJRESM_V2_I7_43.pdf (cit. on p. 13).
- Che, Mingliang et al. (May 2018). “An Enhanced Hidden Markov Map Matching Model for Floating Car Data”. In: *Sensors (Basel)* 18.6, pp. 1758–1777. DOI: [10.3390/s18061758](https://doi.org/10.3390/s18061758) (cit. on p. 31).
- Chen, Luo et al. (Sept. 2021). “Fast Map Matching with Vertex-Monotone Fréchet Distance”. In: *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*. Ed. by Matthias Müller-Hannemann and Federico Perea. Vol. 96. Open Access Series in Informatics (OASIs). Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:20. DOI: [10.4230/OASIs.ATMOS.2021.10](https://doi.org/10.4230/OASIs.ATMOS.2021.10) (cit. on p. 27).
- Codeca, Lara et al. (June 2022). “SAGA: An Activity-based Multi-modal Mobility Scenario Generator for SUMO”. In: *SUMO Conference Proceedings* 1, pp. 39–58. DOI: [10.52825/scp.v1i.99](https://doi.org/10.52825/scp.v1i.99) (cit. on p. 43).
- Dietterich, Thomas G. (Aug. 2002). “Machine Learning for Sequential Data: A Review”. In: *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–30. DOI: [10.1007/3-540-70659-3_2](https://doi.org/10.1007/3-540-70659-3_2) (cit. on p. 28).

- Diggelen, Frank van and Enge, Per (Sept. 2015). "The World's first GPS MOOC and Worldwide Laboratory using Smartphones". In: *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation*. ION GNSS+ 2015. Tampa, Florida: Tampa Convention Center, pp. 361–369. URL: <https://www.ion.org/publications/abstract.cfm?articleID=13079> (cit. on p. 22).
- Dijkstra, Edsger Wybe (Jan. 1959). "A note on two problems in connexion with graphs". In: *Numerische Mathematik*. Vol. 1. Springer, pp. 269–271. DOI: [10.1007/BF01386390](https://doi.org/10.1007/BF01386390) (cit. on p. 32).
- Dima, Corina and Popa, Cosmin (Dec. 2023). "A review of map-matching algorithms". In: *Annals of the "Dunarea de Jos" University of Galati. Fascicle II, Mathematics, Physics, Theoretical Mechanics* 2.46, pp. 121–125. DOI: [10.35219/ann-ugal-math-phys-mec.2023.2.09](https://doi.org/10.35219/ann-ugal-math-phys-mec.2023.2.09) (cit. on p. 30).
- Douglas, David H. and Peucker, Thomas K. (Oct. 1973). "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2, pp. 112–122. DOI: [10.3138/fm57-6770-u75u-7727](https://doi.org/10.3138/fm57-6770-u75u-7727) (cit. on p. 34).
- Ebisch, Konrad (Oct. 2002). "A correction to the Douglas–Peucker line generalization algorithm". In: *Computers & Geosciences* 28.8, pp. 995–997. DOI: [10.1016/S0098-3004\(02\)00009-2](https://doi.org/10.1016/S0098-3004(02)00009-2) (cit. on p. 34).
- Eiter, Thomas and Mannila, Heikki (May 1994). *Computing Discrete Frechet Distance*. Tech. rep. Vienna, Austria: Christian Doppler Laboratory for Expert Systems. URL: <https://www.kr.tuwien.ac.at/staff/eiter/et-archive/files/cdtr9464.pdf> (cit. on p. 26).
- Erdmann, Jakob and Ebendt, Rüdiger (Oct. 2014). "Benchmarking Trajectory Matchers with SUMO". In: *European Simulation and Modelling Conference*. 28th European Simulation and Modelling Conference. Porto, Portugal: Eurosis-ETI, pp. 226–230. ISBN: 978-90-77381-86-1. URL: <https://elib.dlr.de/93676/> (cit. on p. 43).
- Fabricio, Rodolfo Nogueira Leal and Nascimento, G. A. Gomes do (Mar. 2025). "The use of GNSS receivers in smartphones as high precision positioners". In: *The International Hydrographic Review* 29.2. DOI: [10.58440/ihr-29-2-n06](https://doi.org/10.58440/ihr-29-2-n06) (cit. on p. 22).
- Fiedler, David et al. (Feb. 2022). "Map Matching Algorithm for Large-scale Datasets". In: *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*. Vol. 3. ICAART 2022. SCITEPRESS - Science

- and Technology Publications, pp. 500–508. doi: [10.5220/0010849100003116](https://doi.org/10.5220/0010849100003116) (cit. on pp. 31, 32).
- Forney, G.D. (Mar. 1973). “The viterbi algorithm”. In: *Proceedings of the IEEE* 61.3, pp. 268–278. doi: [10.1109/proc.1973.9030](https://doi.org/10.1109/proc.1973.9030) (cit. on p. 26).
- García R, Yván J. et al. (Nov. 1998). “A greedy algorithm for bulk loading R-trees”. In: *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems*. GIS ’98. New York, NY, USA: Association for Computing Machinery, pp. 163–164. doi: [10.1145/288692.288723](https://doi.org/10.1145/288692.288723) (cit. on p. 112).
- Giofre, Vincenzo Pasquale et al. (June 2017). “Localization issues in the use of ITS”. In: *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems*. MT-ITS. IEEE, pp. 427–432. doi: [10.1109/mtits.2017.8005710](https://doi.org/10.1109/mtits.2017.8005710) (cit. on pp. 23, 24).
- Goh, C.Y. et al. (Sept. 2012). “Online map-matching based on Hidden Markov model for real-time traffic sensing applications”. In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, pp. 776–781. doi: [10.1109/itsc.2012.6338627](https://doi.org/10.1109/itsc.2012.6338627) (cit. on pp. 29, 31).
- Grabs, Elans et al. (Mar. 2008). “Accuracy Estimation of GPS Receiver Parameters with Re-Reference System in Static Mode”. In: *Telecommunications and Electronics* 8, pp. 31–36. URL: <https://ortus.rtu.lv/science/en/publications/4827> (cit. on pp. 22–24).
- Gupta, Ajay Kumar and Shanker, Udai (Sept. 2022). “A Comprehensive Review of Map-Matching Techniques: Empirical Analysis, Taxonomy, and Emerging Research Trends”. In: *International Journal of Web Services Research (IJWSR)* 19.1, pp. 1–32. doi: [10.4018/ijwsr.306243](https://doi.org/10.4018/ijwsr.306243) (cit. on p. 30).
- Hannun, Awni (Nov. 2019). *The Label Bias Problem*. URL: <https://awni.github.io/label-bias/> (cit. on p. 28).
- Hart, Peter et al. (Jan. 1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. doi: [10.1109/tssc.1968.300136](https://doi.org/10.1109/tssc.1968.300136) (cit. on p. 32).
- Hu, Jiahuan et al. (Feb. 2023). “A Comprehensive Analysis of Smartphone GNSS Range Errors in Realistic Environments”. In: *Sensors* 23.3, p. 1631. doi: [10.3390/s23031631](https://doi.org/10.3390/s23031631) (cit. on p. 22).
- Hu, Yigong and Lu, Binbin (Dec. 2019). “A Hidden Markov Model-Based Map Matching Algorithm for Low Sampling Rate Trajectory Data”. In:

- IEEE Access* 7, pp. 178235–178245. doi: [10.1109/access.2019.2958982](https://doi.org/10.1109/access.2019.2958982) (cit. on p. 32).
- Huang, Shih-Hao and Lin, Chow-Sing (Jan. 2019). “Rapid Route Comparison Based on GPS Coordinates and Bounding Boxes”. In: *Journal of Traffic and Logistics Engineering* 7.1, pp. 5–9. doi: [10.18178/jtle.7.1.5-9](https://doi.org/10.18178/jtle.7.1.5-9) (cit. on p. 39).
- Huang, Zhenfeng et al. (Mar. 2021). “Survey on vehicle map matching techniques”. In: *CAAI Transactions on Intelligence Technology* 6.1, pp. 55–71. doi: [10.1049/cit2.12030](https://doi.org/10.1049/cit2.12030) (cit. on pp. 12, 30).
- Hummel, Britta (Jan. 2006). “Map Matching for Vehicle Guidance”. In: *Dynamic and mobile GIS*. Ed.: J. Drummond 10, pp. 157–168. URL: <https://publikationen.bibliothek.kit.edu/1000011938> (cit. on p. 26).
- Izet-Ünsalan, K. and Ünsalan, M. (July 2020). “Classroom study of GNSS position accuracy using smartphones”. In: *Scientific Bulletin of Naval Academy XXIII.1*, pp. 83–89. doi: [10.21279/1454-864x-20-i1-011](https://doi.org/10.21279/1454-864x-20-i1-011) (cit. on p. 23).
- Jagadeesh, George R. and Srikanthan, Thambipillai (Aug. 2019). “Fast Computation of Clustered Many-to-many Shortest Paths and Its Application to Map Matching”. In: *ACM Transactions on Spatial Algorithms and Systems* 5.3, pp. 1–20. doi: [10.1145/3329676](https://doi.org/10.1145/3329676) (cit. on p. 32).
- Jiang, Linli et al. (June 2023). “L2MM: Learning to Map Matching with Deep Models for Low-Quality GPS Trajectory Data”. In: *ACM Transactions on Knowledge Discovery from Data* 17.3, pp. 1–25. doi: [10.1145/3550486](https://doi.org/10.1145/3550486) (cit. on p. 30).
- Jin, Meihan et al. (Nov. 2017). “A map-matching approach for travel behavior analysis”. In: *4th International Conference on Systems and Informatics*. ICSAI 2017. Hangzhou, China: IEEE, pp. 1405–1410. doi: [10.1109/ic sai.2017.8248506](https://doi.org/10.1109/ic sai.2017.8248506) (cit. on p. 13).
- Jin, Zhixiong et al. (July 2022). “Transformer-based map-matching model with limited labeled data using transfer-learning approach”. In: *Transportation Research Part C: Emerging Technologies* 140, p. 103668. doi: [10.1016/j.trc.2022.103668](https://doi.org/10.1016/j.trc.2022.103668) (cit. on p. 30).
- Karich, Peter (May 2015). *Map Matching use cases, or why an Open Source and Open Data alternative is superior to the Google Maps Roads API*. URL: <https://www.graphhopper.com/blog/2015/05/26/map-matching-use-cases-or-why-an-open-source-and-open-data-alternative-is-superior-to-the-google-maps-roads-api/> (cit. on p. 13).

- Knopp, Sebastian (Oct. 2006). “Efficient Computation of Many-to-Many Shortest Paths”. MA thesis. Karlsruhe: Universität Karlsruhe. URL: http://i11www.iti.kit.edu/_media/teaching/theses/files/da-sknopp-06.pdf (cit. on p. 32).
- Kos, Serdo et al. (Mar. 2013). “Smartphone application GPS performance during various space weather conditions: A preliminary study”. In: *Proceedings of the 21st International Symposium on Electronics in Transport*. ISEP 2013. Ljubljana, Slovenija: Elektrotehniška zveza Slovenije, pp. 15–18. URL: <https://www.bib.irb.hr/623174> (cit. on p. 22).
- Krajzewicz, Daniel et al. (Aug. 2002). “SUMO (Simulation of Urban Mobility) - An open-source traffic simulation”. In: *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002)*. Ed. by A. Al-Akaidi. 4th Middle East Symposium on Simulation and Modelling, pp. 183–187. ISBN: 90-77039-09-0. URL: <https://elib.dlr.de/6661/> (cit. on p. 43).
- Krumm, John et al. (Apr. 2007). “Map Matching with Travel Time Constraints”. In: *SAE World Congress & Exhibition*. Warrendale, PA, United States: SAE International. doi: [10.4271/2007-01-1102](https://doi.org/10.4271/2007-01-1102) (cit. on pp. 26, 31).
- Kubicka, Matej et al. (June 2015). “Dataset for testing and training of map-matching algorithms”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. Seoul, South Korea: IEEE. doi: [10.1109/ivs.2015.7225829](https://doi.org/10.1109/ivs.2015.7225829) (cit. on pp. 37, 138, 139).
- Kubicka, Matej et al. (July 2016). *Dataset for testing and training map-matching methods*. Seoul, South Korea. doi: [10.5281/zenodo.57731](https://doi.org/10.5281/zenodo.57731) (cit. on pp. 37, 151).
- Kubicka, Matej et al. (Apr. 2018). “Comparative Study and Application-Oriented Classification of Vehicular Map-Matching Methods”. In: *IEEE Intelligent Transportation Systems Magazine* 10.2, pp. 150–166. doi: [10.1109/mits.2018.2806630](https://doi.org/10.1109/mits.2018.2806630) (cit. on p. 30).
- Lafferty, John D. et al. (June 2001). “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 282–289. URL: <https://dl.acm.org/doi/10.5555/645530.655813> (cit. on pp. 27, 28).

- Leutenegger, S.T. et al. (Apr. 1997). “STR: a simple and efficient algorithm for R-tree packing”. In: *Proceedings 13th International Conference on Data Engineering*. IEEE Comput. Soc. Press. DOI: [10.1109/icde.1997.582015](https://doi.org/10.1109/icde.1997.582015) (cit. on p. 112).
- Li, Hengfeng et al. (Nov. 2014). “Spatio-temporal trajectory simplification for inferring travel paths”. In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '14. New York, NY, USA: Association for Computing Machinery, pp. 63–72. ISBN: 9781450331319. DOI: [10.1145/2666310.2666409](https://doi.org/10.1145/2666310.2666409) (cit. on pp. 35, 36, 138, 139, 146).
- Li, Hengfeng et al. (Dec. 2015). “Robust inferences of travel paths from GPS trajectories”. In: *International Journal of Geographical Information Science* 29.12, pp. 2194–2222. DOI: [10.1080/13658816.2015.1072202](https://doi.org/10.1080/13658816.2015.1072202) (cit. on pp. 12, 35).
- Li, Wei et al. (Mar. 2024). “A Novel Map Matching Method Based on Improved Hidden Markov and Conditional Random Fields Model”. In: *International Journal of Digital Earth* 17.1. DOI: [10.1080/17538947.2024.2328366](https://doi.org/10.1080/17538947.2024.2328366) (cit. on pp. 27, 28).
- Liang, Biwei et al. (Apr. 2016). “Online Learning for Accurate Real-Time Map Matching”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by James Bailey et al. Cham: Springer International Publishing, pp. 67–78. ISBN: 978-3-319-31750-2. DOI: [10.1007/978-3-319-31750-2_6](https://doi.org/10.1007/978-3-319-31750-2_6) (cit. on p. 29).
- Liao, Lin et al. (Jan. 2007). “Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields”. In: *The International Journal of Robotics Research* 26.1, pp. 119–134. DOI: [10.1177/0278364907073775](https://doi.org/10.1177/0278364907073775) (cit. on p. 27).
- Lindholm, Hugo (Jan. 2020). “Map Matching to road segments using Hidden Markov Model with GNSS, Odometer and Gyroscope”. MA thesis. Linköping, Sweden: Linköping University, p. 69. URL: <https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-162706> (cit. on p. 43).
- Liu, Xiliang and Lu, Feng (Nov. 2015). “ST-CRF: A Novel Map Matching Approach for Low-frequency Floating Car Data”. In: *Proceedings of the 6th ACM SIGSPATIAL International Workshop on GeoStreaming*. IWGS '15. New York, NY, USA: Association for Computing Machinery, pp. 9–18. DOI: [10.1145/2833165.2833169](https://doi.org/10.1145/2833165.2833169) (cit. on pp. 27, 28).

- Liu, Zhijia et al. (Feb. 2021). “Deep learning enabled vehicle trajectory map-matching method with advanced spatial-temporal analysis”. In: *IET Intelligent Transport Systems* 14.14, pp. 2052–2063. doi: [10.1049/iet-its.2020.0486](https://doi.org/10.1049/iet-its.2020.0486) (cit. on p. 30).
- Lopez, Pablo Alvarez et al. (Nov. 2018). “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, pp. 2575–2582. doi: [10.1109/itsc.2018.8569938](https://doi.org/10.1109/itsc.2018.8569938) (cit. on p. 43).
- Lou, Yin et al. (Nov. 2009). “Map-Matching for Low-Sampling-Rate GPS Trajectories”. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ’09. New York, NY, USA: Association for Computing Machinery, pp. 352–361. doi: [10.1145/1653771.1653820](https://doi.org/10.1145/1653771.1653820). URL: <https://www.microsoft.com/en-us/research/publication/map-matching-for-low-sampling-rate-gps-trajectories/> (cit. on pp. 31, 32, 41).
- Louail, Thomas et al. (June 2014). “From mobile phone data to the spatial structure of cities”. In: *Scientific Reports* 4.1, pp. 5276–5287. doi: [10.1038/srep05276](https://doi.org/10.1038/srep05276) (cit. on p. 41).
- Luxen, Dennis and Vetter, Christian (Nov. 2011). “Real-time routing with OpenStreetMap data”. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ’11. New York, New York, USA: ACM Press, pp. 513–516. doi: [10.1145/2093973.2094062](https://doi.org/10.1145/2093973.2094062) (cit. on p. 41).
- Ma, Xiaoyi et al. (Mar. 2021). “Evaluation of Accuracy of Traffic Flow Generation in SUMO”. In: *Applied Sciences* 11.6, p. 2584. doi: [10.3390/app11062584](https://doi.org/10.3390/app11062584) (cit. on p. 43).
- Marais, Juliette et al. (Nov. 2005). “Land Mobile GNSS Availability and Multipath Evaluation Tool”. In: *IEEE Transactions on Vehicular Technology* 54.5, pp. 1697–1704. doi: [10.1109/tvt.2005.853461](https://doi.org/10.1109/tvt.2005.853461) (cit. on p. 22).
- Meratnia, Nirvana and By, Rolf A. de (Jan. 2004). “Spatiotemporal Compression Techniques for Moving Point Objects”. In: *Advances in Database Technology*. EDBT 2004. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 765–782. doi: [10.1007/978-3-540-24741-8_44](https://doi.org/10.1007/978-3-540-24741-8_44) (cit. on p. 34).
- Millard-Ball, Adam et al. (May 2019). “Map-matching poor-quality GPS data in urban environments: the pgMapMatch package”. In: *Transportation*

- Planning and Technology* 42.6, pp. 539–553. doi: [10.1080/03081060.2019.1622249](https://doi.org/10.1080/03081060.2019.1622249) (cit. on p. 42).
- Mohamed, Ali et al. (Nov. 2012). “ACM SIGSPATIAL GIS Cup 2012”. In: *ACM SIGSPATIAL GIS '12, November 6-9, 2012. Redondo Beach, CA, USA*. SIGSPATIAL '12. doi: [10.1145/2424321.2424426](https://doi.org/10.1145/2424321.2424426). URL: <https://www.microsoft.com/en-us/research/publication/acm-sigspatial-gis-cup-2012/> (cit. on pp. 37, 138, 139, 148).
- Newson, Paul and Krumm, John (Nov. 2009). “Hidden Markov Map Matching through Noise and Sparseness”. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '09. New York, NY, USA: Association for Computing Machinery, pp. 336–343. doi: [10.1145/1653771.1653818](https://doi.org/10.1145/1653771.1653818). URL: <https://www.microsoft.com/en-us/research/publication/hidden-markov-map-matching-noise-sparseness/> (cit. on pp. 23, 27–29, 31, 34, 36, 38, 41, 42, 138, 141, 145, 148).
- Ochieng, W.Y. et al. (Nov. 2009). “Map-matching in complex urban road networks”. In: *Revista Brasileira de Cartografia* 55.2, p. 14. doi: [10.14393/rbcbv55n2-43490](https://doi.org/10.14393/rbcbv55n2-43490) (cit. on p. 29).
- Osogami, Takayuki and Raymond, Rudy (Aug. 2013). “Map Matching with Inverse Reinforcement Learning”. In: *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, pp. 2547–2553. URL: <https://dl.acm.org/doi/10.5555/2540128.2540495> (cit. on p. 36).
- Pereira, Francisco Câmara et al. (Sept. 2009). “An off-line map-matching algorithm for incomplete map databases”. In: *European Transport Research Review* 1.3, pp. 107–124. doi: [10.1007/s12544-009-0013-6](https://doi.org/10.1007/s12544-009-0013-6) (cit. on p. 29).
- Pereira, Rui et al. (Oct. 2017). “Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?” In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2017. New York, NY, USA: Association for Computing Machinery, pp. 256–267. ISBN: 9781450355254. doi: [10.1145/3136014.3136031](https://doi.org/10.1145/3136014.3136031). URL: <https://sites.google.com/view/energy-efficiency-languages> (cit. on p. 87).
- Quddus, Mohammed A. et al. (Oct. 2007). “Current map-matching algorithms for transport applications: State-of-the art and future research

- directions". In: *Transportation Research Part C: Emerging Technologies* 15.5, pp. 312–328. doi: [10.1016/j.trc.2007.05.002](https://doi.org/10.1016/j.trc.2007.05.002) (cit. on pp. 12, 26).
- Rahmani, Mahmood (May 2015). "Urban Travel Time Estimation from Sparse GPS Data : An Efficient and Scalable Approach". PhD thesis. Stockholm, Sweden: KTH, Transport Planning, Economics and Engineering, p. 64. ISBN: 978-91-87353-72-7. URL: <https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-167798> (cit. on p. 13).
- Ramer, Urs (Nov. 1972). "An iterative procedure for the polygonal approximation of plane curves". In: *Computer Graphics and Image Processing* 1.3, pp. 244–256. doi: [10.1016/s0146-664x\(72\)80017-0](https://doi.org/10.1016/s0146-664x(72)80017-0) (cit. on p. 34).
- Ranacher, Peter et al. (Sept. 2015). "Why GPS makes distances bigger than they are". In: *International Journal of Geographical Information Science* 30.2, pp. 316–333. doi: [10.1080/13658816.2015.1086924](https://doi.org/10.1080/13658816.2015.1086924) (cit. on pp. 23, 24).
- Ranacher, Peter et al. (Jan. 2016). "What is an Appropriate Temporal Sampling Rate to Record Floating Car Data with a GPS?" In: *ISPRS International Journal of Geo-Information* 5.1, p. 17. doi: [10.3390/ijgi5010001](https://doi.org/10.3390/ijgi5010001) (cit. on p. 24).
- Rostami, Ali et al. (Sept. 2019). "A Light-Weight Smartphone GPS Error Model for Simulation". In: *2019 IEEE 90th Vehicular Technology Conference. VTC2019-Fall*. IEEE, pp. 1–5. doi: [10.1109/vtcfall.2019.8891089](https://doi.org/10.1109/vtcfall.2019.8891089) (cit. on pp. 23, 25).
- Saki, Siavash and Hagen, Tobias (Aug. 2022). "A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data". In: *SN Computer Science* 3.5. doi: [10.1007/s42979-022-01340-5](https://doi.org/10.1007/s42979-022-01340-5) (cit. on p. 32).
- Schweizer, Joerg et al. (Mar. 2021). "Building a Large-Scale Micro-Simulation Transport Scenario Using Big Data". In: *ISPRS International Journal of Geo-Information* 10.3, p. 165. doi: [10.3390/ijgi10030165](https://doi.org/10.3390/ijgi10030165) (cit. on p. 43).
- Shen, Zhihao et al. (Sept. 2020). "DMM: Fast Map Matching for Cellular Data". In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. MobiCom '20. New York, NY, USA: Association for Computing Machinery. doi: [10.1145/3372224.3421461](https://doi.org/10.1145/3372224.3421461) (cit. on p. 30).
- Soundy, Andy W. R. et al. (Oct. 2020). "Comparison of Enhanced Noise Model Performance Based on Analysis of Civilian GPS Data". In: *Sensors* 20.21, p. 6050. doi: [10.3390/s20216050](https://doi.org/10.3390/s20216050) (cit. on pp. 23, 25).

- Sun, Danyang (Nov. 2022). “Floating Car Data mining to feature out mobility patterns: individual-centered and place-based analyses”. PhD thesis. École des Ponts ParisTech. URL: <https://pastel.archives-ouvertes.fr/tel-03637330> (cit. on p. 13).
- Sun, Wenbin and Xiong, Ting (Jan. 2016). “A Low-Sampling-Rate Trajectory Matching Algorithm in Combination of History Trajectory and Reinforcement Learning”. In: *Acta Geodaetica et Cartographica Sinica* 45.11, p. 1328. doi: [10.11947/j.AGCS.2016.20160046](https://doi.org/10.11947/j.AGCS.2016.20160046) (cit. on p. 36).
- Sutton, Richard S. and Barto, Andrew G. (Jan. 2018). *Reinforcement Learning: An Introduction (Second Edition)*. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press. URL: <http://incompleteideas.net/book/the-book-2nd.html> (cit. on pp. 35, 62).
- Vincenty, T. (Apr. 1975). “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations”. In: *Survey Review* 23.176, pp. 88–93. doi: [10.1179/sre.1975.23.176.88](https://doi.org/10.1179/sre.1975.23.176.88) (cit. on p. 106).
- Watkins, Christopher John Cornish Hellaby (May 1989). “Learning from Delayed Rewards”. PhD thesis. King’s College, p. 234. URL: <https://www.cs.rhul.ac.uk/~chrisw/thesis.html> (cit. on p. 35).
- Watkins, Christopher John Cornish Hellaby and Dayan, Peter (May 1992). *Q-learning*. Tech. rep. 3-4. Machine Learning, pp. 279–292. doi: [10.1007/bf00992698](https://doi.org/10.1007/bf00992698) (cit. on p. 35).
- Wei, Hong et al. (Nov. 2012). “Fast Viterbi Map Matching with Tunable Weight Functions”. In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL ’12. New York, NY, USA: Association for Computing Machinery, pp. 613–616. doi: [10.1145/2424321.2424430](https://doi.org/10.1145/2424321.2424430) (cit. on pp. 27, 31).
- Wei, Hong et al. (Nov. 2013). “Map Matching: Comparison of Approaches Using Sparse and Noisy Data”. In: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL’13. New York, NY, USA: Association for Computing Machinery, pp. 444–447. doi: [10.1145/2525314.2525456](https://doi.org/10.1145/2525314.2525456) (cit. on p. 27).
- White, Christopher E. et al. (Feb. 2000). “Some map matching algorithms for personal navigation assistants”. In: *Transportation Research Part C: Emerging Technologies* 8.1-6, pp. 91–108. doi: [10.1016/s0968-090x\(00\)00026-7](https://doi.org/10.1016/s0968-090x(00)00026-7) (cit. on pp. 26, 31).

- Wöltche, Adrian (Oct. 2021). *Evolving map matching with markov decision processes*. Hof. doi: [10.57944/1051-119](https://doi.org/10.57944/1051-119) (cit. on pp. 4, 88).
- Wöltche, Adrian (Oct. 2023). “Open source map matching with Markov decision processes: A new method and a detailed benchmark with existing approaches”. In: *Transactions in GIS* 27.7, pp. 1959–1991. doi: [10.1111/tgis.13107](https://doi.org/10.1111/tgis.13107) (cit. on pp. 4, 41, 42, 88, 138, 140, 153).
- Wöltche, Adrian (Jan. 2025). “Artificial Ground Truth Data Generation for Map Matching with Open Source Software”. In: *Tagungsband FOSSGIS-Konferenz 2025*. FOSSGIS2025. Zenodo, p. 17. ISBN: 978-3-00-081664-2. doi: [10.5281/zenodo.14774143](https://doi.org/10.5281/zenodo.14774143) (cit. on pp. 4, 90).
- Wu, Hao et al. (Aug. 2016). “Probabilistic Robust Route Recovery with Spatio-Temporal Dynamics”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. New York, NY, USA: Association for Computing Machinery, pp. 1915–1924. doi: [10.1145/2939672.2939843](https://doi.org/10.1145/2939672.2939843) (cit. on pp. 36, 38).
- Xi, Lianxia et al. (Oct. 2007). “Map Matching Algorithm and Its Application”. In: *Proceedings of the 2007 International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007)*. Advances in Intelligent Systems Research. Paris, France: Atlantis Press, pp. 740–746. ISBN: 978-90-78677-04-8. doi: [10.2991/iske.2007.127](https://doi.org/10.2991/iske.2007.127) (cit. on p. 12).
- Xu, Ming et al. (Sept. 2015). “Map Matching Based on Conditional Random Fields and Route Preference Mining for Uncertain Trajectories”. In: *Mathematical Problems in Engineering* 2015, pp. 1–13. doi: [10.1155/2015/717095](https://doi.org/10.1155/2015/717095) (cit. on pp. 27, 28).
- Yang, Can (Dec. 2020). “Efficient Map Matching and Discovery of Frequent and Dominant Movement Patterns in GPS Trajectory Data”. PhD thesis. Stockholm, Sweden: KTH Royal Institute of Technology, p. 62. URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-286318> (cit. on p. 32).
- Yang, Can and Gidófalvi, Győző (Mar. 2018). “Fast map matching, an algorithm integrating hidden Markov model with precomputation”. In: *International Journal of Geographical Information Science* 32.3, pp. 547–570. doi: [10.1080/13658816.2017.1400548](https://doi.org/10.1080/13658816.2017.1400548) (cit. on pp. 28, 29, 31, 33, 41, 99).
- Yin, Yifang et al. (Aug. 2018). “Feature-based Map Matching for Low-Sampling-Rate GPS Trajectories”. In: *ACM Transactions on Spatial Algorithms and Systems* 4.2, pp. 1–24. doi: [10.1145/3223049](https://doi.org/10.1145/3223049) (cit. on p. 29).

- Zangenehnejad, Farzaneh and Gao, Yang (Nov. 2021). “GNSS smartphones positioning: advances, challenges, opportunities, and future perspectives”. In: *Satellite Navigation* 2.24, p. 23. doi: [10.1186/s43020-021-00054-y](https://doi.org/10.1186/s43020-021-00054-y) (cit. on p. 22).
- Zhang, Ethan and Masoud, Neda (Feb. 2020). “Increasing GPS Localization Accuracy With Reinforcement Learning”. In: *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12. doi: [10.1109/tits.2020.2972409](https://doi.org/10.1109/tits.2020.2972409) (cit. on p. 35).
- Zhao, Kai et al. (Nov. 2019). “DeepMM: Deep Learning Based Map Matching with Data Augmentation”. In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '19. New York, NY, USA: Association for Computing Machinery, pp. 452–455. doi: [10.1145/3347146.3359090](https://doi.org/10.1145/3347146.3359090) (cit. on p. 30).
- Zhao, Yue et al. (July 2011). “A new method of road network updating based on floating car data”. In: *IEEE International Geoscience and Remote Sensing Symposium*. IGARSS 2011. Vancouver, BC, Canada: IEEE, pp. 1878–1881. doi: [10.1109/igarss.2011.6049490](https://doi.org/10.1109/igarss.2011.6049490) (cit. on p. 12).
- Zheng, Kai et al. (Apr. 2012). “Reducing Uncertainty of Low-Sampling-Rate Trajectories”. In: *2012 IEEE International Conference on Data Engineering*. ICDE 2012. IEEE, pp. 1144–1155. doi: [10.1109/icde.2012.42](https://doi.org/10.1109/icde.2012.42) (cit. on p. 38).
- Zhong, Yanling et al. (Oct. 2022). “A trajectory data compression algorithm based on spatio-temporal characteristics”. In: *PeerJ Computer Science* 8, e1112. doi: [10.7717/peerj-cs.1112](https://doi.org/10.7717/peerj-cs.1112) (cit. on p. 34).
- Ziebart, Brian D. et al. (July 2008). “Maximum Entropy Inverse Reinforcement Learning”. In: *Proceedings of the 23rd National Conference on Artificial Intelligence*. Vol. Volume 3. AAAI '08. Chicago, Illinois: AAAI Press, pp. 1433–1438. URL: <https://dl.acm.org/doi/10.5555/1620270.1620297> (cit. on p. 36).



University
of Bamberg
Press

Map matching is a widely used technology for resolving discrepancies between sequences of geocoordinates from movements and road networks. Such discrepancies generally occur between positions measured with the help of global navigation satellite systems and the underlying roads on which the movements originally occurred. Map matching is a combination of defined models, algorithms, and metrics for correcting these discrepancies. The goal of map matching is to find routes within the road network that most accurately represent the recorded movements.

In this thesis, we present a novel map matching method based on Markov decision processes and reinforcement learning algorithms. Our novel method introduces various advances in multiple aspects of the map matching process pipeline for increasing the accuracy of our results in comparison to existing solutions. Our novel map matching solution is provided as ready-to-use open source software, and we compare our solution with multiple third-party open source solutions in terms of computational speed, resource consumption, and accuracy of the results. Our evaluation demonstrates that our novel map matching method is more robust to noise and outliers than existing methods and that it outperforms them in terms of accuracy and computational performance.



ISBN: 978-3-98989-135-7



9 783989 891357

www.uni-bamberg.de/ubp