A Long Movement Story Cut Short

—

On the Compression of Trajectory Data

A Long Movement Story Cut Short

—

On the Compression of Trajectory Data

Inaugural-Dissertation

zur Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von
Markus Koegel
aus Düsseldorf

Düsseldorf, Dezember 2012

Aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Referent:	Prof. Dr. Martin Mauve Heinrich-Heine-Universität Düsseldorf
Korreferent:	Prof. Dr. Björn Scheuermann Humboldt-Universität zu Berlin

Tag der mündlichen Prüfung: 25.01.2013

Abstract

In mobile computing, a technology is said to be *ubiquitous*, if it is integrated in everyday life to such an extend that it is taken for granted instead of actually being recognized as technology. There are already numerous examples for ubiquitous technologies, such as navigation systems, cell phones, or notebook computers. Many of these enable the users to participate in mobile networks and to benefit from location-aware applications. A prominent use case of ubiquitous computing and networking is the research area of *inter-vehicular communication*: road vehicles are planned to be equipped with computing units that have access to the vehicles' sensors and to wireless communication interfaces, so that vehicles can share information about experienced situations among each other. In doing so, the technique can be used to make road traffic safer, more efficient, and to provide a higher level of convenience to the passengers. To achieve these goals, a number of efficiency and convenience applications record and transmit the vehicles' trajectories, i. e., the sequence of positions over time. However, the transmission of trajectories can induce a significant load to the communication channel and may waste resources that are required by safety applications to work properly. To avoid this situation, a number of lossy trajectory compression algorithms have been proposed in literature that allow for an adjustable compression error bound.

In this thesis, we examine compression algorithms for trajectory data. Though these algorithms are suitable for all sorts of movements, we focus on the use case of vehicular trajectory compression. We take a close look at state-of-the-art solutions that are based on geometric operations, namely line simplification and linear dead reckoning. We argue that though achieving good results, these models can not be the best possible approach with respect to the achieved compression ratio, because vehicular movements are not linear. Instead, we motivate the use of nonlinear algorithms and propose two geometric compression algorithms based on clothoid spline sketching and cubic spline interpolation, respectively. By means of an evaluation based on real-world trajectory data, we show that nonlinear algorithms can provide a better compression ratio than the optimal line simplification solution.

While our spline interpolation based algorithm provides the best compression ratios, it implements a heuristic and therefore merely finds a locally optimal solution. We therefore turn to an information-theoretic approach and aim at finding an algorithm that provides the optimal compression ratio for a trajectory. We therefore use the Shannon information content to define the information content of a trajectory and propose a method to measure it, using a movement estimator, a discretization technique and a probability distribution. We suggest and evaluate different component implementations and find out that implementing an arithmetic coder based on this method yields significantly better compression ratios than the geometric approaches.

We finally turn to lossless compression algorithms and consider the use of conventional byte string compression algorithms, such as several algorithms from the LZ family, the *bzip2* algorithm and arithmetic coding. A plain application of these algorithms to the trajectories would not produce meaningful results, though, because subsequently measured trajectory positions that are close to each other are not necessarily represented by similar byte sequences that would be easy to compress. We therefore propose a byte encoder that preprocesses a trajectory into a form that can be compressed more effectively by the selected conventional compression algorithms. We evaluate the byte coder based on the same trajectory set and see that the achieved compression ratio nearly matches the results from the lossy arithmetic coder for the lowest selected error tolerance.

With these results, we provide an answer to the question of how spatio-temporal trajectories can be compressed so that a maximum compression ratio can be achieved with respect to the application's accuracy requirements.

Zusammenfassung

Im Bereich der Mobilkommunikation werden Technologien als *allgegenwärtig* oder *ubiquitär* bezeichnet, wenn sie sich derart in den Alltag integrierten, dass sie nicht mehr als Technologien wahrgenommen werden, wie zum Beispiel Navigationssysteme, Mobiltelefone oder Notebooks. Viele dieser Technologien sind vernetzt und bieten ihren Benutzern ortsbezogene Dienste an. Ein prominenter Anwendungsfall hierfür ist die *Fahrzeug-zu-Fahrzeug Kommunikation*, die derzeit noch erforscht und prototypisch entwickelt wird. Hierbei werden reguläre Straßenfahrzeuge mit Computern ausgestattet, die Zugriff zu den Fahrzeugsensoren und Funkkommunikationsmodulen haben und somit Informationen aus vergangenen Situationen untereinander austauschen und verbreiten können; so sollen die Sicherheit, die Effizienz und der Komfort im Straßenverkehr gesteigert werden. Hierfür zeichnen eine Reihe von Effizienz- und Komfortanwendungen Zeit- und Bewegungsdaten (Trajektorien) auf und übertragen diese zur Weiterverarbeitung an externe Server oder Netzwerke. Da solche Trajektorien jedoch potentiell sehr groß werden, können deren Übertragungen eine erhebliche Datenlast auf dem drahtlosen Medium verursachen. Dadurch entsteht die Gefahr, dass Ressourcen belegt werden, die von Sicherheitsanwendungen dringend benötigt werden. Um solche Situationen zu vermeiden, wurden bereits verlustbehaftete Trajektoriekompressionsmethoden in der Literatur vorgeschlagen, bei denen eine harte räumliche Fehlerschranke frei einstellbar ist.

In dieser Arbeit untersuchen wir Kompressionsalgorithmen für Trajektorien. Auch wenn diese Algorithmen für beliebige Bewegungen geeignet sind, konzentrieren wir uns dabei auf Fahrzeugbewegungen. Wir betrachten den derzeitigen Stand der Technik, bei dem geometrische Operationen, nämlich Linienvereinfachung und lineare Bewegungsmodelle verwendet werden. Wir argumentieren, dass diese Modelle nicht die bestmögliche Kompression erreichen können, da Fahrzeugbewegungen nicht linear sind und bekräftigen stattdessen die Verwendung von nichtlinearen Modellen. Wir schlagen zwei geometrische Kompressionsalgorithmen vor, die auf Klothoidensplines, bzw. kubischer Splineinterpolation basieren. Unsere Auswertungen auf Basis von Realweltdaten

zeigen, dass nichtlineare Algorithmen eine bessere Kompressionsrate erzielen können als die optimalen Linien vereinfachenden Ansätze.

Während unser auf Splineinterpolation basierender Algorithmus die beste Kompressionsrate für geometrische Verfahren erreicht, arbeitet dieser heuristisch und findet lediglich eine lokal optimale Lösung. Wir verwenden daher einen informationstheoretischen Ansatz, um einen Algorithmus zu finden, der eine bestmögliche Kompressionsrate erreicht. Mithilfe des Shannon-Informationsgehalts definieren wir den Informationsgehalt einer Trajektorie und schlagen eine Methode vor, um diesen zu messen. Hierbei findet ein Bewegungsschätzer, eine Diskretisierung von Schätzfehlern und eine Wahrscheinlichkeitsverteilung Anwendung. Wir beschreiben verschiedene Implementierungsalternativen für diese Komponenten und zeigen in unserer Auswertung, dass die Umsetzung dieser Methode in einem arithmetischen Kodierer deutlich bessere Kompressionsraten als die geometrischen Ansätze erzielt.

Als Abschluss wenden wir uns den konventionellen verlustfreien Kompressionsalgorithmen zu und untersuchen mehrere Algorithmen aus der LZ-Familie, den bzip2-Algorithmus und die arithmetische Kodierung. Eine einfache Anwendung dieser Algorithmen auf die Trajektorien ist allerdings nicht sinnvoll, weil sich die Binärrepräsentationen ähnlicher Trajektoriepositionen nicht zwangsläufig ähneln und somit nicht notwendigerweise gut zu komprimieren wären. Wir präsentieren daher einen Byte-Kodierer, der in einem Vorverarbeitungsschritt den Bytecode einer Trajektorie derart umformatiert, dass er besser durch die ausgewählten konventionellen Algorithmen komprimiert werden kann. Unsere Auswertung zeigt, dass die erreichte Kompressionsrate der verlustfreien Algorithmen beinahe der des verlustbehafteten arithmetischen Kodierers für die niedrigste untersuchte Fehlertoleranz entspricht.

Durch unsere Ergebnisse zeigen wir, wie eine Trajektorie komprimiert werden muss, um eine möglichst hohe Kompressionsrate in Abhängigkeit von den Genauigkeitsanforderungen der jeweiligen Anwendung zu erreichen.

Acknowledgments

First and foremost, I would like to express my gratitude to my doctoral advisor, Martin Mauve, who offered me the opportunity to start my doctoral studies and to join the computer networking group at the department of computer science at the Heinrich Heine University of Düsseldorf. During my studies, he supported me with valuable comments, suggestions and discussions, and thus helped me to stay on track and to successfully complete my work. I could not have had a better doctoral advisor than him and wish him all the best and many successful years and projects yet to come.

Even after his time at the Heinrich Heine University, Björn Scheuermann took his time to discuss my new ideas and to suggest different perspectives to look at particular problems. I would like to thank him very much for the past eight years, in which he has been my advisor for the Bachelor and Master Thesis, and supported me in my time as a research assistant.

I am also grateful to Volker Aurich, who helped me to improve the quality of my articles and this dissertation by reading the drafts and providing valuable feedback.

I would like to thank my colleagues Daniel Baselt, Tobias Escher, Sabine Freese, Norbert Goebel, Philipp Hagemeister, Yves Jerschow, Jedrzej Rybicki, Pascal Siegers, and Thomas Spitzlei for the excellent working atmosphere that really counteracted a lot of stressful situations and thus helped to get through some rather difficult times. I was also very lucky to be able to work together with Gian Perrone, Michael Singhof, Dennis Dobler, Erzen Hyko, Matthias Radig, and Andreas Disterhöft, who all did great jobs in implementing and discussing the prototypes and ideas for the trajectory compression algorithms; thank you all a lot!

I do not want and cannot forget to thank my friends Frank, Thomas, Eva, Steffi, Ralf, Javi, Elena, Andreas, Raphaela, and Steffen for proofreading my paper and dissertation drafts, sharing exciting and laid-back moments or simply for being there!

A very special “thank you” goes to my whole family and particularly to Astrid for providing unique support especially in tough times and for listening to my gibberish in all kinds of situations. I would not know what to do without you!

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ASCII	American Standard Code for Information Interchange
BWT	Burrows-Wheeler Transformation
C2C	Car to Car
C2I	Car to Infrastructure
CDR	Connection-preserving Dead Reckoning
DBN	Dynamic Bayesian Network
DOP	Dilution of Precision
FCD	Floating Car Data
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GNSS	Global Navigation Satellite System
GPL	GNU General Public License
GPS	Global Positioning System
HMM	Hidden Markov Model
IEEE	Institute of Electrical and Electronics Engineers
LDR	Linear Dead Reckoning
LIDAR	Light Detection and Ranging
LZ	Lempel-Ziv
LZMA	Lempel-Ziv-MArkov
LZW	Lempel-Ziv-Welch
MOD	Moving Objects Database
MTF	Move-To-Front
NMEA	National Marine Electronics Association
OBU	On-Board Unit
ODbL	Open Data Commons Open Database License
OSM	OpenStreetMap
POMDP	Partially Observable Markov Decision Process

PPM	Prediction by Partial Matching
RINEX	Receiver Independent EXchange (Format)
RLE	Run Length Encoding
RSU	Road Side Unit
TDOA	Time Difference of Arrival
UERE	User Equivalent Range Errors
WLAN	Wireless Local Area Network
XFCD	eXtended Floating Car Data

1 Introduction

Computer networking and communications play an important and continuously increasing role in modern life. These technologies not only enable a rapid exchange of all sorts of information, such as voice or hypertext data, but have also evolved into being central aspects of present-day businesses. Furthermore, computation devices are long since mobile, and so is the communication between them. Of course, this encompasses common and wide-spread every-day devices, like notebook computers or cell phones, but in general, computing and networking has become more and more *ubiquitous* in recent years. Doing so means, in accordance to Mark Weiser's idea from 1988, that computing devices "weave themselves into the fabric of everyday life until they are indistinguishable from it" [Wei91, p. 1]. In his article from 1991 [Wei91], Weiser not only envisioned the ubiquitousness of computing devices, but he realized that such devices would need to participate in a network and be aware of their positions, either in an absolute or at least relative sense to unfold their full potential.

As a prominent use case of ubiquitous computing and networking, *inter-vehicular communication*, also known as *car to car (C2C)* or *car to infrastructure (C2I)* communication, has been researched and developed in various projects, such as [proa, prob, proc, prod]. For inter-vehicular communication, road vehicles are equipped with *on-board units (OBUs)* that are connected to a number of electronic sensors, for instance to measure the distance to surrounding obstacles with *Light Detection and Ranging (LIDAR)*, to detect humidity, or to determine its current absolute position with a *Global Navigation Satellite System (GNSS)*. The OBUs may of course vary in their performance, depending on the class of the vehicle in which they are installed; there may be simpler units built into lower class vehicles on the one side, but for upper class

vehicles, there might be more powerful OBUs available that could host more complex applications. Generally, OBUs also contain networking capabilities, so measurements can be exchanged and shared among vicinal vehicles via wireless local area networking (WLAN) techniques or sent via a mobile cell communication link for further processing and distribution. Based on this shared information basis, applications can be implemented to improve the safety and efficiency of road traffic or that provide convenient services, such as information about local points of interest or multimedia streaming.

Some services, such as fleet management, roadway monitoring or map generation, require vehicles to report their trajectories to a central unit, where these pieces of information can be processed. However, the reported vehicles' trajectories also mostly contain temporal information, and may be extended with even more sensor data; so, the data volume that is sent from the vehicles might become very large. Especially if the employed medium is very limited or if the wireless communication comes with additional costs, it is important to reduce the induced data load on the channel. In case that cellular communication is not available, vehicles can only communicate via local area networking and so, a connection to a central unit is usually only given through *Road Side Units (RSUs)*. Basically, RSUs are low-performance computing and communication units that are mounted at the shoulder of the road or at overhead gantry signs and that are connected to a backbone network. Therefore, they can serve as gateways to relay certain types of messages. In such scenarios, when many vehicles attempt to transmit their trajectory reports over the wireless link in the surrounding of an RSU, the link can easily get congested; in this case, the necessary capacity for more important safety applications that work on the same link might not remain. To alleviate this situation, a number of lossy compression algorithms for spatio-temporal trajectories, i. e., sequences of time-position tuples, have been proposed in the community of ubiquitous computing. These algorithms provide an accuracy parameter that determines the upper error threshold that must not be exceeded. The actual error tolerance depends on the use case; for animal tracking, an accuracy of 100 m may be sufficient, while for the vehicular case, the bound might be set to a few centimeters.

Although these approaches have shown to achieve acceptable compression ratios, the vast majority of proposed algorithms assume a linear movement and employs geometric *line simplification* methods or *linear dead reckoning (LDR)* and thus do not model vehicular mobility correctly. Based on this work, it remains unclear what compression rate can optimally be obtained and how much room for improvement remains at the moment. In this thesis, we address this issue and raise the question:

> *how do spatio-temporal trajectories need to be compressed so that a maximum compression ratio can be achieved?*

As this question clearly states, our focus lies on maximizing the compression performance; however, we also need to regard several requirements that will emerge in realistic use case scenarios. It could, for instance, make a difference for particular applications to compress the measured data in blocks or to handle it as a stream. Also, one always needs to keep in mind that all compression algorithms need to work on various platforms, as mentioned above, which again puts up requirements to the runtime complexity of the respective algorithms. In this context, more complex algorithms could be applicable to data blocks instead of data streams which could be disadvantageous for certain use cases. Finally, the complexity of the decompression is also an interesting factor that should not be forgotten.

To establish a solid understanding of the background to spatio-temporal trajectories, we give an introduction into satellite positioning technologies in Chapter 2 and explain how spatio-temporal measurements can be conducted. We also present an overview of applications that rely on positioning information and discuss the requirements that these applications put to the position measurements. Although we are especially interested in the use case of vehicular communications, we will see in the overview that the demand for trajectory compression is also present for other applications. Again, as the compression algorithms should be employed in realistic scenarios, they need to cope with noisy and potentially erroneous data. We therefore explain in detail how we established a data basis of real-world spatio-temporal trajectories that we obtained from the OpenStreetMap project [proe].

As already mentioned above, the major previous work in the area of trajectory compression has been conducted with line simplification algorithms and other geometric and linear models and therefore we regard this class of approaches more closely in Chapter 3. All of these previously published geometric compression algorithms are lossy and provide an accuracy parameter that can be used to adjust a strict error tolerance that defines the distance in which the decompressed trajectory may differ from the original in a point-to-point comparison. We first present and review the main contributions that have been published and discuss two representative line simplification algorithms that have been used in this course: next to the heuristic *Douglas-Peucker* algorithm, we introduce an optimal line simplification algorithm that constructs a graph structure and calculates the simplification result by finding the shortest path from the first to the last trajectory point representation in the graph. The evaluation of the linear approaches shows a good compression performance of these algorithms, especially for larger error tolerances of more than 1 m. However, we argue that modeling trajectories in a linear fashion does not match the principles of kinematics that regard object movements as smooth functions over time. We therefore examine other

geometric approximation methods employing nonlinear functions. We first follow the approach that if the road structure was known, the trajectories of vehicles following this structure should be easily compressible. As roads are usually designed as compositions of linear segments, circular arcs and so called clothoids that are used as transition curves, we discuss two algorithms from the area of computer graphics that sketch such compositions, also known as *clothoid splines*, based on noisy point sequences. However, these algorithms do not regard a strict approximation error tolerance; because of this, the resulting compression ratio does not reach the benchmark results achieved with the linear approaches. We therefore propose a second algorithm that approximates trajectories with cubic splines. Though the algorithm has a higher runtime complexity, it achieves significantly better compression ratios than its linear counterparts and thereby underlines our assumption that nonlinear models should be regarded when trajectories are to be approximated geometrically.

Despite the better compression ratio that can be achieved with our method using cubic spline interpolation, the optimal compression method for spatio-temporal trajectories remains unclear. Therefore, we propose to approach this topic from an information-theoretic view in Chapter 4, which is our main contribution in this thesis. In this chapter, we first introduce the information-theoretic fundamentals and describe the entropy coding schemes *Huffman coding*, *arithmetic coding*, and *range coding*. The latter is an implementation paradigm that approximates the arithmetic coding by using integer intervals instead of real numbers for expressing probabilities. From the basics, we then derive a definition of the information content of a spatio-temporal trajectory and propose an abstract way of how to calculate it, employing a movement estimator. While the abstract model is applicable to a wide range of movements, we also present an exemplary implementation of the information measurement for which we suggest realizations for all the model's components and discuss different configuration possibilities. We evaluate the proposed model and show that it significantly outperforms the geometric compression algorithms.

As our final contribution in this thesis, we leave lossy compression approaches behind and examine the potential of conventional lossless algorithms for the trajectory compression in Chapter 5. We regard several algorithms from the LZ family, the *bzip2* algorithm and the arithmetic coding. However, a plain application of these algorithms to the trajectories would not produce meaningful results, because the conventional algorithms work on byte streams; even if subsequently measured trajectory positions should be reasonably close to each other, the binary representation of such chained positions do not necessarily feature a large number of similar byte sequences due to coding scheme effects. Therefore, we propose a lossless byte encoding scheme for spatio-

temporal trajectories, the output of which can better be compressed by the the selected conventional compression algorithms. To achieve this, the coder basically implements a delta encoding by calculating the difference vectors of up to second degree for the trajectory elements, which should result in a majority of close-to-zero bytes. Our evaluation shows that our assumptions hold and that the preprocessing step with the delta encoding results in compression ratios that are comparable to the ones achieved with the lossy arithmetic coder at the lowest error tolerance of merely 5 cm.

2

Position Measurements and Applications

This chapter gives an introduction to the background of trajectory compression: we first describe briefly how modern satellite self-positioning mechanisms work and how spatio-temporal measurements can be conducted. We then give an overview of typical applications that base on position measurements, and discuss their requirements with respect to the employed position measurements. Finally, we explain how we collected the spatio-temporal trajectories that we will use throughout this thesis for the design and evaluation of our compression mechanisms.

2.1 Applications Relying on Position Measurements

Self-positioning by means of satellite signal measurements has evolved from a technique that has almost exclusively been employed for professional naval and aerial navigation and for military purposes to a service that is not only available to, but also widely used by civilian use cases and applications. This covers straightforward application scenarios, such as static measurements for exact land surveying or mobile measurements for personal map navigation while driving in a car or while hiking in unknown terrain. Also, a large number of more sophisticated applications have emerged that not only display the positioning information, but process or store it for future use and evaluation. It is obvious by this differentiation alone that such a variety of applications yields a wide spectrum of requirements that need to be fit. In particular, we regard the demands of the use cases for positioning accuracy and the timeliness of position measurement reports of a mobile unit at a central entity.

In this context, it will often be referred to *spatio-temporal trajectories* and *movement measurements*. While it is clear that static positioning describes the determination of

the current position at a halt, movement measurements can be understood as subsequent position measurements while being in motion. Strictly speaking, the term movement measurement is therefore not accurate, because the movement parameters, such as velocity, heading and acceleration, are not necessarily measured. Primarily, snapshots of the movement at certain points in time are recorded, on the basis of which the movement parameters can then be approximated. In other words, we do not measure the continuous function of a moving object's position over time, but only sample the position at discrete points in time. Then, the concatenation of (at least) time and position tuples is named a (discrete) spatio-temporal trajectory.

Therefore, before focusing on the different applications themselves, we have a brief look at the technical background of *Global Navigation Satellite Systems (GNSSs)*. Using the example of the *Global Positioning System (GPS)*, we establish a basis for understanding the opportunities and limitations of self-positioning. For a more detailed explanation, please refer to [HWLC97].

2.1.1 Basic Satellite Navigation

The basic concept of satellite navigation systems is very straightforward: as the name suggests, GNSSs employ a set of satellites to determine the position of a measuring unit. These *navigation satellites* circuit Earth on well-defined orbits and emit radio signals that carry the navigation information. The most important pieces of information are the time at which the signal is sent and the satellite's orbit parameters (the *ephemeris*) that determine the satellite's position, with respect to a transmission time. A measuring unit then needs to receive the signals of at least four navigation satellites to determine its own three-dimensional position by means of *pseudoranging* and *multilateration*: it calculates the time that the signal has traveled, the *Time Difference of Arrival (TDOA)*, and uses it, in conjunction with the speed of light, to determine its distance (pseudorange) to each satellite. Once the distances are known, the receiver constructs spheres around the satellites' positions for the multilateration, using the determined respective distances as radii. In a perfect setting, these spheres would intersect in a single point, i. e., the receiver's position. [Str95, HWLC97]

As straightforward as this description may seem, it is more complex when observed more closely. Several effects almost certainly cause the spheres to not perfectly intersect in one single point, caused by approximation errors in the pseudoranging (*User Equivalent Range Errors (UERE)*): as described, a receiver needs to determine the distance to the navigation satellites by calculating the TDOA. For this purpose, both the satellites and the receiver require highly accurate clocks. The clocks that are built into

Segment Source	Error Source	GPS 1σ error (m) estimation	conservative est.
Space	Satellite clock stability	3.0	3.0
	Satellite perturbations	1.0	1.0
	Other (thermal radiation, etc.)	0.5	0.5
Control	Ephemeris prediction error	4.2	4.2
	Other (thruster performance, etc.)	0.9	0.9
User	Ionospheric delay	5.0	10.0
	Tropospheric delay	1.5	2.0
	Receiver noise and resolution	1.5	4.8
	Multipath	2.5	1.2
	Other (interchannel bias, etc.)	0.5	0.5
System UERE	Total (root sum square)	8.0	12.5

Table 2.1: GPS C/A-Code Pseudorange Error Budget [VD00].

the navigation satellites provide such a high absolute correctness, but they are very expensive, complex and huge, so most receivers need to cope with much simpler and smaller clocks that still feature a high relative accuracy. The receiver's clocks therefore often implement methods to iteratively improve the correctness of their clocks and thus improve the positioning accuracy. However, even the satellites' clocks may suffer from slight drifts that can impede an exact TDOA measurement. Additionally, the satellite signals do not travel at a constant velocity, because they traverse different layers of Earth's atmosphere, such as the ionosphere and the troposphere. Depending on a satellite's position, the length of its signals' way through these layers may differ significantly and therefore may cause *atmospheric errors* in the pseudorange calculation. As a last example of error sources, satellite signals are subject to the same multipath effects, as they are also known from radio signals used for wireless networking with IEEE 802.11 or 3G/4G communication. As a result of these inaccuracies in the pseudoranging, the spheres around the satellites do not intersect in a single point, but the receiver needs to approximate the position as the point with the minimal distance to all spheres. In general, one refers to the final positioning inaccuracy caused by these error sources as *position measurement noise*. Table 2.1 gives an overview of 1σ UERE error values for the GPS navigation system, as published in [VD00].

2.1.2 Applications and Requirements

Commercial Systems

An important branch for spatio-temporal use cases compasses commercial systems, with the prominent example of fleet management: vehicles are equipped with GPS receivers and communication devices and report their position to a central station, as in [THR07]. Here, the focus lies on the timeliness of the data, i. e., the position information maintained in the central station must be as current as possible, so the fleet management can utilize vehicles at locations close to their respective current positions. This is especially essential for police patrol car or taxi management [KWRKM05, STBW02]. A spatially high resolution is not crucial in these cases, as the reported position is most likely to be projected on map material, where a position error of several meters is mostly tolerable.

In the area of vehicular communications, application scenarios are considered that aim at improving the safety, efficiency and the comfort of traffic. To achieve these objectives, the vehicles exchange *Floating Car Data (FCD)* or *eXtended Floating Car Data (XFCD)* [HLO99] that basically contain detailed spatio-temporal information, enhanced with additional sensor data in the case of XFCD. The applications that are realized upon the exchange of such data are diverse and cover, for example, road weather monitoring [MPS02] and map refinement or generation applications [SWR+04, BEJS05, KP08]. While the first requires a timely event report and merely a rough localization, the characteristics of the latter applications are quite the opposite: *probe vehicles* act as measurement entities and are supposed to collect their spatio-temporal trajectory data at a high accuracy, but may transmit it at a later point in time.

Research Assistance

In the research sector, position measurements are also used very often, especially to monitor longsome evolutions, where measurement points might be hard to reach or might be spread over a large area, such as for tectonic plate movement measurements for earth quake research [DAC+08]. For such measurements, intervals in the range of seconds are unnecessary, while the measurement accuracy is more important. For the tracking of flocks or herds, on the other hand, position inaccuracies of up to ten meters are tolerable at varying measurement intervals, depending on the achievable velocity of the tracked unit [JOW+02, SBM+07].

Community Applications

The impressive development is best illustrated by the spreading of applications with positioning requirements in non-commercial communities or recreational contexts. While semi-professional application visions, such as peer-to-peer network based traffic information systems [RSKM09] feature requirements comparable to commercial systems, the requirements of applications from other backgrounds may vary heavily. In sports communities, such as *smartrunner* [sma], applications are likely to be delay-tolerant and focus on logging trajectories without hard accuracy bounds, as they can be corrected with or mapped onto map material afterwards. Other use cases include the recovery of broken-down model aircrafts, which again demands a higher position resolution.

We see that applications with spatio-temporal backgrounds are used in variety of contexts. Although these contexts may be diverse, so are the requirements of the respectively covered applications, regarding spatial and temporal accuracies. Techniques that are to be applicable in a wide range of use cases should therefore provide the possibilities to adapt to a wide range of requirements.

2.2 Setting up a Measurement Pool

In this thesis, we present and discuss algorithms for the compression of spatio-temporal trajectories. Like other algorithms that have a strong binding to real-world measurements, these need to be tested and validated with suitable data sets. To ensure that these algorithms perform well in realistic scenarios and to gain reliable and representative results, it is important to work with a real-world measurement pool instead of purely simulative models. Such models merely resemble the effects that may occur and may neglect others which could result in a biased algorithmic design or in an evaluation that does not show the true behavior of an algorithm in a realistic environment. In this section, we therefore explain how the measurement pool is set up that is used throughout this thesis.

2.2.1 Collecting Trajectories

For our collection of real-world trajectories, we accessed the data base of the *OpenStreetMap (OSM)* project [proe]. In this project, volunteer contributors and donators work together to create a world map that is free for use, distribution and altering, licensed under an open data license, the *Open Data Commons Open Database License (ODbL)* [ope]. The map generation process can roughly be split up into the *data col-*

lection, *uploading*, and the *editing*. For the data collection, contributors mostly record movement trajectories (or *tracks*), e. g., with GNSS receivers. Alternatively, tracks can be retrieved as donations from municipalities or from other sources that are compatible with the ODbL. The data is then uploaded to the project website and thus made available to all other contributors and the public. Finally, the raw data can be edited and integrated into the map material. In this step, special *tags* can be defined for new or existing street segments that, for instance, give information about whether the segment belongs to a highway or to a pedestrian precinct.

For a meaningful evaluation of the algorithms discussed and developed in this thesis, a data base with a large number of trajectories is required. The OSM project does not provide an easy-to-use interface over which the raw measurement tracks could be retrieved: Instead, the project website is organized with a page-based download section, thus making a certain number of tracks accessible per page. To automatically access the download section, we implemented a web crawler that scans through the pages and downloads the raw data.

As already mentioned, the data on the OSM website is potentially recorded with a wide spectrum of positioning devices, so the traces themselves also feature a variety of measurement frequencies, detail degrees and accuracies. Though the concepts in this thesis are applicable to trajectories that originate from all sorts of movements, we focus on vehicular trajectories that meet the requirements of high-end commercial applications that basically request a high data resolution. We therefore limit the data basis for our studies to contain only trajectory descriptions that have been recorded at a fixed measurement frequency of 1 Hz. This is provided even by current off-the-shelf GPS hardware, while more sophisticated positioning systems that use e. g., Kalman filters [Bro98] or inertial navigation systems [Woo07], can easily accomplish this task anyway. We are also only interested in trajectories with at least 100 measurements. To odd out trajectories that are unlikely to originate from motorized vehicles, we implemented a rough pre-filter that dropped all trajectories with a majority of their velocities being below a threshold of $8.3\,{}^{m}/_{s}$; in this way, a total trajectory set S_0, containing 20191 trajectories, was collected.

For our evaluations in the next chapters, three different types of trajectory sets are needed: for the algorithms with a high runtime complexity that are examined in Chapter 3, we need a set of short trajectories and choose with our experience from previous publications [KKKM11, KBMS11] a trajectory length of 250 elements for this set, S_1. However, the trajectories for S_1 are not directly obtained from the OSM website, but extracted as subsequences from the OSM trajectories, instead: we initialize a sliding window with size 250 and place it at the beginning of longer trajectories. In each

step, the content of the window is then stored as a new trajectory and the window is shifted forward by 125 measurements. Thereby, the influence of edge effects due to disadvantageous window positions is weakened. For the longer trajectories, we use all trajectories with a measurement count between 1000 and 1300 that were obtained with the crawler. This window approach further simulates a realistic data collection policy: position measurements are pushed into a buffer and once the buffer is full, its content is passed to the compression algorithm, the buffer is cleared and the data collection starts over. In contrast to that, we assemble a set of uncut trajectories, S_2, for the evaluation of algorithms that feature a comparably low runtime complexity. These trajectories are directly taken from the data that the crawler has collected. Further, both the sets S_1 and S_2 are split into subsets according to special movement characteristics. The determination of these characteristics and the selection process based on them is described in Subsection 2.2.2. Finally, we regard a compression techniques that need to absolve *training phases* before being applied on movement data. Therefor, a set S_3 is prepared that is composed of two subsets: a *training set* S_3^{train}, and a *test set* S_3^{test}, the composition of which is described in the next subsection.

2.2.2 Categorizing Movements

In general, compression algorithms aim at identifying and removing redundancy of a data set and thereby optimizing the data encoding. We are focusing on vehicular trajectories, so we need to regard kinematic factors that determine the movements of the measurement units and examine in how far these factors influence the performance of the compression algorithms. In a next step, we can then cluster the trajectories from our measurement pool according to these factors and analyze the regarded algorithms' compression performance for these clusters.

The procedure of analyzing the nature of an object's movement has been intensively studied before. In the field of context recognition, one prominent challenge is to detect the means of transportation on the basis of the moving objects' trajectories and other sensor data. In doing so, many sophisticated approaches have been regarded that all provide satisfactory results: in [LPFK05], the authors use a Dynamic Bayesian Network (DBN) to learn and estimate a tracked person's location and activity, and Zheng et al. propose to split trajectories at points of halts and to process the separate segments separately in [ZCL+10]. Therefore, they analyze the nature of walking (especially low velocities), car driving (higher velocities, limited heading changes), bus driving (like car, but more frequent stops), and cycling (moderate velocity, more irregular heading changes) and use these, together with transport mode transition probabilities

to determine the transportation mode. In [RMB+10], Reddy et al. follow a similar philosophy in using a moving object's velocity, acceleration and a transportation mode transition probability model in the form of a preliminarily trained hidden Markov model to infer whether the movement is by walking, cycling or driving with a motorized vehicle. They also show that especially the velocity is a reliable criterion to make a classification decision. This is also highlighted in [SA09], as the authors set up a classification system purely based on the maximum and average velocity and the acceleration derived from GPS measurements. In addition to the information that can directly be gained from the positioning and from the acceleration sensors, *Geographic Information System (GIS)* data bases that contain map information are also used for classification purposes, such as in [BLvO12].

The above-mentioned excerpt of relevant literature makes clear that the question for the transportation mode context can be answered in a very high degree: the accuracies of the proposed algorithms vary between 70 % and more than 99 %, a good and detailed overview can be found in [BLvO12]. However, to achieve such a high accuracy, most algorithms make use of at least the velocity and the measured acceleration. In our case, it is even debatable whether the question for the actual transportation means fully applies; we aim at only regarding vehicular traces and in accordance with the above-mentioned publications, we argue that in the vehicular context, velocity is the most influential factor on a vehicle's movement and kinematic characteristics. Furthermore, there are no acceleration measurements attached to the trajectories in our data basis and we would limit the generality of our approach if these sensor data would be added to the required inputs. Also, it is not clear to be a good idea to calculate the acceleration as the second derivative over time from the positioning data, because the result is likely to be noisy—we will learn more about the influence of this noise in Sections 3.4 and 4.6. The authors of [SA09] follow this approach nevertheless and use a Gaussian smoothing for the acceleration and employ the result in a *fuzzy logic* model. They achieve good results, but their fuzzy rules indicate that the acceleration is mainly used as an indicator for the determination of what kind of motorized vehicle is used.

In summary, instead of using sophisticated solutions that need further work in calibrating or training a model, we differentiate trajectories along with their velocity profiles: we set up two velocity classes for low and high velocities and follow international velocity guidelines in setting up the threshold between these two classes. The low velocity class covers the velocity interval $[8.3; 16.7)\,{}^{m}/_{s}$, and the high velocity class covers all velocities in $[16.7, \infty)\,{}^{m}/_{s}$. Thereby, the low velocity class should roughly cover urban movements with lower velocities and narrower corners, and we expect the high velocity class to contain trajectories that mostly cover extra urban move-

Set	Description	Measurement frequency	Trajectory sizes	Set cardinality
S_1^{low}	cut, low velocity traces	1 Hz	250	5536
S_1^{high}	cut, high velocity traces	1 Hz	250	4599
S_2^{low}	uncut, low velocity traces	1 Hz	≥ 100	3257
S_2^{high}	uncut, high velocity traces	1 Hz	≥ 100	6839
S_3^{train}	uncut, training traces	1 Hz	≥ 100	10095
S_3^{test}	uncut, validation traces	1 Hz	≥ 100	10096

Table 2.2: Trajectory sets in our measurement pool.

ments that took place on highways, for example. We classify a trajectory to one of the two classes, according to which of the two intervals covers more of the respective trajectory's velocities.

Table 2.2 summarizes the properties of the compiled trajectory sets in our measurement pool. We applied our classification first to the total trajectory set S_0. As a result, 6513 trajectories were classified as low velocity, and 13678 as high velocity trajectories. These sets were then randomized and split in half, using one half of each as S_2^{low} and S_2^{high}. Then, we set $S_3^{\text{test}} = S_2^{\text{low}} \cup S_2^{\text{high}}$ and $S_3^{\text{train}} = S_0 \setminus S_3^{\text{test}}$. The trajectory sets S_1^{low} and S_1^{high} were finally calculated from the trajectories with 1000 up to 1300 positions from S_2^{low} and S_2^{high}, respectively, following the sliding window approach as explained above.

3

Geometric Trajectory Point Reduction

3.1 Introduction

The most wide-spread class of trajectory compression techniques covers *geometric* methods. These methods employ geometric structures, such as polygonal curves or circular arcs for the compression process. There are some advantages to such techniques over more abstract methods like those that are discussed in Chapter 4: not only are geometric approaches rather concrete, because it is easier to follow the algorithm on a visual basis. Additionally, it is often beneficial to store a geometric context—position coordinates of a simplified trajectory, for instance—when this form of information is easier to process by a particular application. Fleet management systems, such as the one being used in [GSTW04], could maintain a database containing the fleet vehicles' trajectories for various request types. Now, for an exemplary use case, the user could be interested in a specific vehicle's trajectory for a particular time period to evaluate the route selection from the starting point to the destination. Alternatively, the fleet vehicles' current positions might be of interest to estimate remaining travel times or the time until the fleet will return to the base. In both cases, it is very easy to work with geometric information rather than with binary data streams that would need to be decoded initially.

Geometric trajectory compression methods often work by means of trajectory approximations that can be encoded with a reduced point set. To obtain such sets, one basically attempts to find a geometric curve that approximates the particular trajectory within a specific error corridor defined by an error tolerance ϵ. The reduced set shall contain only those trajectory points or other parameters that are necessary to

reconstruct this geometric curve and thus to retrieve the omitted points, typically by means of simple interpolation.

In this chapter, geometric compression approaches are discussed and their performance is evaluated. To establish a sound basis for our argumentation, we first present a clear formulation of the problem statement, and review related work. Since a majority of the contributions from the related work employs linear approaches, we directly apply two representative linear compression algorithms to our trajectory data basis to establish a benchmark to which the new algorithms that are described in this work can directly be compared. We then increase the algorithmic complexity by turning to nonlinear methods, employing clothoids and circular arcs and finally regard a solution based on cubic splines. For each approach, it is also defined how the compression ratio is calculated, so that the compression techniques can be easily compared among each other in the evaluation sections.

3.2 Problem Statement and Notation

For the geometric compression methods, we consider both linear and nonlinear approaches, and therefore need a clear problem statement that is applicable in both cases. This statement is kept universally, so that it can be easily applied it to all regarded approaches, thus establishing a comparability between them.

In the following, we will denote a movement trajectory or position measurement trace as a sequence $\langle m_j \rangle_{j \in J}$, mapping the elements of an (ordered) index set J to d-dimensional *measurement tuples*:

$$m : \ J \to \mathbb{R}^d, \quad j \mapsto m_j = (a_0, \ldots, a_{d-1}) \ .$$

The discussed techniques are more generally applicable, but for the sake of simplicity we focus on the two-dimensional case of geographic coordinate pairs here:

$$m : \ J \to \mathbb{R}^2, \quad j \mapsto m_j = (x_j, y_j) \ . \tag{3.1}$$

Furthermore, we assume that a metric accuracy bound $\epsilon \geq 0$ is given.

Then, generally speaking, we aim to find a compact representation in the form of another point sequence $\langle m'_j \rangle_{j \in J'}$ that allows us to reconstruct all elements from $\langle m_j \rangle$ with a maximum error of ϵ; $\langle m'_j \rangle$ does not necessarily need to be a subsequence of $\langle m_j \rangle$. The reconstructed element sequence will be references as $\langle \widehat{m}_j \rangle_{j \in J}$ and in analogy

to (3.1), the sequences $\langle m'_j \rangle$ and $\langle \widehat{m}_j \rangle$ are defined as

$$m' : \ J' \to \mathbb{R}^2, \quad j \mapsto m_j = (x'_j, y'_j) \ ,$$

$$\widehat{m} : \ J \to \mathbb{R}^2, \quad j \mapsto m_j = (\widehat{x}_j, \widehat{y}_j) \ .$$

To find $\langle m'_j \rangle$, we use different, purpose-tailored compression and decompression algorithms; let us denote such an ensemble by $E(c, r, \epsilon)$. Here, c denotes the compression mapping:

$$c : (\mathbb{R}^2)^{|J|} \to \mathfrak{P}(J) \times (\mathbb{R}^2)^{|J'|}, \quad c(\langle m_j \rangle) = (J', \langle m'_j \rangle), \quad |J'| \leq |J|$$

with $\mathfrak{P}(J)$ being the power set of J. The corresponding decompression mapping is denoted by r:

$$r : \mathfrak{P}(J) \times (\mathbb{R}^2)^{|J'|} \to (\mathbb{R}^2)^{|J|}, \quad r(J', \langle m'_j \rangle) = \langle \widehat{m}_j \rangle \ .$$

so that the sequences $\langle m_j \rangle$ and $\langle \widehat{m}_j \rangle$ do not differ by more than ϵ element by element corresponding to a specific distance metric. As we are dealing with metric positions, we use the Euclidean distance metric; formally,

$$\forall j \in J : \sqrt{(x_j - \widehat{x}_j)^2 + (y_j - \widehat{y}_j)^2} \leq \epsilon \ .$$

Different versions for the compression and decompression mappings c and r will be presented in the following sections.

Under this notation for geometric compression schemes, the *compression ratio* σ is defined by

$$\sigma = 1 - \frac{|J'|}{|J|} \ , \tag{3.2}$$

as long as the mapping r does not require additional data for the reconstruction.

3.3 Related Work

The previous work published for linear and nonlinear geometric compression methods represents the main previous work in the area of trajectory compression. We therefore review these contributions in-depth in this section. As many approaches employ linear simplification, we first present two representative algorithms and evaluate the compression performance that they achieve and that we use as a benchmark for the nonlinear approaches.

3.3.1 Line Simplification Algorithms

Optimal Line Simplification

A line simplification algorithm that finds the minimal subsequence of $\langle m_j \rangle$, where the minimal subsequence approximates $\langle m_j \rangle$ with a maximal approximation error ϵ as defined above is said to solve the *minimum subsequence approximation problem* or *minimum-# problem*; in our notation, this minimal subsequence represents the sequence $\langle m'_j \rangle$. Such an algorithm has been proposed in [II86]: in a first step, the algorithm creates a directed graph $G(V, E)$, where V includes all nodes from the original polygonal curve, $\langle m_j \rangle$. An edge $e = (i, j) \in E$, iff $i < j$ and the distances of $m_{i+1}, \ldots, m_{j-1}$ to the line segment between m_i and m_j are smaller than ϵ. Formally,

$$\forall k : i < k < j : d_{l_{i,j}}(m_k) \leq \epsilon \ .$$

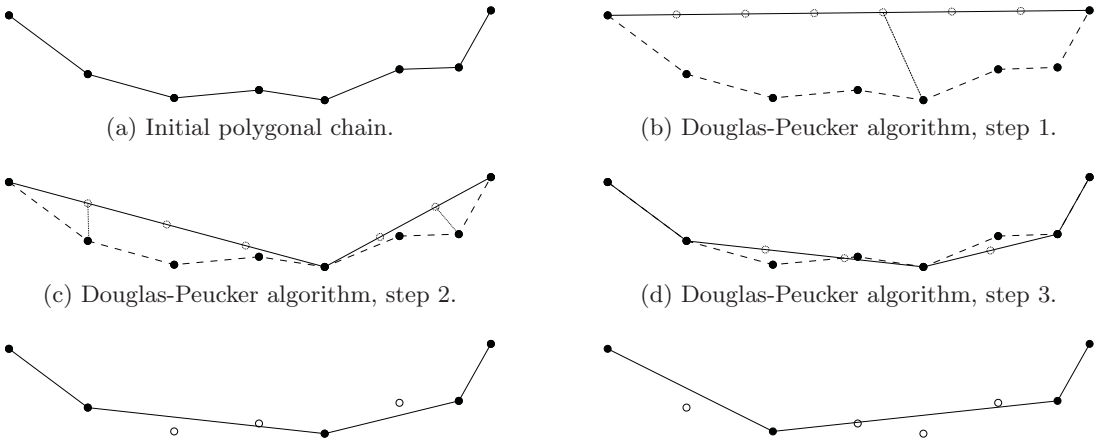
with the distance metric $d_{l_{i,j}}(\cdot)$ of a point to the line segment from m_i to m_j. If only the shortest distance to the line matters, then an orthogonal projection can be used that determines the shortest distance to the line. As we deal with spatio-temporal trajectories, though, we need to consider that omitted points will be recovered by mere linear interpolation. Therefore, if a line segment of length l bypasses b points, our distance metric determines the Euclidean distance from the i-th bypassed point to the position on the line segment with the proportional offset $\frac{i \cdot l}{b+1}$. Once this graph has been constructed, the minimal subsequence is determined by finding the shortest path from m_0 to m_{n-1}, for example with the *Dijkstra* shortest path algorithm [Dij59].

As shown in [FT84], the shortest path problem can be solved in $\mathcal{O}(|V| \log |V| + |E|)$, if Fibonacci heaps are used. However, the creation of G, using a brute force algorithm as originally proposed, runs in $\mathcal{O}(|E|^3)$. The latter runtime complexity could be improved to $\mathcal{O}(|E|^2)$ in [CC92], so that the overall complexity of the algorithm is $\mathcal{O}(|E|^2)$.

With the optimal linear approximation, the compression mapping c is the simplification process itself. For the decompression mapping r, the omitted nodes from $\langle m_j \rangle$ can be approximated by linear interpolation, for which reason the indexes of the nodes need to be preserved.

Douglas-Peucker – A Heuristic Approach

The Douglas-Peucker algorithm [DP73] implements a line simplification heuristic and finds a subsequence of a given polygonal curve $\langle m_j \rangle$ with a maximal approximation error ϵ using a divide-and-conquer strategy: first, the first and last node of a polygonal chain are taken as end points of a line segment. For all nodes in between, the distances



(a) Initial polygonal chain.
(b) Douglas-Peucker algorithm, step 1.
(c) Douglas-Peucker algorithm, step 2.
(d) Douglas-Peucker algorithm, step 3.
(e) Douglas-Peucker algorithm, step 4.
(f) Optimal line approximation result.

Figure 3.1: Exemplary steps for line approximation algorithms.

to this line segment are determined with respect to a specific distance metric. Here, the above discussion on the distance metric applies as well. If any distance exceeds the accuracy bound ϵ, the polygonal chain is separated in such a way that the point with the greatest distance is both the end and the start node for the new created subchains, respectively. Then, the algorithm starts anew for each of these subchains and terminates, once no point-line distance exceeds ϵ any more.

Figures 3.1a–3.1e show a simple example for a polygonal chain consisting of eight nodes: the original polygonal curve is separated (Figure 3.1c) and the two resulting subchains are each separated as well (Figure 3.1d). As the result of the algorithm, the remaining nodes set up the sequence $\langle m'_j \rangle$. Please note that in this example, the proportional distance metric from above has been used and the dotted circles on the line segments denote the proportional projections (and thus the potential interpolations) of the currently omitted points. As mentioned before, the Douglas-Peucker algorithm merely employs a heuristic, i. e., the result $c(\langle m_j \rangle) = \langle m'_j \rangle$ is not necessarily the global optimum. Its advantage, on the other hand, is its runtime complexity, which lies in $\mathcal{O}(nk)$ with $n = |J|, k = |J'|$ in a simple implementation, but can be improved for two-dimensional polygonal curves to $\mathcal{O}(n \log n)$ [HS92]. Finally, Figure 3.1f shows a possible result of the optimal line simplification algorithm for the example polygonal chain from Figure 3.1a. It clearly differs from the result obtained with the Douglas-Peucker algorithm, and is, in this example, slightly more compact.

For the Douglas-Peucker algorithm, c and r are defined by analogy with the optimal line simplification algorithm: the compression mapping c is the line simplification itself and the decompression mapping r is realized by means of linear interpolation.

To reconstruct the removed points from $\langle m_j \rangle$ at the correct positions, the original indexes need to be stored. In literature, this is achieved by encoding temporal context data either explicitly with timestamps or implicitly with sequence numbers for periodic update protocols. Since we handle periodic position measurements, as pointed out in Section 2.2, we can reduce the overhead even further by using a bit field of length $|J|$ for this purpose. In such a bit field, a bit indicates whether the position measurement at the respective position has been kept. This implies an extra effort of merely one bit per measurement.

Additionally, we need to store the point of time for the first position measurement; in combination with the bit field—that is in this context not only an overhead for the spatial reconstruction—we can reconstruct the temporal information of the trajectory. This allows, for instance, to conclude on a passed road segment's position within a fundamental diagram of traffic flow or to monitor the average speed for it as in [STBW02]. A fundamental diagram correlates the traffic flux (vehicles per hour) and the traffic density (vehicles per kilometer) for traffic flow estimations.

Evaluation

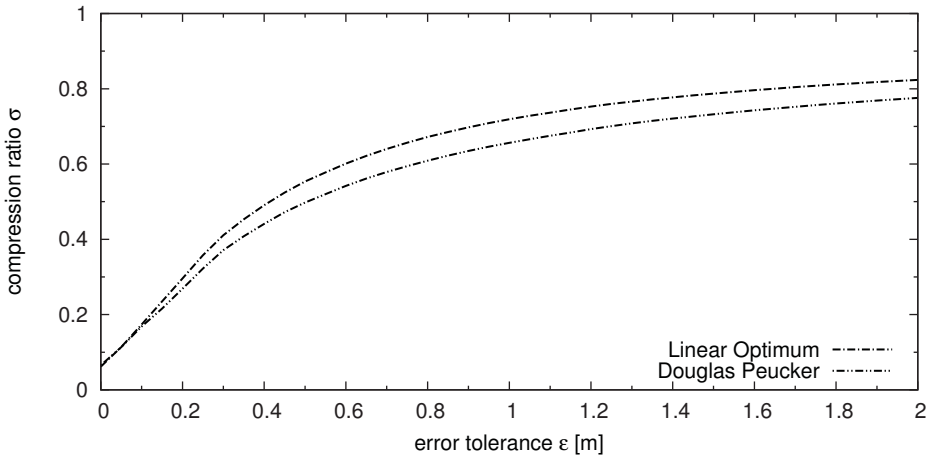
A short note on the compression overhead For the evaluation of the line simplification compression schemes, we need to regard the overhead for the compression performance that is caused by the discussed bit field. To achieve effective data compression, enough positions need to be removed from $\langle m_j \rangle$, so the sum of the compression result size and the bit field size is less than $|J|$, and thus to compensate this overhead. The actual amount of nodes that need to be removed depends on the size of the binary representation for each position, $2 \cdot s_n$, the overhead size $s_o = \left\lceil \frac{|J|}{8} \right\rceil$ (both given in bytes), and the reduced knot sequence length $|J'|$. Given these parameters, we can formally describe that the size of the reduced node sequence and the overhead need to be smaller than the original sequence's size:

$$2 \cdot s_n \cdot |J| > 2 \cdot s_n \cdot |J'| + s_o \ . \tag{3.3}$$

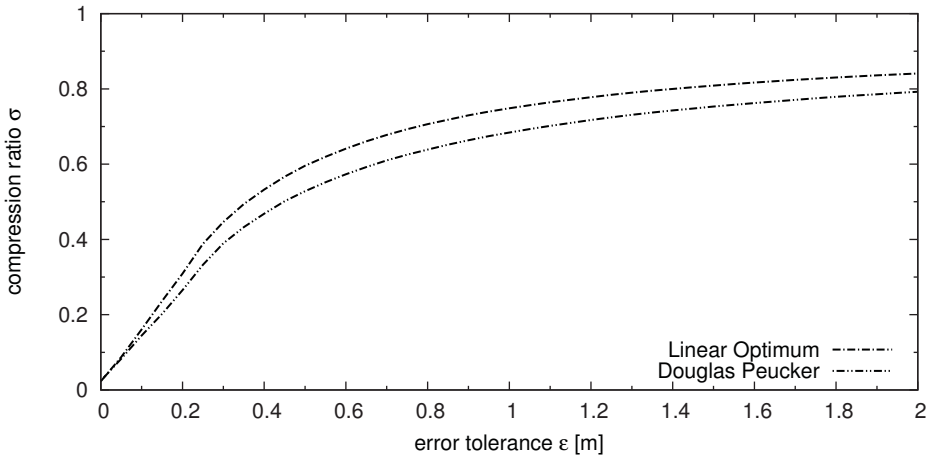
From this, we can derive the absolute threshold that needs to be underrun for the particular line simplifications to result in successful compressions:

$$|J| - |J'| > \frac{s_o}{2 s_n} \tag{3.4}$$

For the evaluation, we use trajectories with $|J| = 250$ elements, and assume a binary position size of $2 \cdot s_n = 8$ bytes. This encoding results in a maximal initial representation



(a) Low velocity trajectories (S_1^{low}).



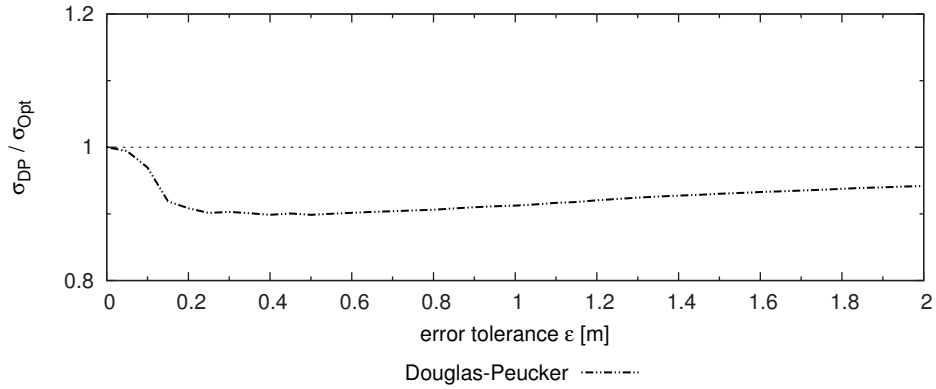
(b) High velocity trajectories (S_1^{high}).

Figure 3.2: Line simplification: compression analysis.

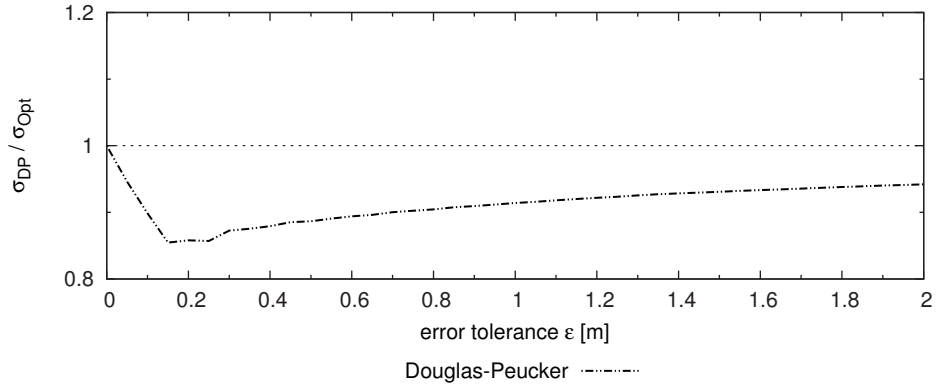
error of $\approx 0.7862\,\text{cm}$ for any geodetic coordinate, which we accept as insignificant. Under these conditions,

$$\begin{aligned} |J| - |J'| &> \frac{32}{8} &&= 4 \\ \text{and} \qquad \frac{|J'|}{|J|} &< 1 - \frac{32}{8 \cdot 250} &&= 0.984\ . \end{aligned}$$

This means that for this configuration, at least 4 positions need to be removed in the simplification process and the raw compression ratio, as defined in (3.2), of $\sigma = 0.016$ needs to be exceeded to achieve an effective compression.



(a) Low velocity trajectories (S_1^{low}).



(b) High velocity trajectories (S_1^{high}).

Figure 3.3: Line simplification: relative compression analysis.

Compression Performance The compression results for the linear approaches, calculated according to Statement (3.2), are interpreted as mapping of the error bound ϵ to the mean compression ratio achieved under this condition. This mapping is shown in Figure 3.2 for the low (S_1^{low}) and high (S_1^{high}) velocity trajectory sets. Both approaches benefit particularly from error tolerances of up to 50 cm, as the higher slopes of the compression performance curves indicate. For larger ϵ, the curves flatten and error tolerances of more than 1 m have merely a minor impact on the compression ratio. This behavior can be explained with the noise that underlies the position measurements. Once the error bound is larger than the mean positioning noise, the algorithms can approximate the trajectories with higher tolerances, so the trajectories can be encoded more efficiently. The figures do not provide information about the accuracy of the mean value, because the 95 % confidence intervals for both approaches have an average width of approximately 0.6 % and would not even be visible in the figures.

The performance of the Douglas-Peucker algorithm comes very close to the upper bound set up by the optimal line simplification. Figure 3.3 depicts the compression performance of the Douglas-Peucker algorithm relative to the optimal line simplification: $\frac{\sigma_{\mathrm{DP}}}{\sigma_{\mathrm{Opt}}}$. Overall, for $\epsilon > 0.1\,\mathrm{m}$, the Douglas-Peucker algorithm performs approximately 5 % to 15 % worse than the optimum in both the low and the high velocity settings.

3.3.2 Linear Trajectory Compression Approaches

Linear trajectory compression methods have almost exclusively been presented in the fields of *Mobile Objects Databases (MODs)* and mobile tracking [CLFG+03, GS05]. Such databases are employed for the tracking of mobile objects as discussed above in the exemplary use cases: for these, all participating mobile units send position updates to a server hosting an MOD; again, depending on the application or the use case, such update transmissions may be triggered by different circumstances. In the simplest case, a mobile node sends update notifications periodically. In more complex scenarios, mobile nodes reconstruct the tracking information base that is maintained on the server side and send update messages once it is recognized that the server can no longer approximate the mobile node's position within the afore-mentioned upper uncertainty bound ϵ.

In general, mobile objects can perform the position updates in an *offline* or *online* fashion, according to the requirements of the particular application. *Offline* approaches are applied in the event that it is unnecessary for the database or the tracking unit to be accurately current: a mobile node collects its trajectory data until, for instance, a spatial or temporal threshold has been exceeded since the most recent update. Then, the node compresses the trajectory data using its *post-factum* knowledge. This can be achieved using *linear dead reckoning (LDR)* mechanisms; with LDR, the movement of an object is linearly extrapolated on the basis of past position measurements. This kind of movement estimation is very simple but effective, especially if the position measurement frequency is reasonably high. In literature, numerous publications on position tracking have made use of LDR: in earlier work, Wolfson et al. presented a cost model that took into account the communication cost, the tracking unit's uncertainty of the mobile unit's position and the LDR estimation error [WSCY99]. The authors of [LR01] focus on the efficiency of location information update protocols using LDR and analyze the minimum and mean tracking accuracy at the tracking unit. In [CJP05], the authors apply LDR tracking techniques in combination with shared map material and present an evaluation based on real GPS data. Other publications propose the compression of spatio-temporal trajectories by means of heuristic or opti-

mal line simplification algorithms that we discussed above. In [MdB04], Meratnia and de By propose spatio-temporal compression techniques and compare their performance to the performance of the Douglas-Peucker algorithm and so-called *Opening Window* algorithms. The latter basically perform ongoing line simplifications with an anchor node and the current measured position until the uncertainty bound ϵ is exceeded. The authors of [CWT06] define specific query classes, discuss and analyze several distance metrics for the Douglas-Peucker algorithm for spatio-temporal data reduction and focus on the aging of trajectories by allowing larger uncertainties for older trajectories. Motivated by this publication, Gudmundsson et al. improve and extend the proposed ideas in [GKM+07], so all query classes can definitely be answered regarding an uncertainty bound; finally, they present a modified Douglas-Peucker algorithm that performs better for self-intersecting trajectories. Trajectory compression can also be employed for data storage preprocessing, as proposed in [HGNMs08]: incoming position updates are buffered and the buffer is then compressed by offline compression techniques. Obviously, the line simplifications found by the optimal line simplification approach are upper bounds for all approximations calculated by the approaches based on linear movement modeling and LDR discussed here. We therefore do not evaluate each single approach from the related work but regard the results achieved with the optimal approach instead.

In contrast to that, *online* trajectory data reduction mechanisms aim at compressing collected position measurements in real-time, thus allowing for up-to-date spatio-temporal information at the tracking unit while reducing the network and communication load at the same time. For online trajectory data reduction, LDR mechanisms are widely used, because they are less complex than performing line simplification algorithms after each newly taken position measurement: Trajcevski et al. proposed the first online algorithm in [TCS+06] that employs LDR and regards a previously defined uncertainty bound ϵ. The authors of [LDR08] adapt the idea of using LDR for online trajectory reduction and present a *Connection-preserving Dead Reckoning (CDR)* that reduces the communication load and the remaining trajectory data volume. In [LFDR09], they subsequently present a generic tracking scheme by separating tracking from simplification and combine their approach with heuristic and optimal line simplifications. Finally, the authors of [HGRM10] compare simple line simplification algorithms to more complex algorithms and ones working with map material and conclude with recommendations for the use of algorithms with a view to several requirements.

In conclusion of the literature review, it can be stated that the offline linear algorithms generally achieve a better compression performance than online algorithms; on

the other hand, online algorithms allow for low communication loads at higher update frequencies, thus helping to save channel capacities, while providing more up-to-date position information in the tracking or storage unit.

3.3.3 Nonlinear Trajectory Compression Approaches

None of the discussed publications consider the use of nonlinear movement modeling or polyline simplification. The only previous work on trajectory data reduction with nonlinear functions has been presented in the context of spatio-temporal data base indexing: in [CN04], the authors motivate that a good representation for spatial vehicular trajectories are so-called minimax polynomials. These approximate an original function in such a way that the maximum approximation error is minimal for the given approximation parameters. They propose the use of Chebyshev polynomials that have been shown to be a very good approximation of optimal minimax polynomials. However, they focus on the spatial dimension of a trajectory and disregard the influence of time. The authors of [NR07] extend this work by focusing on the temporal dimension and optimizing the degree of the polynomials. Both contributions use the degree of the Chebyshev polynomial as input parameter and calculate the resulting approximation error after the calculation. They do not present an efficient way of constructing a Chebyshev or other polynomial representation with a higher-than-linear degree for moving object trajectories, for which the maximal approximation error does not exceed a previously set up upper bound ϵ.

3.4 Clothoidal Trajectory Approximation

In the previous section, we have seen that vehicular trajectories can be well-approximated with linear curves. However, we also have seen limitations of the regarded approaches, especially in low velocity environments, that are curvy rather than linear. On such road segments, more line segments are necessary to satisfy the accuracy bound ϵ. This, and the fact that vehicular movements are *not* linear, motivate the use of nonlinear geometric methods for trajectory compression schemes.

A promising approach for the compression of vehicular trajectories is to regard the design of the roads on which vehicles travel. Once it is determined of which basic geometric elements a road is composed, it can be assumed that any vehicular movements along such roadways can likewise be expressed or at least approximated by variations of these elements.

In the design and construction of roadways, linear segments, circular arcs, and clothoid segments—often referred to as *curve primitives*—are widely used to achieve

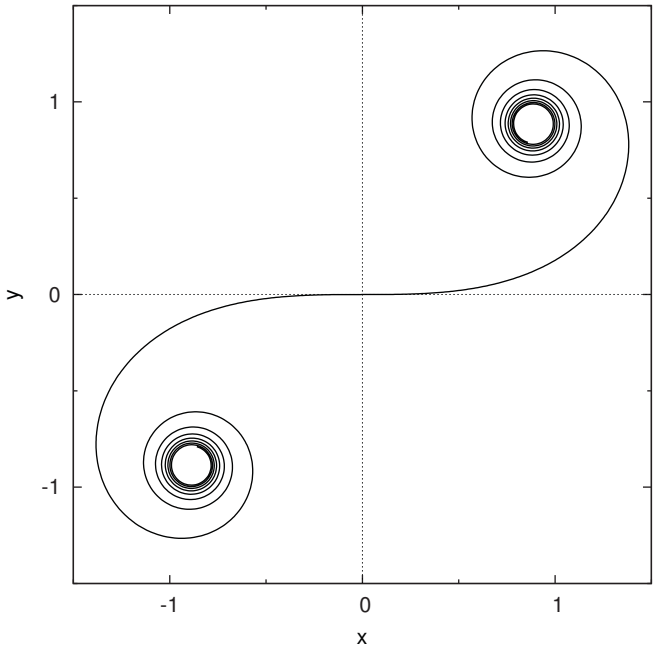
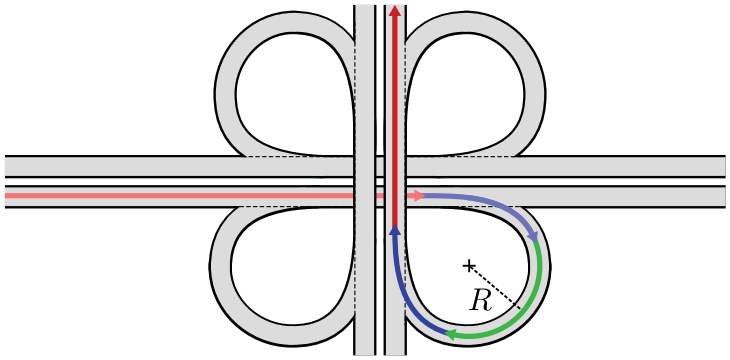


Figure 3.4: Unit clothoid.

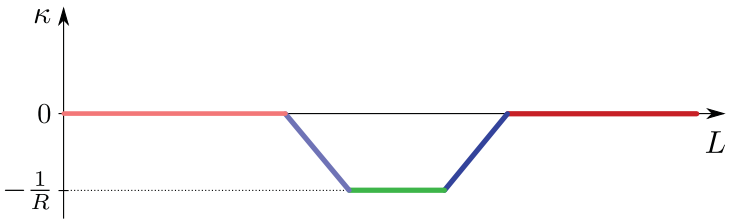
optimal trafficability [Baa84, MW92]. This is achieved, when the curve primitives are fitted together in such a way that vehicles can drive on a such constructed road smoothly, that is, without any jolts or shocks that would result from abrupt, but necessary steering changes. These curve primitives reflect the nature of fundamental vehicular movements as they can be experienced in everyday traffic: for yaw rates equal to zero, i. e., the steering wheel is constantly at a neutral position, a vehicle obviously moves ahead on a straight line. With a uniformly varying yaw rate, i. e., a constant steering wheel angle other than the neutral position, the vehicle's movement describes a circular arc with a radius determined by the steering angle. Finally, for the smooth transition between linear and circular movements or between circular movements with different curve radii, vehicles move on clothoidal curve segments. Clothoids are curves the arc length L of which is inversely proportional to the curve radius R, and thus proportional to the curvature κ:

$$L = A^2/R = \kappa/A^2 \ ,$$

where A is a constant scaling factor. Figure 3.4 shows the double-ended *unit clothoid*, i. e., $A = 1$.



(a) Roadway design.



(b) Curvature plot.

Figure 3.5: Roadway design using lines, circular arcs and clothoids.

The relation between road segments and the resulting steering behavior is shown in Figure 3.5: Figure 3.5a shows a part of a schematic, but typical highway interchange. An exemplary trajectory of a vehicle coming from the left and changing onto the highway segment towards the top is depicted and its curve primitive components are color-coded: linear sections are marked in red, clothoidal segments in blue and circular arcs are highlighted in green. Figure 3.5b plots the corresponding schematic curvature κ over the covered distance L: the road curvature starts with zero for the line segment, linearly decreases[1] to $\kappa = -\frac{1}{R}$ in the clothoidal section and then remains constant for the circular arc. Finally, the arc passes into a linear segment via a clothoid with a linear increase of the curvature up to zero.

As mentioned above, clothoid splines are used to guarantee the smoothness of a vehicle's drive along a roadway. In the following, we will express this smoothness with the help of mathematical continuity definitions: a smooth function is C^d (d-th order parametric) continuous, iff its first d derivatives exist and are continuous. Then, two curves are said to meet at a joint point with G^d (d-th order geometric) continuity, iff their parameterized functions meet at this joint point with C^d continuity [BD84].

[1] Left turns feature a positive curvature, whereas for right turns it is negative.

3.4.1 Clothoidal Curve Fitting

In [LNRL06], the authors focus on smooth trajectory planning for autonomous vehicles based on given map material. The map material uses a geometric description of the road segments in terms of straight lines and circular arc segments with given curve radii. Using these elements, Labakhua et al. propose fitting clothoidal splines to the map material by calculating the clothoid curvature and rotation parameters so that a vehicle can smoothly drive along its route with an optimal velocity profile.

This approach basically shows that a trajectory reconstruction with clothoid splines is indeed reasonable and feasible, although the preconditions are quite different to our situation: instead of highly-accurate map material, we are working with noisy position measurements that merely roughly describe the underlying road topology, if at all. We therefore need to find algorithms that approximate such polylines with clothoid splines instead of simply calculating transition curves between well-known line and circular arc segments.

Since the discussed curve primitives are uniquely defined by their curvature descriptions, a promising approach is to derive the sought-after curve primitive composition based on the collected way point measurements ($\langle m_j \rangle$) with a curvature analysis for this polygonal chain. We will next focus on two realizations of this idea from computer graphics and then work out how the compression and decompression mappings c and r in analogy to Section 3.2 can be implemented using these approaches.

Basic Clothoid Curve Sketching

In [MS09], McCrae and Singh present an approach for modeling piecewise clothoid curves (*clothoid splines*) to improve the *fairness* of polylines. Their motivation is to process input originating from free-hand drawings or other comparably noisy sources. They propose to approximate the curvature over the arc length by means of linear regression: for each pair of curvature points $\kappa_i, \kappa_j, i < j-1$, a linear regression and the resulting mean error is calculated. Then, using penalties for both regression errors and regression line counts, they minimize the number of resulting curvature segments and the overall regression line fitting error, thus obtaining a cost-optimal approximation of the curvature that may consist of multiple regression line segments. While the fitting error penalty is derived from the regression result itself, the penalty value for the number of additional regression line segments is an important accuracy parameter of the algorithm. With this curvature approximation, the authors attempt to remove most of the input noise in order to fair the data: once the characteristic curvature progressions are freed from noise, the curve primitives can directly be read off. This

simple linear regression will, however, rarely result in curvature approximations that perfectly indicate circular arcs or line segments; in fact, many regression line segments corresponding to noised arcs or lines will rather feature slopes close, but not equal to zero and will thus result in the construction of clothoids. To alleviate this, regression line segments with a slope beneath a threshold thresh_s are turned into parallels to the x-axis and those parallels with an offset to the x-axis beneath a threshold thresh_o are shifted onto the x-axis. After this step, the curve primitive parameters can easily be read from the curvature approximation. Using a smoothness criterion such as G^2 continuity for the complete spline, the primitives are aligned, concatenated and scaled to fit the original polyline with a minimal distance.

This approach works well for short polylines with weak noise levels, as they typically result from free-hand sketching. However, both higher noise levels and longer polylines affect the quality of the clothoid splines computed with this algorithm: first, the errors from the linear curvature regression are integrated twice during the computation of the primitive parameters, therefore having a great impact on the final curve. This effect is amplified by errors resulting from the curvature estimation process: though relatively good statistical curvature estimation methods for discrete curves exist, such as in [KSNS07], curvature calculation for noisy discrete curves still implies at least small errors, especially when the average noise is large in comparison to the distance between adjacent measurement points or even exceeds it. Finally, the clothoid spline, as created by the algorithm by McCrae and Singh, can not be modified once the curvature estimation is complete, regardless of the fitting accuracy. Thus, the twice integrated inaccuracies from the curvature regression accumulate over the arc length of the final spline and can result in high fitting errors that can not be corrected; this is a critical point especially for long and noisy polylines. Moreover, this basic approach does not provide a direct way to set up a threshold for the resulting fitting or measurement point approximation error as demanded in the problem statement. Instead, the only accuracy parameter is the penalty value for the number of additional regression lines for the curvature approximation. The second clothoid spline sketching approach does not only resolve the problem that a once constructed clothoid spline cannot be modified any more; it also provides a fitting error threshold parameter; this approach is discussed in the following.

Clothoid Curve Sketching using Shortest Paths

Shortly after the publication of McCrae's and Singh's algorithm, Baran et al. proposed another method to sketch a polyline with clothoid splines based on a shortest path

algorithm [BLP10]. Their algorithm is able to correct inaccuracies of a constructed clothoid spline and provides an approximation error threshold that is regarded during the sketching process. The approach of Baran et al. comprises a complex seven-step algorithm to fair polylines using lines, arcs, and clothoids: first, the algorithm *closes polylines* for which the distance between the first and last point falls below a certain threshold. Second, the algorithm splits the polyline at *corners*, i. e., positions with G^0 continuity, and processes each isolated sub-polyline separately. Each polyline is then *resampled* to adapt the point distribution to the curvature situation. As a result, the resampled curve features a higher point density in regions with a high curvature. Afterwards, the algorithm performs a *curve fitting* on the resampled point sequence: for every subsequence of points, curve primitives are constructed regarding an error tolerance. So, instead of simplifying the curvature to find one specific sequence of curve primitives as for the basic approach, a huge number of curve primitives are created that could possibly be used in the final spline. In the fifth step, a *directed, weighted graph* is constructed from the fitted primitives: for every curve primitive, a vertex is created; an edge is inserted between two vertices if the respective curve primitives can be interconnected. The edge weight indicates the quality of the transitions and depends on the fitting errors of the connected curve and possible transition primitives, and the curve primitive transition continuity (G^d: the greater d, the smaller the weight). Unlike the fourth step, the fitting errors caused from this step on are *not* ensured to stay within the error threshold; fitting errors due to new primitive transitions that exceed the threshold are merely penalized more severely to ensure that the primitives from the shortest path can definitely be connected. The *shortest path* through the graph is then calculated based on the edge weights and their transitions between the curve primitives are verified. Since the graph $G(V, E)$ is directed and acyclic, this can be achieved in $(O)(|E|)$ with a breadth-first search [Lee61]. Due to the cost function and depending on the choice of parameters, it is in fact possible that not only desired G^2 transitions are used, but also G^1 or G^0 transitions, in case these imply a lower fitting error. The found path contains the minimal sequence of curve primitives that approximate the given polyline. In the seventh step, the found curve primitives are finally *merged*.

This algorithm constitutes a more promising approach than the basic one presented in the previous subsection, because it does not completely rely on a curvature approximation and thus does not result in accumulating approximation errors. Instead, a sequence of curve primitives is found which takes the effective fitting error into account already during the calculation. However, despite this advantage this algorithm also can not *guarantee* compliance with the error threshold. Also, since our motivation

is to find a compression mapping, which is to be executed in mobile (e. g., vehicular) on-board units with limited computational capabilities, the high algorithmic complexity is a clear disadvantage. Furthermore, the algorithm has numerous parameters that influence performance and accuracy; the source code provided by the authors [Bar] defines more than 30, at least five parameters need to be chosen very carefully. These compass the G^d continuity costs, the approximation error costs and the approximation error threshold. Finally, both approaches suffer from the same problem as the linear approaches presented in Subsection 3.3.1: without any further information about the measurement point distribution along the curve, a uniform distribution has to be assumed, which is unlikely to match reality. While the line simplification methods can directly take the uniform distribution assumption into account, this is not possible for the clothoid sketching methods. So, even if the accuracy threshold were definitely not exceeded, assuming a wrong point distribution could easily cause offset shifts along the clothoid spline's arc, and would thus result in massive approximation errors.

3.4.2 Deriving a Clothoidal Compression Scheme

Although these clothoid sketching approaches originally serve the purpose to fair a noisy polygonal chain, and despite the discussed disadvantages, it is nevertheless possible to derive a simple compression scheme based on clothoid sketching. Basically, both approaches fulfill the formal requirements for the compression mapping c: given a polygonal chain $\langle m_j \rangle$, they compute a clothoid spline, i. e., a structure consisting of one or more G^1 or G^2 continuous sequences of parameterized curve primitives as compression result. In the following, we will only refer to G^1 continuous curve sequences, because every clothoid spline that can be constructed by the above algorithms can also be split at corners (G^0 continuous positions) into separate G^1 continuous curves. However, due to the severe shortcomings of the basic clothoid spline version, only the shortest-path approach is considered for a compression scheme.

To set up a compression scheme, we need to find a compact description for such clothoid splines. Since all curve primitives are well-defined by their respective curvature descriptions, we will use it as one component of the clothoid spline description: every edge point of the curvature graph is encoded as a two-dimensional point; in case of discontinuities, e. g., for merely G^1 clothoid splines, two points are assigned to the arc length, where the discontinuity occurs. Thus, in general, each of these G^1 continuous sequences can be uniquely identified by the start and end positions as reference points and a curvature graph: by means of the curvature graph, the curve primitives can be defined and concatenated. Once this is accomplished, the raw clothoid spline can be

scaled, rotated and shifted using the stored start and end points. We need to store the reference points for each transition with G^0 continuity, because in this case, adjoining curves' orientations are ambiguous and need to be stored explicitly by intermediate points. Given this information, the whole curve primitive spline can be restored, regardless of the continuity degree or what primitives the spline contains in detail.

For a simple encoding, we propose to store both the coordinates and curvature progression data elements as two-dimensional points. Without loss of generality, let us assume that a spline is found consisting of u primitive subsequences with G^1 continuity; furthermore, let the i-th subsequence consist of v_i primitives and let its curvature graph feature w_i discontinuities. Then, there need to be $u + 1$ necessary two-dimensional reference points and $v_i + w_i + 1$ points for the curvature graph description of the i-th subsequence. The overall compression result $\langle m'_j \rangle$ thus consists of

$$|J'| = (u + 1) + \sum_{i=1}^{u} (v_i + w_i + 1)$$

two-dimensional points. The compression ratio can therefore be stated as defined in Section 3.2.

The only missing information is the point distribution along the spline that approximate the original measurement tuples in $\langle m_j \rangle$. As mentioned before, an additional structure would be necessary, such as a mapping of the original measurements' indexes to the respective arc length. There are recent publications using such mappings, such as [Kro09], but their combination with the clothoid sketching approaches in a data compression manner is non-trivial and out of the scope of our research. We are aware, however, that the achieved compression ratios with this compression scheme serve as an upper bound due to the missing overhead.

3.4.3 Evaluation

To evaluate the compression performance of the presented clothoid spline approximation algorithms, we first briefly describe the implementation and parameter choices of the clothoidal compression scheme. We then analyze the effective fitting error that occurred during the compressions, because this may exceed the particular used uncertainty bound ϵ, as discussed in Subsection 3.4.1. Finally, we evaluate the compression performance of the clothoidal spline approximation.

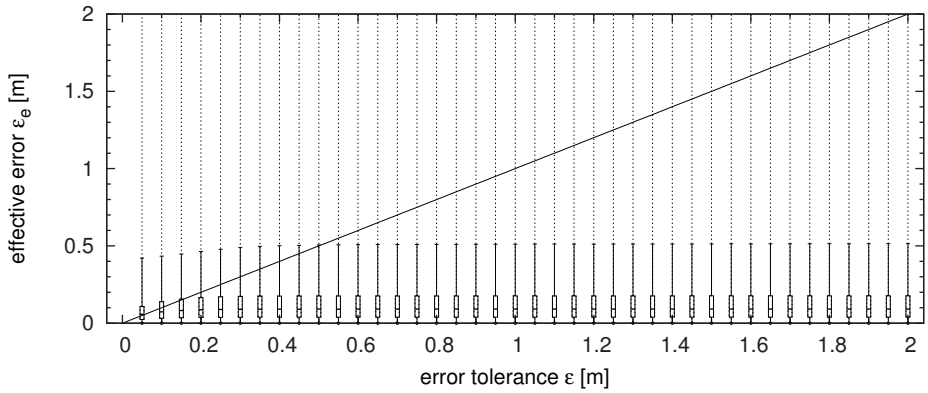
Parameter	Used value	Default
G^0 transition cost	101.0	∞
G^1 transition cost	51.0	∞
Error cost	52.0	5.0
Shortness cost	1.0	2.0
Points per circle	50.0	20.0
Scaling factor S	10.0	n.a.
Curvature estimation region	$S \cdot 10$	20.0
Error threshold	$S \cdot \epsilon$	4.0

Table 3.1: Cornucopia parameters as used in CornuConsole.

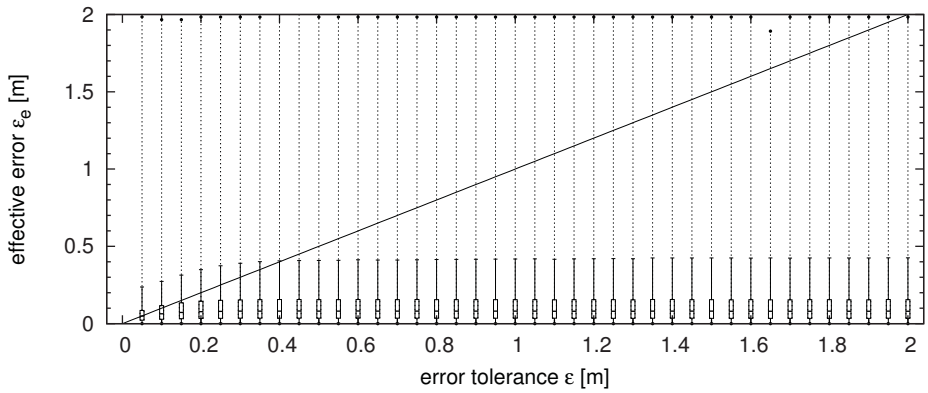
Compression Scheme Implementation

As mentioned earlier, the clothoid sketching software *Cornucopia* [Bar] that implements the shortest-path based algorithm from Subsection 3.4.1, requires approximately 30 parameters that are preset with default values. For our evaluation, we implemented a wrapper to Cornucopia, *CornuConsole* [Koe], which enabled us to perform the compression of the input data in a batch processing manner. As Cornucopia, CornuConsole is licensed under GPL3 [gpl]. To optimize the handling of vehicular movement traces, we made a number of parameter adjustments (cf. Table 3.1), the most important of which are as follows: the costs for G^0 and G^1 transitions were lowered, because the algorithm failed for several subsequences and error tolerances due to too high default smoothness demands. Also, we needed to increase the error cost, i. e., the amplification factor for fitting errors in the cost model, thus increasing fitting error penalties. In doing so, the accuracy of the final clothoid splines to the original trajectory could be improved. Also, a factor S was introduced with which the input coordinates and consequentially the curvature estimation region and the error threshold were scaled. This scaling was necessary, because we used input traces in a metric coordinate format and the original software could not cope with the high accuracy bounds in the range of centimeters, since it was designed to work not on meters and centimeters but on pixels.

With this configuration, the algorithm was able to compress the collected trajectory data. However, for some traces, the Cornucopia software failed reproducibly due to erroneous memory handling. These cases are therefore not included in the results presented in the following.



(a) Low velocity trajectories (S_1^{low}).



(b) High velocity trajectories (S_1^{high}).

Figure 3.6: Effective clothoid spline fitting errors.

Fitting Error

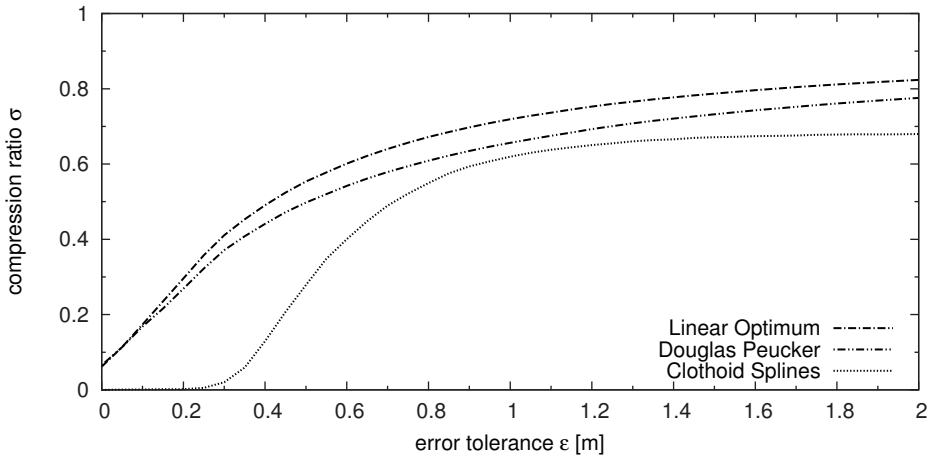
To validate the error bound compliance of the clothoid sketching approach and thus to make sure that it generates valid compression results, we first analyze the effective fitting error ϵ_e over an increasing error threshold ϵ. This check for the clothoid sketching approach needs to be performed, because it does not regard the error threshold directly throughout the complete approximation.

For the analysis, we group the effective errors of all trajectory approximations for each specific error tolerance value and then examine the resulting distributions. The results of this evaluation are shown in Figure 3.6 as box plots; the points below and above the whiskers show the minimum and maximum values, the whisker ends mark the 0.02 and 0.98 percentiles. The box itself covers the percentiles from 0.25 to 0.75 and the band within the box marks the median of the determined approximation errors per ϵ value. Dotted lines from the whisker ends to the points merely help to assign

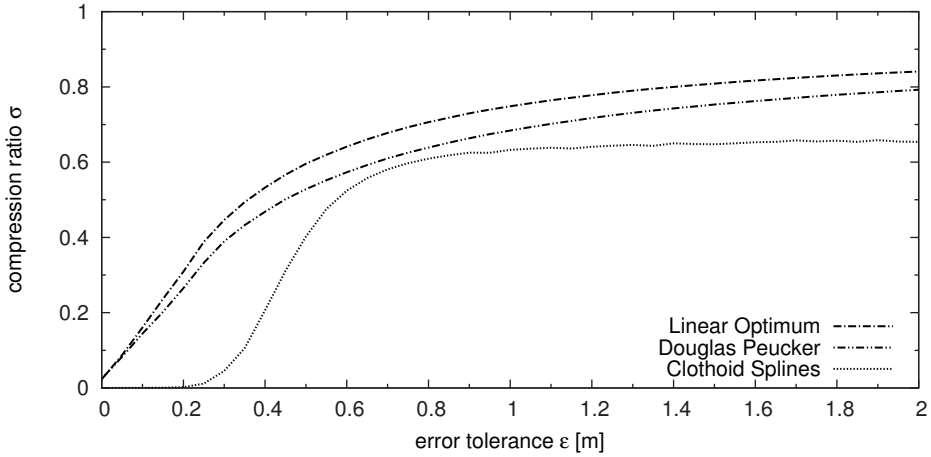
the points to the corresponding whiskers. Please note that all maximum values for the low velocity trajectories lie around 3.3 m and thus far outside of the displayed error range, while for the high velocity trajectories, most maximum errors for the ϵ steps are smaller than 2.0 m. The diagonal line is the static space diagonal with a slope equal to 1.0 and marks the validation criterion: for the algorithm to produce valid compression results for a given error tolerance, the upper point of the particular box plot, representing the highest occurred effective error, needs to lie underneath the diagonal. However, the plots reveal two interesting patterns: first, the effective fitting errors are hardly affected by the value of the error threshold for any topology at all. In fact, it appears that the algorithm quickly reaches an equilibrium for the trade-off between the number of curve primitives and the total fitting cost including error penalties and remains in or close to this equilibrium from there on. As a result, the approximation errors quickly stagnate for $\epsilon > 0.5$ m. Second, we see that for every error tolerance value, parts of the box plot diagrams lie above the space diagonal, i. e., there were always at least some trajectories for which approximation errors ϵ_e occurred that were larger than the tolerance ϵ. This means that even the clothoid spline fitting algorithm presented in [BLP10] heavily violates the error tolerance in some cases and therefore is not completely suitable for being employed for trajectory compression in its current state. To take the cases in which the error threshold is violated into account, we reduce the compression ratio for each of these cases, where $\epsilon_e > \epsilon$, to 0.0 %.

Compression Performance

For both the low and high velocity trajectory sets, the compression results of the clothoid-based method that are depicted in Figure 3.7 show a different pattern than the ones obtained by the linear simplification methods: due to the discussed high effective fitting errors and the resulting compression ratios of 0.0 for a huge number of trajectories, the compression curve remains below 0.05 until $\epsilon \approx 0.3$ m and then rises up to 0.6. This compression ratio is reached at $\epsilon \approx 0.9$ m for the low velocity trajectory sets and at $\epsilon \approx 0.75$ m for the high velocity trajectory sets. At this point, the curve flattens out and rises up to approximately 0.65. From this it is apparent that the compression performance is slightly better for the high velocity trajectories; this is the same effect that we can see in the results of the linear approaches and that results from more regular and smooth movements at higher velocities. Overall, the compression scheme based on the clothoid spline sketching therefore does not reach the benchmark performance of the linear simplification methods for any error tolerance value or topology.



(a) Low velocity trajectories (S_1^{low}).



(b) High velocity trajectories (S_1^{high}).

Figure 3.7: Clothoid splines: compression analysis.

As mentioned, the clothoid spline approach seems to run into an equilibrium state, as the effective fitting error results from Figure 3.6 indicate. This makes the comparison to other techniques not appear completely fair, as these regard the error tolerance ϵ directly. However, we see that though the equilibrium is reached at $\epsilon > 0.5\,\text{m}$, the compression ratios rise up to $\epsilon \approx 0.9\,\text{m}$, because the final calculation of the compression ratio directly includes the error tolerance, as well. It should be kept in mind that a competitive clothoid compression approach should rather work with respect to the error tolerance.

Since the poor compression results of the clothoid spline approach originate from the large approximation errors, keeping these within the preset error bound is the key

challenge on the way to create a clothoid spline compression scheme with a better performance. Please also keep in mind that for the employed clothoid spline approach, we can merely plot the results for ϵ constraining the *fitting error*, not the *approximation error*; for a necessary encoding of the measurement distribution over the arc length, a significant additional overhead is anticipated, the exact amount of which depends on the employed encoding scheme, though.

3.5 Cubic Spline Trajectory Point Reduction

Though the trajectory compression using clothoid splines does not work as well as expected, the idea of using smooth functions for the representation of vehicular movements still suggests itself: in vehicle dynamics [Gil92], the movement of vehicles is modeled on the basis of the principles of kinematics, covering detailed calculations of longitudinal and lateral acceleration forces, among others. Vehicle trajectories in these models are directly influenced by the moving object's speed, acceleration and steering angle. These components can be described as smooth functions, because their changing rates are continuous. This implies that basically, vehicle trajectories themselves can also be expressed as smooth functions $s \in C^2$, mapping a progress variable to a geographical position:

$$s : \mathbb{R} \longrightarrow \mathbb{R}^2, \qquad s(t) = (x, y)$$

Due to its weight and the mass inertia, for example, a vehicle is not able to perform an instant turnaround at a high speed but has to adjust its path and velocity continuously. With respect to the above mentioned position measurements, this continuity translates to redundancy: there is a *pattern* (based on kinematics) underlying these measurements. We propose to take advantage of this property to compress the information about the vehicle's trajectory.

Against this background, it is not surprising that in the context of autonomous robots and vehicles, smooth higher degree polynomial functions are used for trajectory planning. In [SK07, CE08], for example, the authors use Bézier curves, but emphasize that polynomials of higher degree tend to be unstable and exhibit higher oscillations. Therefore, Bézier curves with a degree of three are often employed, as they provide a satisfying smoothness and a low oscillation behavior. Another way of trajectory planning is by means of splines [LNRL06, DMF+06, Keh07]. In numerics, splines are employed to approximate points that lie in-between the elements of a sequence of given data points, also referred to as *knots*. To this end, it is assumed that the underlying and unknown function can be approximated with a higher-polynomial smooth con-

catenation of curves that pass through the knot sequence, so polynomials are defined between each two succeeding knots under certain conditions that ensure the spline's smoothness.

Though the publications cited above strengthen the argument that higher order polynomials are basically suited for the approximation of trajectories, they do not provide a solution to the problem of reducing the descriptive data volume for such structures. In contrast, the basic assumptions of the work in this field include an exact knowledge of the road geometry and partially even equidistant points, which is contrary to the nature of the position measurements that we can expect.

In this section, we discuss the interpolation of pure geometric data with two-dimensional cubic splines and present a greedy algorithm that finds a locally optimal solution for the error-aware reduction of a spline knot sequence. This algorithm has been published in [KKKM11]. Since sophisticated vehicular applications report more than just spatial data, as already discussed in Subsection 2.1.2, we also show that non-geometric data can easily be added and present a very efficient way to encode spatio-temporal information sets by spline interpolation. We then turn towards modifications that we proposed in [KBMS11] and that improve the compression performance of the basic algorithm and evaluate the results in comparison to the line simplification approaches from Subsection 3.3.1.

3.5.1 Cubic Spline Interpolation

Before we turn to our compression scheme that is built upon spline interpolation, we give a brief insight into the origin and mathematical background of splines. For a more detailed introduction to the theory and applications of splines, please refer to [JAW67].

The design of splines originates from elastic stripes often made of wood that were used by draftsmen to construct smooth and elegant curves that should pass through a given sequence of points. In numerics, this concept was adapted to interpolate points that lie on the curve of an unknown function, the exact values of which are only known at a finite number of places. In analogy to the technical drawing tools, the known place-value tuples are used as fixed points for the curve. Between each two of these fixed *knots*, polynomials are defined that fit together smoothly. The smoothness criteria translate into the demand for C^{n-1} continuity for *n-th degree polynomial splines* at the knots. A prominent example for splines are *cubic* splines, i. e., splines that employ third-degree polynomials and fulfill the C^2 smoothness criterion.

Formally, given positions $\Delta : x_1, \ldots, x_N$ and corresponding values $Y : y_1, \ldots, y_N$,

we are looking for a spline function $S_{\Delta,Y}(x)$ with

$$S_{\Delta,Y}(x_i) = y_i \ , \quad 1 \leq i \leq N \ .$$

The spline function is piecewise defined by a set of polynomial functions $s_j, 1 \leq j < N$ that are defined on the respective intervals $[x_j; x_{j+1}]$ and that satisfy

$$\begin{aligned} s_j(x_j) &= y_j \ , \\ s_j(x_{j+1}) &= y_{j+1} \ . \end{aligned} \tag{3.5}$$

Furthermore, the smoothness criteria need to be met for $1 \leq j < N-1$:

$$s_j^{(k)}(x_{j+1}) = s_{j+1}^{(k)}(x_{j+1}) \ , \quad 1 \leq k < n \tag{3.6}$$

for n-th degree polynomial splines. Let us in the following fix on cubic splines. As there is a function for each of the $N-1$ intervals and each function has four $(= n+1)$ degrees of freedom, this results in a linear system of $4N-6$ equations with $4N-4$ unknowns. There are two degrees of freedom, because for *nonperiodic splines*, i. e., splines that do not repeat themselves so that $y_i = y_{i+c \cdot N}, c \in \mathbb{N}^{>0}$ does not apply, the smoothness criteria for the border knots at x_1, x_N are undefined. There are several strategies to resolve this situation, for example to demand $s_0''(x_1) = s_{N-1}''(x_N) = 0$ as for *natural splines*. We employ *not-a-knot* splines, though, that attempt to determine the spline's boundary behavior based on the context of the inner parameters to achieve a more meaningful approximation; therefore, they require the third derivatives to match as well:

$$s_1'''(x_2) = s_2'''(x_2) \ , \quad s_{N-2}'''(x_{N-1}) = s_{N-1}'''(x_{N-1}) \ .$$

The solution to this linear system can be found in $\mathcal{O}(N)$.

In our use case, it is necessary to work with two-dimensional data, as our measurements contain longitudinal and lateral coordinates. Therefore, we use *two-dimensional splines*: using an index variable t, the dimensions are split, but still mapped using the same indexes. Then, both dimensions are handled by independent spline functions $S_{\Delta,Y_{\text{lon}}}$ and $S_{\Delta,Y_{\text{lat}}}$. For several reasons, two-dimensional cubic splines are well-suited to our demands: first, due to the C^2 continuity, their continuous curvature κ, given as

$$\kappa = \frac{S'_{\Delta,Y_{\text{lon}}}(t) \cdot S''_{\Delta,Y_{\text{lat}}}(t) - S''_{\Delta,Y_{\text{lon}}}(t) \cdot S'_{\Delta,Y_{\text{lat}}}(t)}{\left(S'_{\Delta,Y_{\text{lon}}}(t)^2 + S'_{\Delta,Y_{\text{lat}}}(t)^2\right)^{3/2}} \ ,$$

ensures that a realistic steering behavior can be modeled as argued above. Second,

cubic splines provide an especially low oscillation behavior due to their minimal curvature between two successive knots, resulting in a special smoothness of the final spline curve. Third, with the index variable, skipped points between two remaining knots can be very accurately interpolated with only minimal effort. Finally, with this systematic reconstruction, an error bound can easily be implemented; we therefore set up a compression scheme based on cubic spline interpolation in this section.

3.5.2 Basic Trajectory Point Reduction

To encode vehicle trajectories, we approximate the input dimensions of a measured trajectory $\langle m_j \rangle$ separately and seek a two-dimensional cubic spline that interpolates every taken position measurement. Given an accuracy bound ϵ, we compress the trajectory by removing redundant elements that can later be reconstructed by means of spline interpolation without violating the accuracy bound. In this context, a data point is redundant if it can be approximated by the cubic spline with a maximum error of ϵ. Then, the resulting subsequence of spline knots is the sought-after point sequence $\langle m'_j \rangle$.

We can now state the problem as follows: given a sequence of position measurements $\langle m_j \rangle$ as defined in (3.1) and $\epsilon \geq 0$. What is the minimal knot subsequence $\langle m'_j \rangle$ that can approximate $\langle m_j \rangle$ by means of cubic spline interpolation within the upper interpolation error bound ϵ?

To find a globally optimal solution to this problem, one would need to determine the interpolation error for every subsequence of $\langle m_j \rangle$ and then select $\langle m'_j \rangle$ as the smallest subsequence with an interpolation error not exceeding ϵ. Given the length of the original measurement sequence $n = |J|$, there are 2^n possible mutually distinct subsequences of $\langle m_j \rangle$. Since the calculation of the interpolation error has a linear complexity in n, the overall complexity of this approach of finding a globally optimal solution lies within $\mathcal{O}(n \cdot 2^n)$. In the domain of vehicular communication, however, several hundred positions can easily be included within a single measurement sequence. This makes the described naive approach unfeasible. Instead, we propose to approximate the optimal solution for the stated problem. Following this premise, we now propose a greedy algorithm running in $\mathcal{O}(n^3)$ that finds such an approximate solution, referred to as $\langle m'_j \rangle$.

Our algorithm uses a greedy iterative search to reduce $\langle m_j \rangle$ down to $\langle m'_j \rangle$ as presented in Algorithm listing 3.1: given the measurement sequence $\langle m_j \rangle$ and an error bound $\epsilon \geq 0$, the algorithm checks in each iteration for each but the first and last remaining element in $\langle m_j \rangle$ what the highest resulting interpolation error will be, if

Algorithm 3.1 Basic greedy spline reduction.

Require: sequence of position measurements $\langle m_j \rangle_{j \in J}$
Require: error tolerance $\epsilon \geq 0$
1: $I_{\text{rem}} \leftarrow \emptyset$
2: **repeat**
3: $\quad \epsilon_{\min} \leftarrow \infty$
4: $\quad j_{\min} \leftarrow -1$
5: $\quad$ **for all** $j \in J \setminus I_{\text{rem}}$ **do**
6: $\quad\quad$ **if** $j \in [1; |J| - 2]$ **then**
7: $\quad\quad\quad I_{\text{rem}} \leftarrow I_{\text{rem}} \cup \{j\}$
8: $\quad\quad\quad \langle \widehat{m}_j \rangle \leftarrow \text{interp}(\langle m_j \rangle, I_{\text{rem}})$
9: $\quad\quad\quad \epsilon_j \leftarrow \text{compare}(\langle m_j \rangle, \langle \widehat{m}_j \rangle)$
10: $\quad\quad\quad$ **if** $\epsilon_j < \epsilon_{\min}$ **then**
11: $\quad\quad\quad\quad j_{\min} \leftarrow j$
12: $\quad\quad\quad\quad \epsilon_{\min} \leftarrow \epsilon_j$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad\quad I_{\text{rem}} \leftarrow I_{\text{rem}} \setminus \{j\}$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: $\quad$ **if** $\epsilon_{j_{\min}} \leq \epsilon$ **then**
18: $\quad\quad I_{\text{rem}} \leftarrow I_{\text{rem}} \cup \{j_{\min}\}$
19: $\quad$ **end if**
20: **until** no further knot could be removed (I_{rem} remains unchanged)
21: $\langle m'_j \rangle \leftarrow \text{remove\_knots}(\langle m_j \rangle, I_{\text{rem}})$
22: **return** reduced knot sequence $\langle m'_j \rangle$

this element is removed from the spline knot sequence (cf. line 8f). For this purpose, the method interp$(\cdot, \cdot)$ calculates the spline of a knot sequence, interpolating the knots at the indexes from a given set. Then, the method compare$(\cdot, \cdot)$ compares two knot sequences and returns the largest pointwise Euclidean distance. The index of the sequence element with the smallest of these interpolation errors is then stored in I_{rem} (cf. line 18). The algorithm stops once no further element can be removed without violating the error bound ϵ. Then, the elements at the indexes stored in I_{rem} are removed from $\langle m_j \rangle$ and the resulting reduced subsequence $\langle m'_j \rangle$ is returned. This algorithm can be denoted as a simplification algorithm that keeps original data points and removes redundant ones, similar to the line simplification algorithms discussed in Subsection 3.3.1. Therefore, we also need to maintain a bit field memory to restore the removed knots from $\langle m_j \rangle$ and the compression ratio achieved by this algorithm can analogously be defined as in Statement (3.2).

3.5.3 Adding Non-Geometric Data

In the discussion of the Douglas-Peucker algorithm, we already introduced how temporal information can be attached to the pure spatial description contained in $\langle m_j \rangle$ or $\langle m'_j \rangle$. More sophisticated applications extend the spatio-temporal vehicular trajectories by adding more sensor data, e. g., measurements of temperature, humidity or friction coefficients. Such data is, for example, included in the *Floating Car Data (FCD)* that has already been discussed in Section 2.1.

In case that non-geometric information is added to the trajectory description, the dimensionality d of the data collection increases and we can generalize our previous definition of a measurement sequence from Statement (3.1) accordingly to

$$m : \ J \rightarrow \mathbb{R}^d \ .$$

The above-stated upper bounds for such measurement collections can be generalized as well: assuming a measurement tuple size of $d \cdot s_n$, and since the overhead s_o is independent of d, Statement (3.4) for the upper bound, respectively, needs to be adapted, so that

$$|J| - |J'| > \frac{s_o}{d \cdot s_n}$$

needs to hold for a successful reduction. This implies that the necessary effective reduction ratio of the measurement set decreases for an increasing dimensionality d of the measurement tuples.

Please note that though cubic splines are considered exceedingly useful for trajectory interpolation in particular, the reduction technique should always fit the particular context. For example, friction parameters might be interpolated by a much simpler construct than cubic splines. Furthermore, the dimensionality d of the input data is irrelevant for the algorithm's asymptotic runtime complexity: as long as the error determination per reduction step does not exceed $\mathcal{O}(n)$, the algorithm's complexity remains in $\mathcal{O}(n^3)$, since a fixed dimensionality does not affect its asymptotic behavior. The only supplement that has to be added to the algorithm is a further error threshold for each new dimension or combination of new dimensions.

As mentioned earlier, the focus will from now be on temporal information as non-geometric data due to its special interrelation with spatial information.

3.5.4 Unseaming Dimension Contexts

The basic version that has just been presented has a drawback that may inhibit a better compression performance: in the first step, we compute the interpolation errors

Algorithm 3.2 Greedy spline reduction with an unseamed dimension context.

Require: sequence of position measurements $\langle m_j \rangle_{j \in J}$
Require: error tolerance $\epsilon \geq 0$
1: $I^0_{\text{rem}}, I^1_{\text{rem}} \leftarrow \emptyset$
2: $\langle m^0_j \rangle, \langle m^1_j \rangle \leftarrow$ separate_dimensions($\langle m_j \rangle$)
3: **repeat**
4: **for all** $d \in \{0, 1\}$ **do**
5: $\epsilon_{\text{min}} \leftarrow \infty$
6: $j_{\text{min}} \leftarrow -1$
7: **for all** $j \in J \setminus I^d_{\text{rem}}$ **do**
8: **if** $j \in [1; |J| - 2]$ **then**
9: $I^d_{\text{rem}} \leftarrow I^d_{\text{rem}} \cup \{j\}$
10: $\langle \widehat{m}_j \rangle \leftarrow$ merge_dimensions(interp($\langle m^0_j \rangle, I^0_{\text{rem}}$)), interp($\langle m^1_j \rangle, I^1_{\text{rem}}$))
11: $\epsilon_j \leftarrow$ compare($\langle m_j \rangle, \langle \widehat{m}_j \rangle$)
12: **if** $\epsilon_j < \epsilon_{\text{min}}$ **then**
13: $j_{\text{min}} \leftarrow j$
14: $\epsilon_{\text{min}} \leftarrow \epsilon_j$
15: **end if**
16: $I^d_{\text{rem}} \leftarrow I^d_{\text{rem}} \setminus \{j\}$
17: **end if**
18: **end for**
19: **if** $\epsilon_{j_{\text{min}}} \leq \epsilon$ **then**
20: $I^d_{\text{rem}} \leftarrow I^d_{\text{rem}} \cup \{j_{\text{min}}\}$
21: **end if**
22: **end for**
23: **until** no further knot could be removed (I^0_{rem} and I^1_{rem} remain unchanged)
24: $\langle m'_j \rangle \leftarrow$ merge_dimensions(remove_knots($\langle m^0_j \rangle, I^0_{\text{rem}}$), remove_knots($\langle m^1_j \rangle, I^1_{\text{rem}}$))
25: **return** reduced knot sequence $\langle m'_j \rangle$

for possibly-removed knots (cf. line 8f in Algorithm listing 3.1). For this, the absolute Euclidean distance between an original knot and its interpolated counterpart is calculated, i. e., both dimensions of the spline are compressed simultaneously as a *context*. In doing so, the basic algorithm does not take into account that one of the dimensions may be more complex and therefore harder to compress than the other; in fact, the compression result of the two-dimensional context can only be as good as the one of the more complex dimension.

To overcome this issue, we propose a modified version of our algorithm that compresses each dimension separately. This version is shown in Algorithm listing 3.2, the modified and new lines are highlighted: the two steps of the outer loop, namely the determination of the highest resulting interpolation errors for each knot (cf. lines 7-18) and the removal of the knot that minimizes these errors (cf. lines 19-21), are now

applied to the dimensions sequentially. In doing so, instead of removing a knot at the same index in both dimensions at the same time, one index for each dimension is determined independently. The dimensions are thus compressed separately, still regarding the threshold ϵ for the total interpolation error (cf. line 19). In the code listing, the index for the longitudinal (or x, index 0) dimension is determined first, and the lateral (or y, index 1) dimension second. The changes performed on the first dimension influence the compression of the second one and therefore a changed order of dimensions is likely to affect the overall compression, albeit marginally. Note that it is now possible that, at some point, one dimension can not be compressed any more, whereas the other still can. The algorithm may then skip one dimension in the remaining cycles until the compression of the second dimension has also reached a local optimum. The runtime complexity of our algorithm is not affected by this modification, since we merely multiply the quadratic effort of determining the interpolation error by the (fixed) number of dimensions: $\mathcal{O}(d \cdot n^3) = \mathcal{O}(n^3)$. Even though this *context-loosened* compression is likely to provide better results, it also causes a higher overhead in one respect: since the dimensions are compressed in a largely decoupled fashion, it is now necessary to provide one bit field *per dimension* to remember the removed knot indexes. We verify this by describing the absolute overhead threshold for the context-loosened compression mapping by analogy with (3.3) and (3.4) as follows:

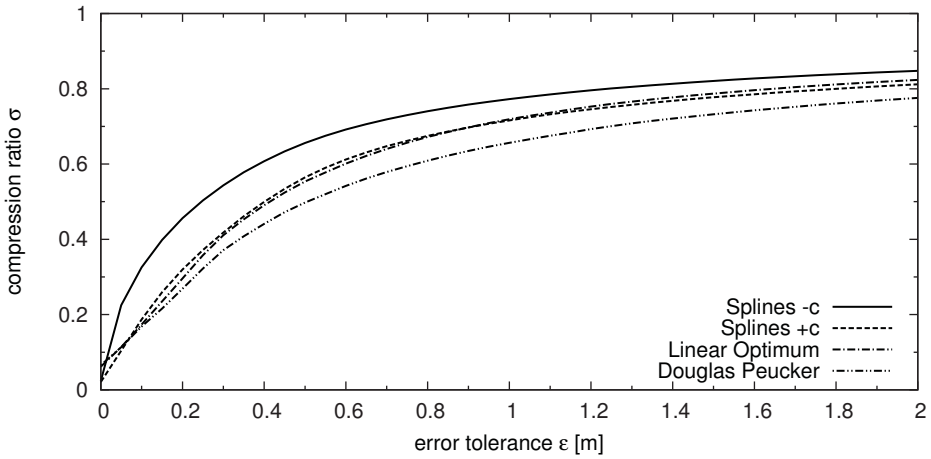
$$\begin{aligned} & 2 \cdot s_n \cdot |J| > 2 \cdot s_n \cdot k + 2 \cdot s_o \\ \Leftrightarrow \quad & |J| - k > \frac{s_o}{s_n} \end{aligned} \tag{3.7}$$

where k is now the mean dimension knot sequence length for $\langle m'_j \rangle$. As expected, the comparison to the basic algorithm according to (3.4) shows that with the context-loosened compression, twice as many knots need to be removed from $\langle m_j \rangle$ to achieve a compression.

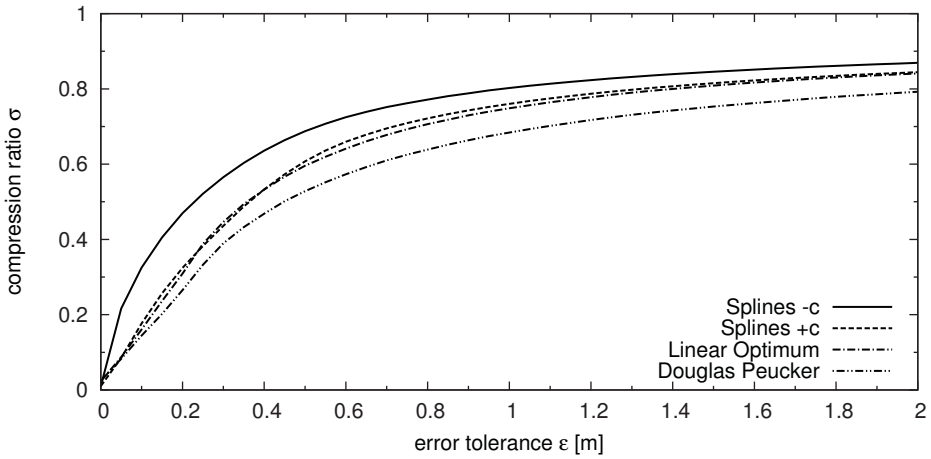
3.5.5 Evaluation

Compression Performance

We first take a closer look at the compression performance of the discussed approaches. The compression results for the cubic spline compression schemes are shown in Figure 3.8. Basically, the compression ratio curves are similar to those for the line simplification approaches: for small error tolerance values, there is a steep increase of the compression ratio, but the curves flatten out from $\epsilon \approx$ 50-100 cm on. Again, this is true for both the low and the high velocity trajectory sets.



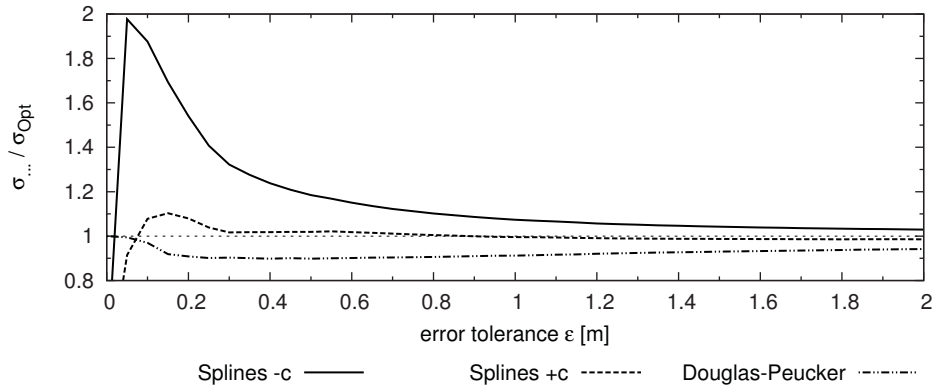
(a) Low velocity trajectories (S_1^{low}).



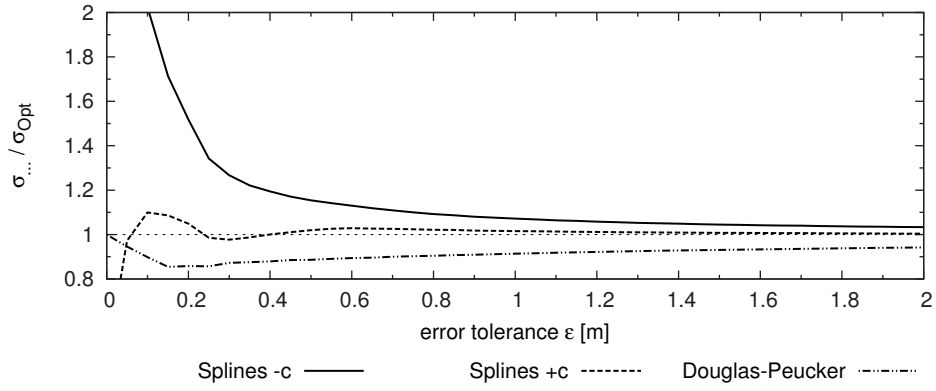
(b) High velocity trajectories (S_1^{high}).

Figure 3.8: Cubic splines: compression analysis for varying error thresholds.

The basic context-oriented cubic spline compression approach (Splines +c) does not perform significantly better than the line simplification approaches, it sometimes even performs slightly worse. This result differs clearly from previous studies published in [KBMS11], but is similar to our more recent results from [KM11]. The reason is that we used vehicular trajectories with a much better quality for the first-mentioned publication, i. e., the trajectories were recorded with a high-quality GPS receiver with a very low positioning noise. More currently, our data base is considerably larger, but, as already mentioned in Section 2.2, originates from the OpenStreetMap project where a very heterogeneous set of community member GPS receivers is used. Of course, the better the quality of the trajectory, the better the compression ratio. Since the



(a) Low velocity trajectories (S_1^{low}).

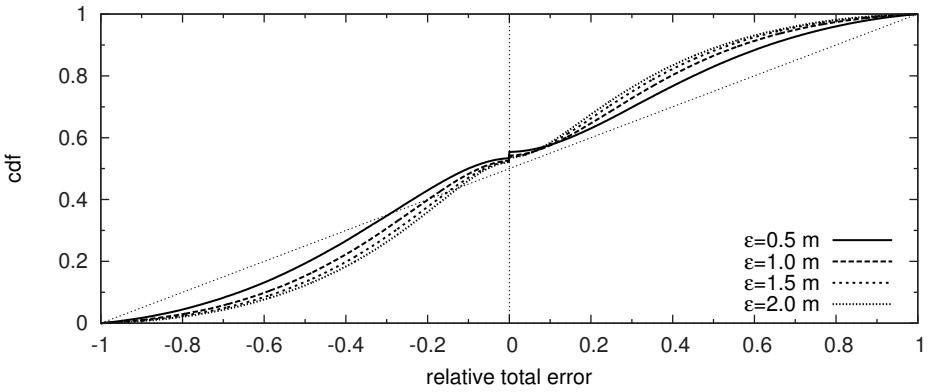


(b) High velocity trajectories (S_1^{high}).

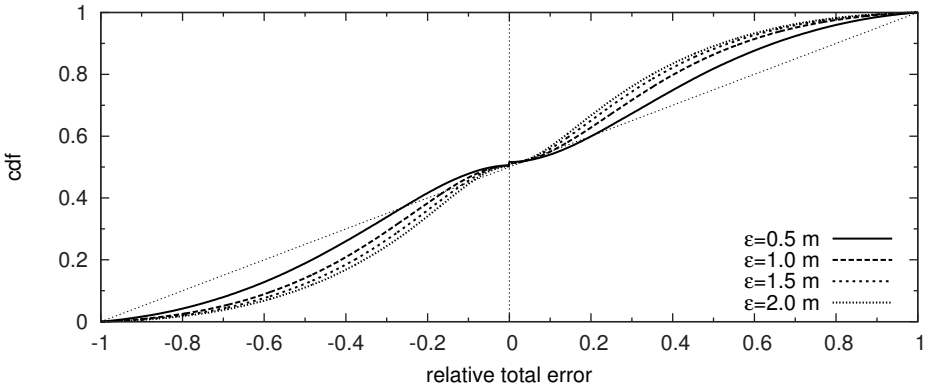
Figure 3.9: Cubic splines: relative compression analysis for varying error thresholds.

performance of the cubic spline approaches depends heavily on the smoothness of the trajectory, these approaches suffer more from coarser position measurements.

The context-loosened variant of the cubic spline approach (Splines -c), however, shows a clear improvement of the compression ratio in comparison to the basic version. The improvement is better visible in Figure 3.9 that shows a relative compression performance analysis over a varying error tolerance: once the compression ratios of the approaches are stable ($\epsilon \approx 0.1$), the basic spline approach performs approximately as well as the the optimal line simplification approach. The context-loosened spline approach outperforms all other approaches and provides a more than 10 % better compression ratio than the optimal line simplification for $\epsilon < 0.8\,\text{m}$. This clearly underlines our fundamental hypothesis that, if geometric approaches are employed, vehicular trajectories should be approximated with smooth instead of linear functions.



(a) Low velocity trajectories (S_1^{low}).



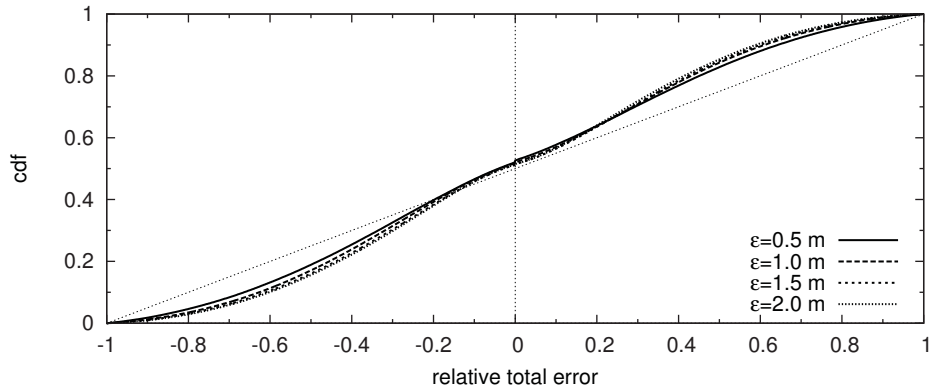
(b) High velocity trajectories (S_1^{high}).

Figure 3.10: Total error analysis (Splines +c).

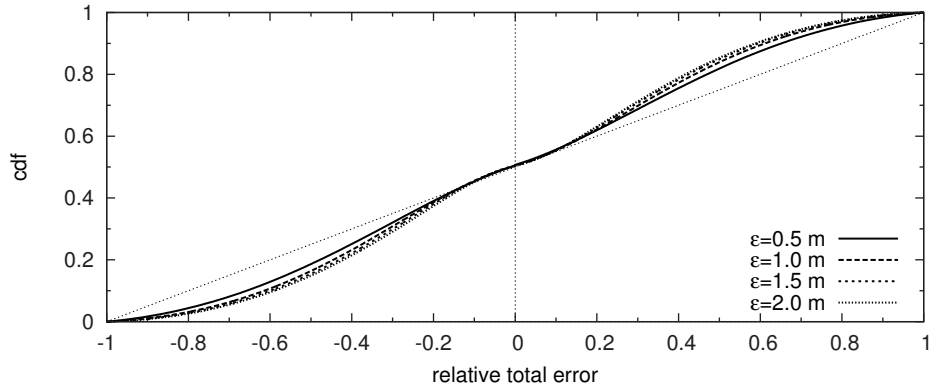
Interpolation Error Analysis

We have seen that even a relatively small interpolation error threshold allows for high compression ratios for the collected trajectories. In the next step, we take a closer look at the nature and distribution of this interpolation error.

Figures 3.10 and 3.11 show the *cumulative distribution functions (cdf)* of the total relative interpolation error for the low and high velocity trajectories and for a selection of interpolation error bounds. The relative interpolation error describes the ratio of the occurred interpolation error to the interpolation error bound. A negative interpolation error occurs, if the original point lies on the left hand side of the interpolation, in the direction of movement. The results show a symmetric interpolation error distribution, i. e., positive and negative interpolation errors are distributed alike. Only for the low velocity trajectories, there is a slight overbalance of the negative interpolation errors; this is visible for both the basic and the context-loosened approach and therefore seems



(a) Low velocity trajectories (S_1^{low}).



(b) High velocity trajectories (S_1^{high}).

Figure 3.11: Total error analysis (Splines -c).

to be topology-specific. Additionally, we see that with an increasing error tolerance, the distributions become narrower to the center (i. e., to a relative total error = 0). This is because the interpolation errors are not spread evenly, but only few measurement points are interpolated at a close-to-maximum interpolation error. This effect is more distinctive for the basic approach; the context-loosened approach yields rather evenly distributed interpolation errors, which finds expression in a closer distance to the dotted diagonal with the function expression $f(x) = 0.5x + 0.5$.

Though the total interpolation error distribution is a good indicator for the quality of the trajectory interpolation, it is not enough to make clear statements. A more thorough understanding can be provided by differentiating the components of the interpolation error, instead. Therefore, we will regard its longitudinal and lateral parts for our considerations as well, Figure 3.12 gives a schematic overview: it shows a polygonal chain as a dotted gray polyline and the corresponding spline approximation as a

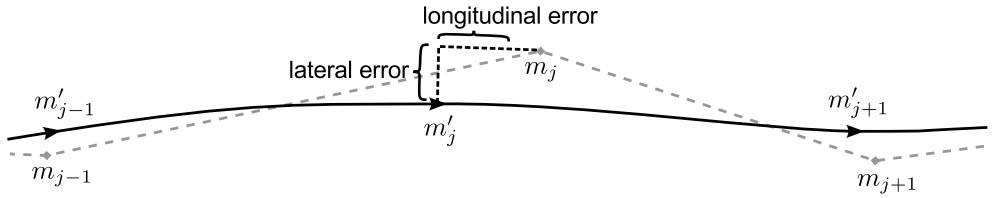
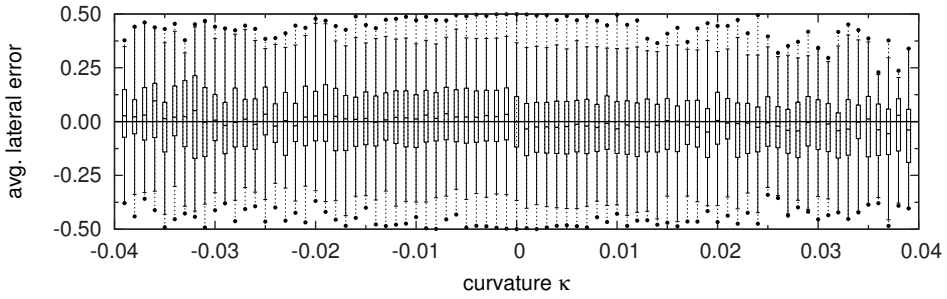


Figure 3.12: Longitudinal and lateral error components.

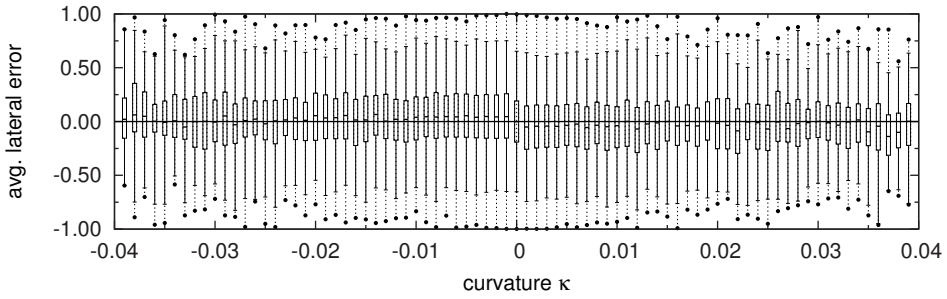
solid black curve. Additionally, the mentioned error components for the interpolated knot m'_j of the original position measurement m_j are depicted. While the longitudinal interpolation error describes the divergence along the movement direction, the lateral error refers to the part perpendicular to it.

As mentioned in the beginning of this section, cubic splines provide an especially low oscillation behavior due to a minimal curvature κ between two successive knots. Therefore, it is surely possible that systemic errors in relation to the curvature of the spline interpolation occur, so that curves and bends are interpolated tighter than they actually were in the original data. In this case, a clear correlation between the curvature of the spline and the lateral error would exist: for $\kappa < 0$ (i. e., right turns), one would expect mostly negative lateral errors (i. e., original measurements lying on the left side of the interpolation) and for $k > 0$, positive lateral errors should prevail. This is why especially the lateral error is a good metric for the accuracy for the spline interpolation of trajectories.

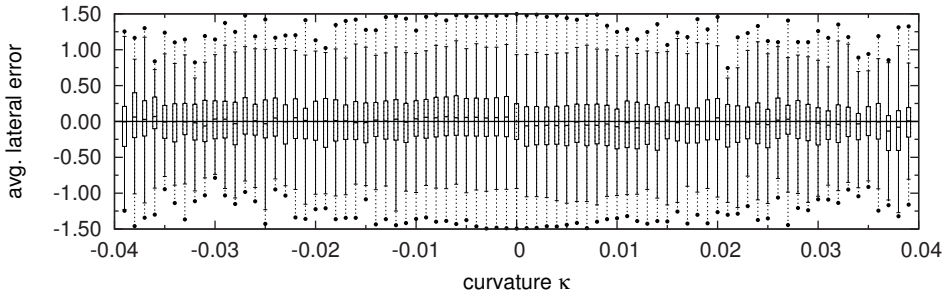
Figures 3.13 and 3.14 show the lateral error distributions as box plots in relation to the spline's curvature; again, the points indicate the minimum and maximum values, the whisker ends mark the 0.02 and 0.98 percentiles, the box itself covers the percentiles from 0.25 to 0.75 and the band within the box marks the median. Since the curvature $\kappa = \frac{1}{R}$ is the inverse of the curve radius R, a high absolute curvature value translates into a tight curve or bend. The figure covers curvatures of up to $|\kappa| = 0.04$, resulting in curve radii of up to 25 m that correspond to tight curves with respect to the measured velocities. The figures show a balanced lateral error distribution, but with a slight systemic error by shifts of mostly less than 5 %. However, this error has the exact opposite nature of what is expected: for $\kappa < 0$, the median of most box plots is positive, while for $k > 0$, most error medians are negative. This means that the splines tend to approximate curves in the trajectories wider than they actually are. Though this behavior occurs for both versions of the algorithm, the results for the context-loosened variant appear slightly more regular, while the errors of the context-oriented version seem to be a little more distorted.



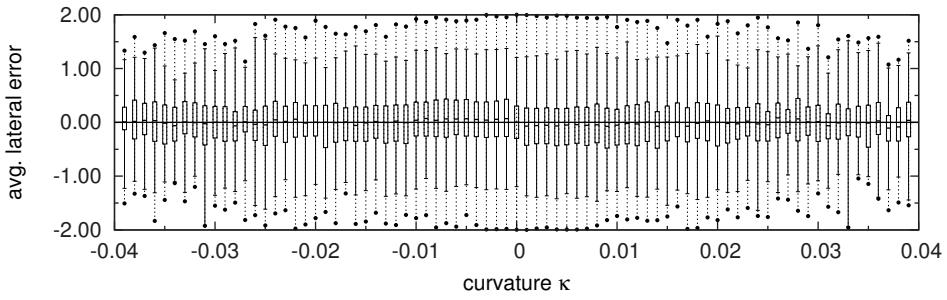
(a) $\epsilon = 0.50\,\mathrm{m}$



(b) $\epsilon = 1.00\,\mathrm{m}$

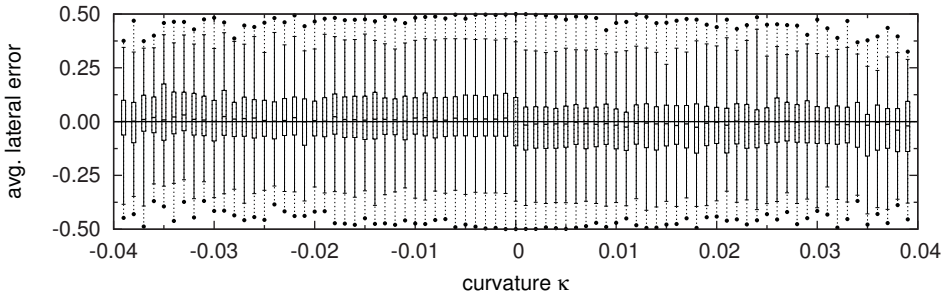


(c) $\epsilon = 1.50\,\mathrm{m}$

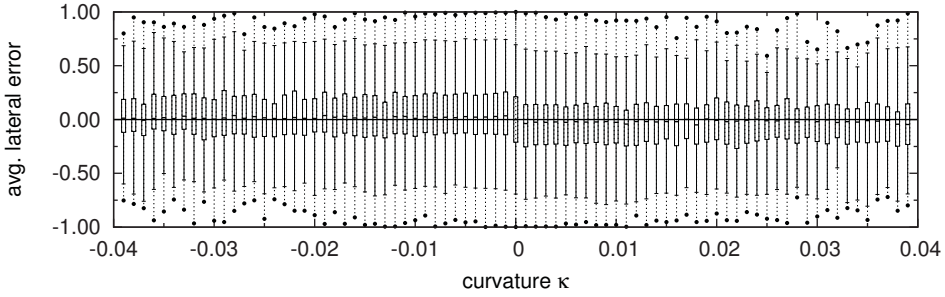


(d) $\epsilon = 2.00\,\mathrm{m}$

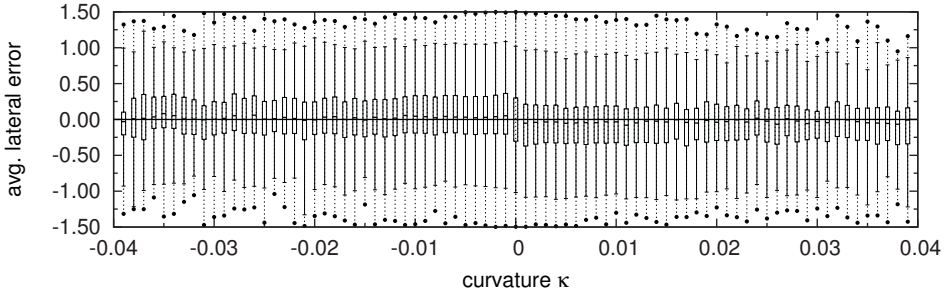
Figure 3.13: Lateral error analysis for high velocity trajectories (Splines +c).



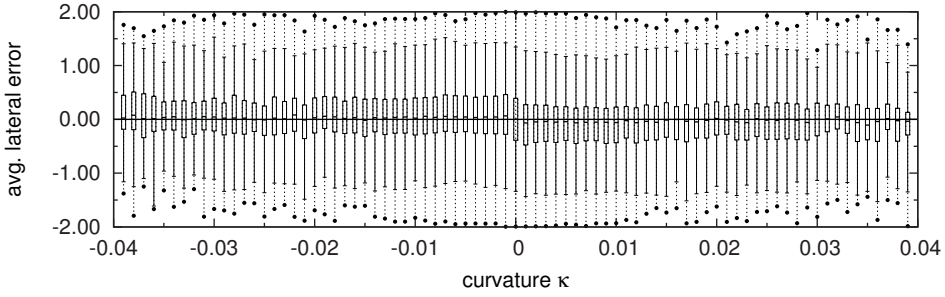
(a) $\epsilon = 0.50\,\mathrm{m}$



(b) $\epsilon = 1.00\,\mathrm{m}$



(c) $\epsilon = 1.50\,\mathrm{m}$



(d) $\epsilon = 2.00\,\mathrm{m}$

Figure 3.14: Lateral error analysis for high velocity trajectories (Splines -c).

3.6 Conclusion

In this chapter, we investigated lossy geometric compression schemes for the encoding of spatio-temporal vehicular trajectories that are able to compress trajectories regarding an upper approximation error bound ϵ. We analyzed the compression performances of optimal and heuristic line simplification algorithms that are currently the state of the art for the remote tracking of object movement.

We then regarded geometrical design patterns of roadways, clothoidal splines, that are compositions of clothoidal curve segments, circular arcs and line segments. We examined the suitability of clothoidal splines and proposed a new compression scheme on the basis of a clothoid spline fitting algorithm from the area of computer graphics. However, the problem statements that serve as foundations for these algorithms differ essentially from ours, as they do not involve a strict error bound, but rest upon rather fuzzy error cost models. Therefore, the resulting clothoid splines often violated the error bound so that the compression results that were achieved with these approaches were significantly worse than the ones of the line simplification algorithms.

Finally, we proposed a trajectory compression scheme based on cubic spline interpolation. We presented a basic version of a greedy algorithm that approximates a given trajectory with a runtime complexity of $\mathcal{O}(n^3)$. We also stated the possibilities to handle trajectories with additional non-geometrical data dimensions without the need of extensive modifications of the algorithm. We also presented a modification to the algorithm that compresses the spatial dimensions of a trajectory not bundled as a context, but separately. Our evaluation showed that the basic spline approach could not perform significantly better than the linear compression schemes and even performed slightly worse than the optimal line simplification algorithm. However, the proposed context-loosened modification outperformed all regarded compression algorithms and achieved constantly around 18 % to 22 % better compression ratios than the optimal linear simplification algorithm. This result clearly shows the suitability of smooth geometric functions for the compression of spatio-temporal trajectories.

Although we could achieve very good compression ratios, it is not clear what the best possible compression technique is or what the upper compression bound for spatio-temporal trajectories is. It is possible, for instance, that a compression scheme based on clothoid splines is the best-suitable approach and there is simply no algorithm yet that is capable of approximating polygonal chains efficiently within a strict error bound. Viewed from a different perspective, there are still open questions: how much *information* does a spatio-temporal trajectory contain and how can this *information content* be measured? If we could find an approach to answer these questions, it

would be easier to approximate an optimal compression scheme for spatio-temporal trajectories. We will therefore leave the geometric context to tackle these questions in the next chapter from an information-theoretic point of view.

4

Information Theoretic Approaches

4.1 Introduction

In the previous chapter, we described compression techniques for moving object trajectories that are based on geometric approaches, such as line simplifications, circular arcs or cubic spline interpolation. We could show that the use of these methods provide very good compression results and that the use of smooth functions can improve the compression performance, especially for high-accuracy constraints. However, it was not possible to derive a general upper bound for trajectory compression that is given by the *information content*, of such a movement trace.

In this chapter, we therefore leave the geometric context and focus on trajectory compression from an information-theoretic point of view. First, we review previous related work on trajectory compression and probabilistic positioning. Also, a brief introduction into Shannon's information theory is given and it is explained how these concepts can be used to find optimal *stream codes* that can be produced by an arithmetic coder. We then focus on the following question: *given a prediction model for object movements, how much information does a trajectory contain with respect to this model and what upper compression bound does this imply?* We introduce our idea of information content for moving object trajectories and present a method to measure this information content. This knowledge is then used to discuss how to apply this idea to vehicular trajectories and describe details of a basic arithmetic coder implementing the proposed model. The practical implementation consists of multiple components that we will regard more closely and discuss several alternative realizations of them. Finally, we present a detailed evaluation of the arithmetic coding model and its components.

Again, we will use the vehicular domain as an example to illustrate our findings and to prove its applicability to real world data. However, the ideas presented here can be used to analyze and compress any form of trajectory, provided that a prediction model for the respective mobility can be constructed.

Parts of the content of this chapter were previously published in [KRHM12], but we will present more details about the code model and its components in here. Also, we discuss our findings on the symbol alphabet and the probability distribution in more detail, as well as their impact on the arithmetic coding performance. In particular, we put emphasize on the trained probability distributions and how these could be retrieved.

4.2 Related Work

The probability of the outcome of a position determination is an important factor in *probabilistic positioning*. This self-positioning technique is widely used for the navigation of autonomous mobile robots, mostly within buildings: a robot, equipped with distance sensors and map material, explores its surrounding and finds its position on the map. An early publication of probabilistic robot positioning is [NPB95]; the authors describe the algorithms and the setup of their robot *DERVISH* that knows its initial position and navigates to a particular destination, thereby avoiding obstacles on its way. The robot uses rudimentary probabilistic positioning by matching its sensor measurements to its approximate position, thus deducting its position on the map. The authors of [SK95] improve on the positioning by introducing *uncertainty* into the system: they propose to use a *partially observable Markov decision process (POMDP)* to express the robot's position by a probability distribution over a given map. In [CKK96], the authors propose the use of *Markov Localization* instead POMDPs due to its lesser computational complexity. Markov Localization is also employed in [Fox98, RBFT99], in which the presented robot navigation systems determine the routes with respect to a minimal-entropy policy, i. e., the uncertainty of the position along the route is minimized. In our work, we follow a similar concept in defining a probability distribution over a limited region, but we do not rely on previously known map material, but only on position estimations and spatial boundaries set based on kinematic assumption.

Two publications from the area of machine learning and mobile communication are close to our contribution: in [RGRB04], the authors analyze navigation decisions of homing pigeons on their way between pigeon lofts. From the navigation decisions, they derive the routing uncertainties of the pigeons using a window approach, where a window is moved along a pigeon's trajectory. The navigational uncertainty is then

calculated based on the stochastic complexity of the regarded trajectory sections with a *Hidden Markov Model (HMM)*. Though the concept of positional entropy is close to our work, the study itself is merely loosely connected to ours: though the presented model allows to estimate the movement uncertainty, the probabilities are determined a posteriori by an abstract window scheme, which is hardly applicable to general moving object trajectories. The authors of [BD99] apply Shannon's concept of information content and entropy to the movement of users in a cellular network. The gained knowledge is then used to implement an intelligent location update protocol, *LeZi-Update*. In their work, they use Markov models to calculate the information content of the user movement and determine the state transition probabilities based on relative location frequencies. With this model, the authors derive compressed position update messages. This approach is a special case of information content measurements and gives good hints on how to approach the area of spatio-temporal entropy. However, it also can not directly be generalized to arbitrary movements. In this chapter, we present a formal information content measurement model that can not only be seen as a generalization of these two approaches but can also be adapted to any other application area or object movement.

None of the existing approaches consider a general upper bound for trajectory compression that is given by the information content of a movement trace. In this chapter, we will show that doing so will lead to significant improvements in the compression ratio for trajectories.

4.3 Information Theory and Entropy Coding

In this section, we introduce the information-theoretic concepts that we will need to develop our model for the determination of the spatio-temporal information content of trajectories. We then explain how this knowledge can be used to derive an optimal symbol and streaming code with *Huffman* and *arithmetic coding*, respectively. Finally, we introduce *range coding*, an efficient way to implement arithmetic coding.

4.3.1 Information Content and Entropy

Let us begin with the necessary information-theoretic concepts that can be studied in more depth in [Sha48]; for our terminology, we follow [Mac02]: given a random variable X with a finite sample space $\mathcal{A}_X$, that we will also refer to as the (discrete) *alphabet* of X,

$$\mathcal{A}_X = \{a_1, \ldots, a_N\}$$

and a *probability distribution*

$$\mathcal{P}_X = \{p_1, \ldots, p_N\}$$

so that

$$p_i = P(x = a_i), \quad \forall 1 \leq i \leq N : p_i > 0 \quad \text{and} \quad \sum_{i=1}^{N} p_i = 1 \ .$$

The concept of Shannon then states that the higher the outcome probability of an event, i. e., an alphabet symbol, the lower is its information content. In addition to that, the information content is not proportional to the inverse probability, but it decreases logarithmically, the more probable the outcome. Formally, the *Shannon information content* $h(x)$ (given in *bits*) of an event $x \in \mathcal{A}_X$ is defined as

$$h(x) = \log_2 \frac{1}{P(x)} \ ,$$

where $P(x)$ is the probability of its occurrence. Then, the *entropy* of a random variable X refers to the *average information content* of an outcome of X and is weighted according to the respective probabilities. The entropy $H(X)$ is defined as

$$H(X) \quad = \quad \sum_{x \in \mathcal{A}_x} P(x) \cdot h(x) \quad = \quad \sum_{x \in \mathcal{A}_x} P(x) \cdot \log_2 \frac{1}{P(x)} \ .$$

In other words, the entropy is the average number of necessary bits to represent a random outcome $x \in \mathcal{A}_X$.

4.3.2 Huffman Coding

In his fundamental paper on what he established as information theory [Sha48], Shannon describes the information contents that lie within symbol sets with attached probabilities and how to measure them. Based on this work, Huffman presents a way to calculate an *optimal symbol code* in the form of a set of binary code words corresponding to such a given alphabet and a probability distribution [Huf52]. In this context, an optimal code yields minimum redundancy as defined by Shannon, and a minimum expected code word length. To achieve this, Huffman proposes an algorithm to construct a tree structure that holds all alphabet symbols as leafs and defines the code words corresponding to the symbols as the paths from the root to the respective leaf. Such trees define *prefix codes*, i. e., codes in which no code word is the prefix to another. With this property, code words are always uniquely determinable from the code message. This structure can be constructed for codes with arbitrary positional notation radixes, but again, we focus on the binary case.

For the code tree construction, one tree node is initially set up for each alphabet symbol. In general, the algorithm follows the philosophy that the longest code word lengths are assigned to the symbols with the lowest probabilities. To this end, the symbol nodes are then inserted into a priority queue that regards the symbol probabilities for the ordering: the nodes are ordered by increasing symbol probabilities. If two nodes contain symbols of equal probabilities, the order of these two nodes is irrelevant, as the expected code word length (the entropy) will be the same. Then, the following steps are repeated until there is only one node left in the queue:

1. Take the first two nodes from the queue and create a parent node for them. The probability of the parent node is the sum of the two nodes' probabilities.
2. Insert the parent node into the priority queue.

Once there is only one node left in the priority queue, the algorithm terminates; this node is the root node of the code tree. The code words for the alphabet symbols can then directly be read as the binary path descriptors of the respective leafs, as seen from the root node.

The Huffman code provides *optimal symbol codes*, i. e., each symbol is encoded with minimal integer bit number. However, there are two main disadvantages, as stated in [Mac02]: first, the original Huffman coding works with a static code tree that can not react on changing symbol probabilities and needs to be known to all decoding instances, if it is not to be transmitted ahead of every code word sequence. Although adaptive strategies can be implemented by rebuilding or altering the code tree periodically, as proposed in [Gal78, Vit87], these solutions cause an extra management overhead by additional pointers in the tree. Additionally, the code tree alterations are suggested to be made after every R-th read symbol (with a larger R) to avoid noticeable coding slowdowns, thereby accepting suboptimal codes in the meantime. Second, since the Huffman code is a symbol code and each code word consists of an integer number of bits, there is always an encoding overhead of up to one bit per symbol. This can be alleviated by defining a Huffman code not only for the symbol unigrams, but for all symbol combinations of a certain length. For symbol strings of a known length M, this overhead could therefore be minimized by defining a Huffman code for all possible symbol combinations of size M. As M is usually unknown in practice, it is beneficial to use stream codes, such as *arithmetic codes* instead of symbol codes.

4.3.3 Arithmetic Coding

In [Abr60], the author proposes a *stream code* algorithm that separates the probabilistic model from the actual encoding step. To this end, the probability distribution of the

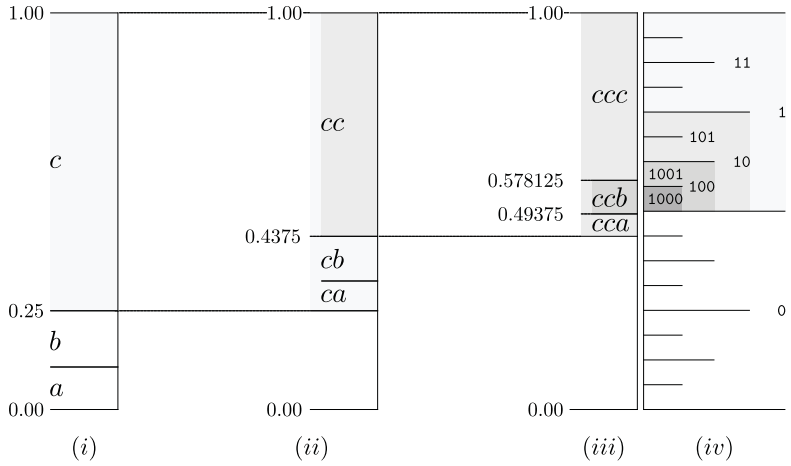


Figure 4.1: Arithmetic coding: exemplary coding of symbol string *ccb*.

input data needs to be estimated first and can then be updated upon each read input symbol. The approach of the Huffman coding, where the correct distribution can be determined by analyzing the input is not applicable here, as the arithmetic coding was designed to work on symbol streams. However, due to the separated probability and coding steps, it is very easy to adapt the probability distribution on the fly. In the worst case, the encoder can prepend an identifier for the used distribution to the code output stream, as it is necessary for Huffman codes. The probability in the encoder and decoder need to be kept synchronized, which can easily be achieved in feeding the probability model with the most recently read or just decoded symbol in the encoder and decoder, respectively.

Basically, an arithmetic coder encodes a sequence of input symbols $a_{k_1}, a_{k_2}, \ldots, a_{k_S}$ as a conditional probability $P(a_{k_S}|a_{k_1}, \ldots, a_{k_{S-1}})$. In other words, it views the sequence of input symbols as a particular subinterval within the probability range $[0; 1)$. Therefore, it divides the whole probability range $[0; 1)$, according to the probabilities given in $\mathcal{P}_X$. After the first symbol is read, say symbol a_{k_1}, the corresponding probability interval $[l_1, h_1)$, with

$$l_1 = \sum_{i=1}^{k_1-1} p_i, \qquad h_1 = \sum_{i=1}^{k_1} p_i$$

is further regarded. Next, after reading a_{k_2}, the regarded subinterval is recalculated

as $[l_2, h_2)$, with

$$l_2 = l_1 + (h_1 - l_1) \cdot \sum_{i=1}^{k_2-1} p_i, \qquad h_2 = l_1 + (h_1 - l_1) \cdot \sum_{i=1}^{k_2} p_i \quad .$$

In general, after reading the input symbol a_{k_j}, the interval $[l_j, h_j)$ is regarded, with

$$l_j = l_{j-1} + (h_{j-1} - l_{j-1}) \cdot \sum_{i=1}^{k_j-1} p_i, \qquad h_j = l_{j-1} + (h_{j-1} - l_{j-1}) \cdot \sum_{i=1}^{k_j} p_i \quad , \qquad (4.1)$$

and $l_0 = 0, h_0 = 1$, initially. For the encoding, the final subinterval needs to be identified with a minimum number of bits. Therefor, the initial probability interval $[0; 1)$ is binary subdivided: on the first level, the interval $[0; 0.5)$ would be described by the identifier `0`, while $[0.5; 1)$ is referred to by `1`. Analogically, the subinterval $[0.25; 0.5)$ would be described by `01`, and so on. This procedure is repeated until the largest binary interval is found that completely fits into the final probability interval. In doing so, the input symbols are not necessarily encoded by an integer number of bits. In fact, it could even be that less than one bit per symbol is necessary to encode a symbol input sequence, e. g., if the input is the repetition of a symbol a_j with probability $p_j > 0.5$. So, the issue of the Huffman coding that up to one bit overhead may occur, does not apply for the arithmetic coding.

The exemplary subdivision of the probability and the binary coding interval for the symbol string *ccb*, with $\mathcal{A}_X = \{a, b, c\}, \mathcal{P}_X = \{0.1, 0.15, 0.75\}$ is shown in Figure 4.1: the probability interval for the symbol string is deducted in steps (i) - (iii) and is finally given by $[0.49375; 0.578125)$. In step (iv), the binary interval is subdivided until the largest interval is found that fits completely into the probability interval. In our case, this interval is described by the binary string `1000`. In practice, step (iv) is performed in parallel to steps (i)-(iii).

4.3.4 Range Coding

When it comes to practical realizations, one problem of the arithmetic coding is that it deals with decimal numbers of arbitrary high precision. Since it is not possible to implement such decimal numbers efficiently (if at all), the authors of [Mar79] propose an approximative algorithm that does not work on an actual decimal probability interval, but uses approximative integer arithmetic instead. To this end, the probability distribution interval $[0; 1)$ is modeled by an integer interval $[0; 2^l)$ with l being the available word length for the integer data type in bits. Then, the probability intervals

of the particular symbols that need to be encoded are projected onto this integer interval. The actual probability interval is then no longer given as real numbers but as quotients of integers, which causes the approximation errors. The calculation of the interval boundaries is done in analogy to (4.1) for the interval $[l_j, h_j)$:

$$l_j = l_{j-1} + \left\lfloor (h_{j-1} - l_{j-1}) \cdot \sum_{i=1}^{k_j - 1} p_i \right\rfloor, \; h_j = l_{j-1} + \left\lfloor (h_{j-1} - l_{j-1}) \cdot \sum_{i=1}^{k_j} p_i \right\rfloor ,$$

where $l_0 = 0, h_0 = 2^l$, initially. Of course, the approximation errors can grow large over time, once the absolute integer value range of the interval $\frac{h_j - l_j}{2^l}$ gets too small. To avoid this, l should be chosen appropriately large; in fact, the larger the alphabet, the larger should l be selected to minimize the effect of rounding and approximation errors. Additionally, in [Mar79] Martin proposes to rescale the interval once a certain size is underrun. An easy and efficient way to do this is to wait until $h_j < 2^{l-1}$ or $l_j \geq 2^{l-1}$. In the first case, the interval completely lies in the lower half of the value domain and one may simply continue with $h_j = 2 \cdot h_j + 1$ and $l_j = 2 \cdot l_j$. In the second case, the interval completely lies in the upper half, so the interval first needs to be shifted to the lower half, before one can continue as in the first case: $h_j = 2 \cdot (h_j - 2^{l-1}) + 1$, $l_j = 2 \cdot (l_i - 2^{l-1})$. A more sophisticated algorithm for rescaling the integer interval can be found in Bob Carpenter's implementation [Car02], as an example.

Figure 4.2 visualizes this principle with $l = 14$ for the symbol sequence from the above example in Figure 4.1. It can be seen that due to l being chosen too small, there is a rounding error of 0.4 already after step (iii), because $\left\lfloor (h_2 - l_2) \cdot \sum_{i=1}^{k_3 - 1} p_i \right\rfloor = \lfloor (4096 - 1792) \cdot 0.1 \rfloor = \lfloor 230.4 \rfloor$. This error increases dramatically in the next encoding step, if the interval is not rescaled. However, $h_j < 2^{l-1}$ or $l_j \geq 2^{l-1}$ do not apply, so more complex rescaling rules need to be applied which is left to the reader's own research of [Car02].

4.4 The Information Content of Trajectories

In this section, we explain our idea of what the information content of a trajectory is and propose a way how it can be measured. To this end, we introduce a formal model for the entropy calculation of trajectories and discuss its components and parameters.

4.4.1 What is the Information Content of Trajectories?

Any object movement, such as the migration patterns of flocks, the movement of astronomical objects or the trajectories of road vehicles can each be described by a formal

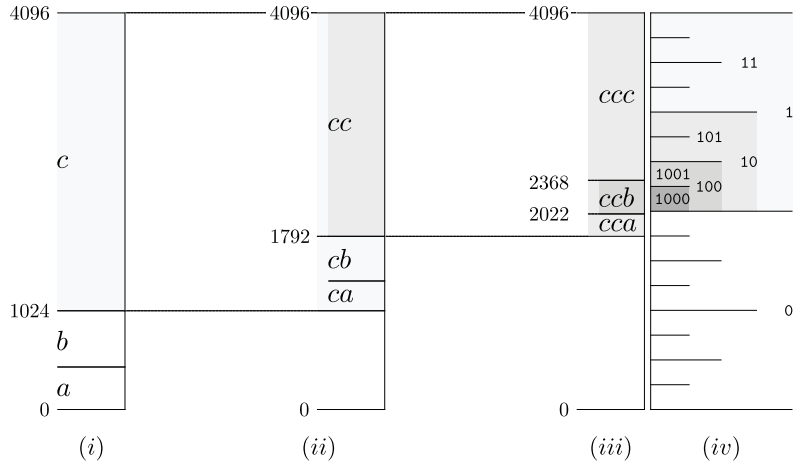


Figure 4.2: Range coding: exemplary coding of symbol string *ccb*.

model. In general, such models can be used to reconstruct or predict the movement parameters of particular objects. This is done based on previous position or other sensor measurements, exploiting the redundancy and predictability of movements. Typically, not all factors that influence the mobility of a mobile object can be modeled perfectly, as models can only approximate a real process with regard to a certain degree of inaccuracy. Therefore, the *actual* position of the object might always differ from the respective prediction. This deviation is commonly referred to as *innovation* and describes the uncertainty of the prediction process.

Of course, the outcomes for the actual innovations depend heavily on the choice and quality of the prediction model. Also, the choice of the model defines the semantic of the information that shall be extracted from a trajectory. In our case, we regard the *spatio-temporal predictability* as this information, but we have also seen in the discussion of the related work in the previous section that different models can allow for different deductions; for instance when a model is used to derive information about the navigational uncertainty of a moving object.

In this section, we investigate the information content of spatio-temporal innovations that describe the spatial deviations of a movement prediction for a distinct point in time. If this is applied to our previous discussion on the Shannon Information Content, we can identify the combination of the movement estimation, the positioning, and the innovation determination as random process that can be mapped to a random variable X. Then, each estimation innovation, being the outcome of an estimation step, refers to a symbol from the alphabet $\mathcal{A}_X$: it represents the estimation uncertainty and bears the information that was missing when the prediction was made. So, if the innovations of all position estimation steps are regarded, we can derive the information content of

a whole movement trace. Second, the alphabet $\mathcal{A}_X$ and the probability distribution $\mathcal{P}_X$ over $\mathcal{A}_X$ yield the entropy of the outcome, being the average information content.

4.4.2 How to determine the Information Content of Trajectories

We now have explained our idea of how the information content of spatio-temporal trajectories can be described. In the next step, we want to discuss how it can be measured. Basically, we have argued that each outcome x of the random variable X is determined by the employed movement estimator. To proceed with this terminology, we need to formalize all involved components: the *movement estimator* and the parts of X, namely the *alphabet* of possible outcome values $\mathcal{A}_X$ and the set of corresponding probabilities $\mathcal{P}_X$.

The movement estimator is a function θ that determines a two-dimensional position based on an observation vector $m = (m_1, \ldots, m_{M-1})$ containing previously collected position measurements[1]:

$$\theta : (\mathbb{R}^2)^{M-1} \to \mathbb{R}^2 \ , \quad \theta(m) = \hat{m}_M \ .$$

Then, the innovation i_M is the spatial difference between the estimation and the actual measurement:

$$i_M = m_M - \hat{m}_M, \quad i_M \in \mathbb{R}^2 \ .$$

We see that the innovation is a two-dimensional real vector itself and can not directly be used for the outcome x, because $\mathbb{R}^2$ is uncountably infinite, i. e., neither countable nor bounded. In order to transfer the innovation into an outcome x, we now suggest a method on how to overcome these two issues.

The innovation domain can be made countable by means of simple discretization: the real innovation vector is mapped onto a grid, with each grid node referring to a particular symbol in $\mathcal{A}_X$. The grid cell width and form are accuracy parameters, their choice is influenced by several aspects. For example, the grid cell size is influenced by the highest tolerable discretization error, i. e., the highest discretization error under which the movement model still produces reasonable results.

Once it is countable, the innovation domain can be made bounded, while still keeping *all* reasonable innovations covered by $\mathcal{A}_X$. That is, all possible positions within reach in the time period since the recent measurement need to be mappable to $\mathcal{A}_X$. Which positions can be reached depends, for example, on the movement model or the assumed position measurement noise. The limitation of the innovation domain is important,

[1] As we assume a constant measurement interval, we refer to two-dimensional observation vector elements; in case that the measurement interval is not constant, the definition needs to be adapted.

because the most probable innovations for any valid trajectory need to be mappable on it: if the limits are set too narrow, i. e., $\mathcal{A}_X$ misses reasonable innovations, such innovations could not be covered by the random variable X. Contrariwise, too wide limits would include implausible innovations in $\mathcal{A}_X$, thus increasing its entropy, which then could be higher than the actual entropy of the trajectory.

Once the movement estimator and the alphabet are known, the probability distribution $\mathcal{P}_X$ is set up by assigning a probability to each symbol in $\mathcal{A}_X$. The probability distribution is crucial for the result of the entropy determination; while effects due to an oversize alphabet can be balanced by assigning especially small probabilities to the surplus symbols, an imprecise probability distribution will have a much stronger effect on the estimation of the information content of a symbol.

The entropy of a random variable X over the alphabet $\mathcal{A}_X$ and with a probability distribution $\mathcal{P}_X$ can be determined directly using $\mathcal{A}_X$ and $\mathcal{P}_X$ using (4.3.1). On the other hand, in order to measure the information content of a trajectory, it is necessary to determine the deviations between the predicted positions and to map actual position measurements to $\mathcal{A}_X$. Then, the information content of each measurement can be determined. The information content of a complete trajectory can be stated as the sum over the information contents of all position measurements.

4.5 Exemplary Implementation of Information Measurement

We can now apply the information-theoretic model for the determination of a trajectory's information content from the previous section to a specific use case and show how to implement these components for vehicular trajectories. For this purpose, we state a number of assumptions, upon which we build our model:

1. We assume that the movement of vehicles is regular, i. e., vehicular movements can be expressed by the formulae from kinematics or Newton's laws of motion.

2. We expect that due to this regularity in movement, we can estimate a vehicle's future movement based on past position measurements and limit the area around this estimate containing all reasonable deviations.

3. We assume that, within this area, the positions closer to the estimate are more likely to match the vehicle's next position than those at the border and that the deviations from the estimate are regular as well, so that they can be *learned*.

4.5.1 Movement Estimator

As trajectory data, we consider periodic position measurements $m_i = (x_i, y_i) \in \mathbb{R}^2$ with a constant measurement interval Δt; then, the velocity ($\vec{v}$) and acceleration ($\vec{a}$) vectors of a vehicle at the position m_i at the time t_i can be approximated by the respective difference quotients over the previous measurement intervals:

$$\vec{v}_i = \frac{m_i - m_{i-1}}{\Delta t} \ , \quad \vec{a}_i = \frac{\vec{v}_i - \vec{v}_{i-1}}{\Delta t} \ .$$

With these quantities, some simple movement models, e. g., from [Kuc07], can be set up to determine the next position estimate $\hat{m}_M$: the first model merely considers the most recent position and velocity vector, as known from the LDR methods discussed in Section 3.3:

$$\theta_{vel} = m_{M-1} + \vec{v}_{M-1}\Delta t \ . \tag{4.2}$$

For the second model, it is additionally approximated that the acceleration of the moving object and extend (4.2) using the acceleration vector:

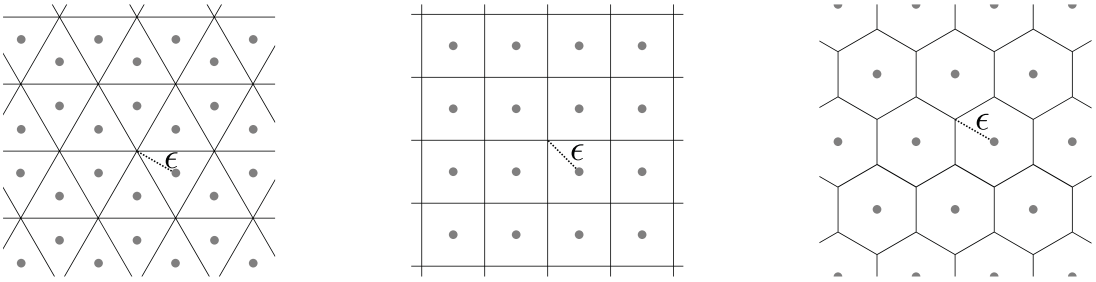
$$\theta_{acc} = m_{M-1} + \vec{v}_{M-1}\Delta t + \frac{\vec{a}_{M-1}}{2}\Delta t^2 \ . \tag{4.3}$$

Obviously, more complex and accurate movement models that use more detailed difference equations or sensor data fusion, for instance, are conceivable. However, for the context of the arithmetic coding that we will face later on, this means that all data that is used for the estimation needs to be transmitted to the remote receiver side as well. This would increase the communication load and would require also to encode or compress the additional sensor data. Therefore, we aim at a minimal data basis for the movement estimator and thus at simple movement models. Moreover, it will be shown that these simple models already perform very well and allow the compression schemes to achieve very good results.

4.5.2 The Discrete Alphabet $\mathcal{A}_X$

As described above, the innovation domain can be made countable and bounded by projecting each innovation to a grid of limited size. In this section, possible configurations for the discretization grid will be presented: we discuss several grid node alignments, how to determine reasonable grid dimensions and how to set up the grid frame.

In [PPS06], a similar technique is presented, in the form of a constructed space limited by two concentric circles and two half-lines, beginning in the two circles' center.



(a) Triangular. (b) Square. (c) Hexagonal.

Figure 4.3: Regular tessellations for the discretization grid.

The half-lines are aligned according to the movement estimation. However, this so-called *safe area* merely serves as a boolean uncertainty measure by checking whether the next position measurement lies in the safe area or not. Thus, it is an alternative to a distance and angle-based threshold measure. In their implementation of a line simplification algorithm, a measurement is dropped, if it lies in the corresponding safe area; a discretization or any further measurement coding is not performed.

Discretization Grid Node Alignment

It is clear that the specific grid design depends on the application context; in the vehicular domain, for example, a uniform approximation error for any region of the grid is desirable; this makes regularly tessellated grids—i. e., using regular triangles, squares and hexagons as shown in Figure 4.3—an interesting option. Also, the dimensions and the density of such grid cells can be easily adjusted by a maximum discretization error ϵ that directly influences the edge length of the polygons.

Each tessellation method has several influences on the model performance: with increasing number of cell edges, both the enclosed cell area and the average discretization error increase as well (cf. Figure 4.3); this leads to a smaller average discretization error of a triangular grid compared to a square or hexagonal tessellation. In turn, the smaller the average discretization error, the better the movement estimation is likely to work. Finally, the number of resulting cells is inversely proportional to the cell size; this means that the alphabet size will decrease with an increasing number of edges per cell, resulting in a smaller entropy of X. Table 4.1 lists a comparison of the grid cell sizes for the regular tessellation schemes.

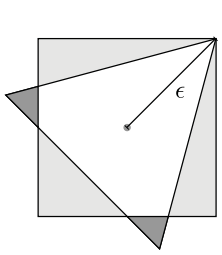
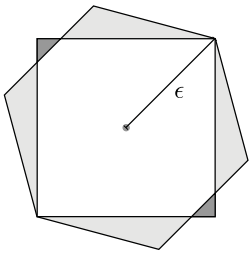
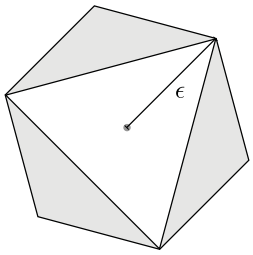
		
$A_3 = \frac{3\sqrt{3}\epsilon^2}{4}$	$A_4 = 2\epsilon^2$	$A_6 = \frac{3\sqrt{3}\epsilon^2}{2}$
$A_3 : A_4$	$A_4 : A_6$	$A_3 : A_6$
$= \frac{3\sqrt{3}\epsilon^2}{4} : 2\epsilon^2$	$2\epsilon^2 : \frac{3\sqrt{3}\epsilon^2}{2}$	$\frac{3\sqrt{3}\epsilon^2}{4} : \frac{3\sqrt{3}\epsilon^2}{2}$
$= \frac{3\sqrt{3}}{8} : 1$	$= 1 : \frac{3\sqrt{3}}{4}$	
$= 0.6495 : 1$	$= 1 : 1.2990$	$= 1 : 2$

Table 4.1: Comparison of grid cell sizes for the regular tessellation schemes.

Discretization Grid Dimensions

While setting up the grid cells is a comparably straightforward task, the limitation of the grid scope is more challenging, because the grid needs to cover all reasonable (and only those!) measurement innovations. For the use case of vehicular movements, the grid boundaries strongly depend on the possible movements of a vehicle. Therefore, in the following, a kinematic model to determine these boundaries is introduced.

For the determination of the discretization grid dimensions, we refer to a logical—not necessarily geometric—grid center, at which the grid is aligned along the movement direction. This grid center is set to the estimated next position according to the *non-accelerated* movement model (4.2), disregarding both acceleration and steering. Given this logical center, the maximal achievable spatial deviations under our model can be calculated, following a straightforward line of simple kinematic arguments:

The longitudinal dimension of the grid—i. e., the dimension along the movement direction—directly results from the highest possible deceleration dec_{max} and acceleration acc_{max} that could cause a deviation from the predictand within one measurement interval. Then, according to (4.3), the longitudinal grid dimension interval relative to the logical grid center is

$$D_{\text{lon}} = [-w_{lon}^{-}; w_{lon}^{+}]$$

with

$$w_{lon}^{-} = \frac{|dec_{max}|}{2}\Delta t^2, \quad w_{lon}^{+} = \frac{acc_{max}}{2}\Delta t^2 \ .$$

For the lateral grid dimensioning, we need to regard an extreme steering behavior to derive the highest achievable lateral deviation from the estimated position. Therefore, the vehicle is considered to pass through a curve, with the vehicle's velocity and the radius of the curve being chosen to such an extent that the lateral deviation is maximized. This deviation is limited, however, by the vehicle's velocity and the radius of the curve: given a curve radius r, a vehicle's speed is upper-bounded by the *critical cornering speed*

$$v_c(r) = \sqrt{a_l \cdot r} \ ,$$

where a_l refers to the highest possible lateral acceleration [SH82]. For the determination of a_l, we state according to Coulomb's friction law:

$$a_l \leq \mu_s \cdot g \ ,$$

where μ_s is the static friction coefficient and $g \approx 9.81 \, \frac{m}{s^2}$ is the earth's standard gravity acceleration. For the choice of μ_s, default reference values as in [Kuc07] can be used, representative examples are listed in Table 4.2. With the critical cornering speed, we can calculate the maximum lateral deviation: at a cornering of more than 90° within the time interval Δt, the lateral deviation equals the sum of the curve radius and the distance that could be driven perpendicular to the assumed driving direction (cf. Figure 4.4a). Otherwise, the lateral deviation is merely the width of the curve that has been passed thus far (cf. Figure 4.4a). Formally, the maximum lateral deviation can be expressed by the function $d_l : \mathbb{R}^{>0} \to \mathbb{R}$,

$$d_l(r) = \begin{cases} r + v_c(r)\Delta t - \frac{\pi r}{2} & v_c(r)\Delta t > \frac{\pi r}{2} \\ r \cdot (1 - \cos(\frac{v_c(r)\Delta t}{r})) & \text{else} \end{cases}$$

As depicted in Figure 4.4c, the graph of d_l resembles a square root curve: after a rapid growth with an increasing curve radius of up to approximately 80 m, the curve stagnates and features merely a minor slope. The heavy drop of all curves results from an upper bound for $v_c(r)$ that has been set to $70 \, {}^{m}/_{s}$. With a friction of $\mu = 0.55$ (tire on tar/asphalt), it can be derived from this analytical model that the grid would need to be at least $2 \cdot 2.7\,\text{m} = 5.4\,\text{m}$ wide. Figure 4.5a shows an exemplary discretization grid with previous position measurements, the position estimate and the measures on the longitudinal and lateral dimension. In the following, the lateral grid size will be referred to as $2 \cdot w_{lat}$.

However, though these grid dimensions are analytically set up, they do not necessarily need to be optimal. Further influences such as increased positioning noise levels

Tire on...	dry	wet, clean	wet, fouled	iced
farmland	0.45		0.2	<0.2
asphalt	**0.55**	**0.3**	**0.2**	**<0.2**
concrete	0.65	0.5	0.3	<0.2
cobblestone	0.6	0.4	0.3	<0.2
gravel	0.7	0.5	0.4	<0.2
tar surface	**0.55**	**0.4**	**0.3**	**<0.2**

Table 4.2: Reference values for the static friction coefficient μ_s [Kuc07].

may cause innovations to lie outside the analytically deduced boundaries. A simple countermeasure for such situations would be to add a single extra symbol to $\mathcal{A}_X$, representing outliers, which only minimally increases the alphabet size and the entropy of the random variable X. Complementary, expected or current noise statistics, such as *dilution of precision (DOP)* values that are provided by some GPS devices, could be regarded for the grid dimensioning: the grid dimensions could be simply increased by a certain size to set up a *guard zone* around the analytically determined grid dimensions, thus allowing for a higher noise level by increasing the alphabet. However, the setup of such a guard zone is nontrivial, if not only arbitrary values shall be used. For this reason, it is omitted and deferred to future work.

Discretization Grid Frame

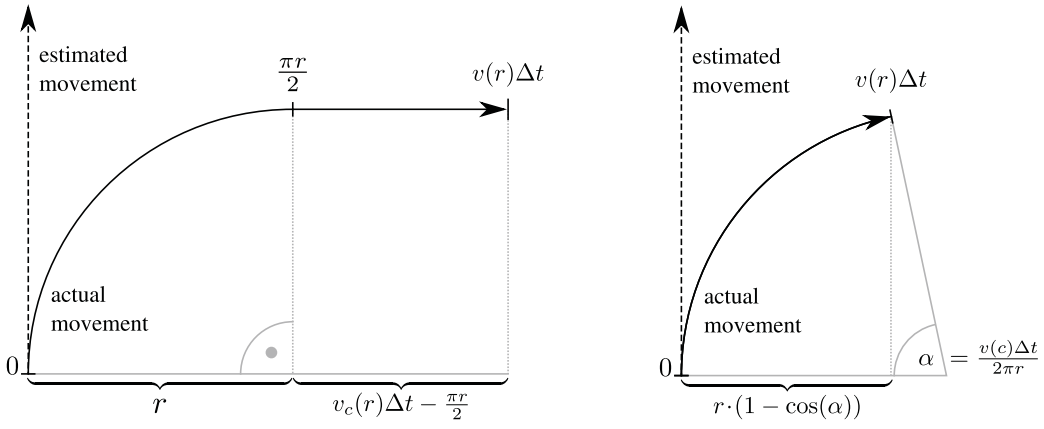
In Figure 4.4c, a rectangular frame shape for the discretization grid is assumed. While this is demonstrative and easy to implement, it does at the same time not accurately reflect a true analytic boundary for the measurement deviations from the predictand. For such a boundary, one would have to account for Coulomb's law, which proves that vehicles can not progress as far on the longitudinal dimension when cornering; if this is regarded, the resulting grid frame would feature an elliptic form.

This elliptic frame is not derived analytically, but approximated by means of *p-norms*, which are less complex to calculate at runtime; a grid node with the metric offset $(\delta_{lon}, \delta_{lat})$ from the logical grid center (the predictand) is mapped to an alphabet symbol iff

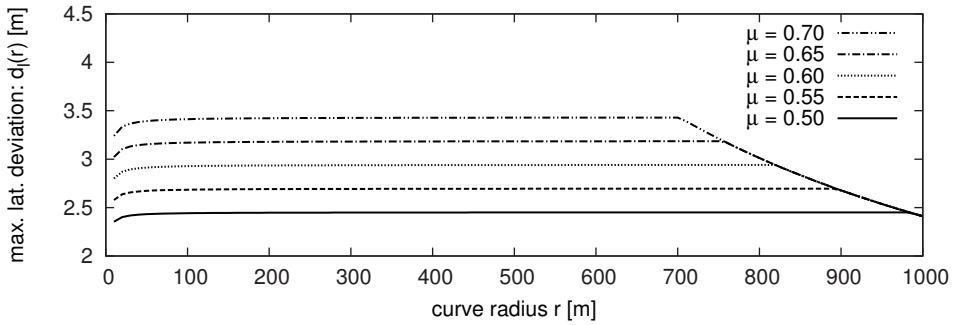
$$({\delta'_{lon}}^{\,p} + {\delta'_{lat}}^{\,p})^{\frac{1}{p}} \leq 1$$

with

$$\delta'_{lon} = \begin{cases} \frac{\delta_{lon}}{w^-_{lon}} & \text{if } \delta_{lon} < 0 \\ \frac{\delta_{lon}}{w^+_{lon}} & \text{else} \end{cases}, \qquad \delta'_{lat} = \frac{\delta_{lat}}{w_{lat}} \ .$$



(a) Cornering $> 90°$. (b) Cornering $\leq 90°$.



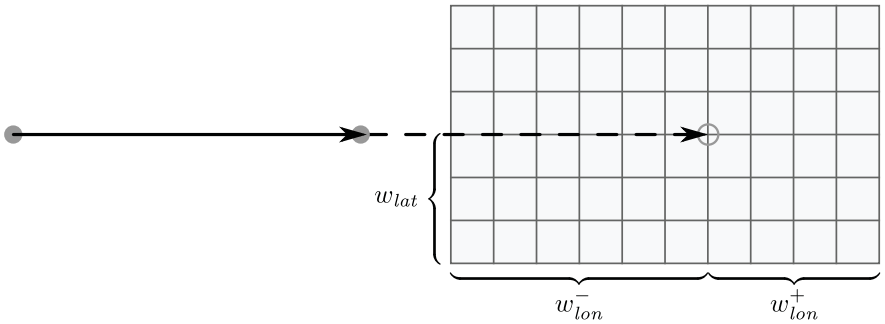
(c) Maximal lateral deviation over curve radii.

Figure 4.4: Discretization grid dimension analysis.

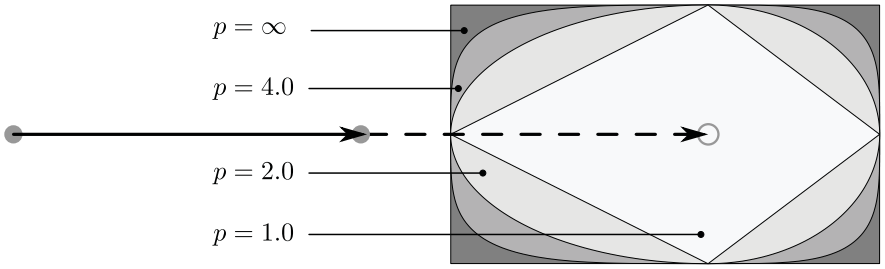
Figure 4.5b shows exemplary vector norm grid frames for $p \in \{1.0, 2.0, 4.0, \infty\}$. The ∞-norm results in a rectangular frame as shown in Figure 4.5a.

4.5.3 The Probability Distribution $\mathcal{P}_X$

The probability distribution $\mathcal{P}_X$ of the random variable X assigns a probability p_i to each symbol $a_i \in \mathcal{A}_X$, as described in Section 4.4. As mentioned earlier, the probability distribution is an important factor in the system, even more important than the alphabet. While an alphabet with more symbols than necessary can be balanced by a probability distribution that assigns very low probabilities to the unnecessary symbols, a badly chosen probability distribution can not be compensated for. Therefore, choosing the correct probability distribution that fits the actual nature of the innovations is non-trivial and needs to be regarded more closely. We now discuss several possible choices for $\mathcal{P}_X$ that we will evaluate with our arithmetic coding scheme in Section 4.6.



(a) Grid dimension measures.



(b) Vector norm grid frame options.

Figure 4.5: Discretization vector norm grid frame options.

Uniform Distribution

The simplest possible distribution is the *uniform* distribution that assigns the same probability to all symbols. Formally,

$$\forall 1 \leq i \leq N : \quad p_i = \frac{1}{N} \ , \quad N = |\mathcal{A}_X| \ .$$

Under the assumption that the employed movement estimator satisfies a reasonable degree of accuracy, small deviations from the position estimate are more likely than large ones. Therefore, this distribution does not resemble what one would expect from $\mathcal{P}_X$ in reality. However, it is a good lower-bound benchmark that can be used to validate the performance of other probability distributions.

Normal Distribution

Since we expect the large majority of position estimates to be accurate, we also assume $\mathcal{P}_X$ to be reasonably close to the normal distribution which is commonly referred to in the context of noisy position measurements [vD98]. We are aware that during an estimation process, the innovations are unlikely to be perfectly normal distributed; still, the approximation is considered adequate.

As explained above, we derive the alphabet from a two-dimensional grid, so we need to employ a *bivariate* normal distribution

$$\mathcal{N}(\mu, \Sigma), \quad \mu = (\mu_x, \mu_y), \quad \Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

with the standard deviations σ_x, σ_y and the correlation coefficient ρ [Tim02]. The distribution's mean is set according to the estimated next position: $\mu = \theta_P(m)$. While the grid always needs to be aligned using the non-accelerated movement model (4.2), because its dimensions have been derived by acceleration estimations, the mean μ of the probability distribution can be determined using other models, e. g., the accelerated movement model (4.3).

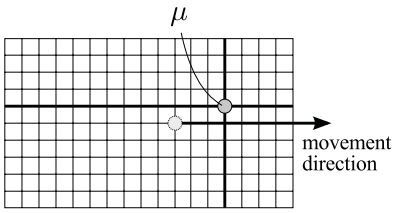
The dimensions of the innovation domain are not expected to correlate, so $\rho = 0$. However, the normal distribution is symmetric, which does not necessarily need to apply to the grid as well. Therefore, we need to find a mapping (skewing) of the probability distribution to the dimensions of the grid and propose a projection of the grid that is schematically depicted in Figure 4.6: the mean μ separates the grid into four quadrants (cf. Figure 4.6a), each of which is scaled to the dimensions $[s_x; s_y]$, where s_k is the number of standard deviations that are supposed to cover the k axis of each quadrant (cf. Figure 4.6b). Due to the scaling, the standard deviation of the distribution can be set to $\sigma = 1$ and so, the probabilities for the grid nodes can be determined using the simplified density function

$$f(x, y) = \frac{1}{2\pi} \exp(-\frac{1}{2}\left(x^2 + y^2\right))$$

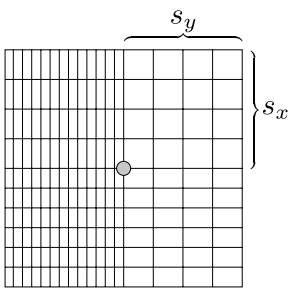
(cf. Figure 4.6c). Afterwards, the assigned probabilities need to be normalized to eliminate scaling effects, so that $\sum_{i=1}^{N} p_i = 1$ applies again (cf. Figure 4.6d).

Alternatively, asymmetric probability (e. g., log-normal) distributions could also be employed, which are partially much more complicated to configure, though.

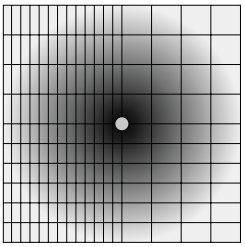
Given these first two distributions, we can now determine the respective entropy of the random variable X, i. e., the expected information content per position measurement for a given alphabet and probability distribution. and can get a first concrete impression of possible entropy values. Table 4.3 shows exemplary entropies for varying measurement intervals and accuracy bounds. For the regular square grid setup we assumed an acceleration interval $[\mathrm{dec}_{\max}; \mathrm{acc}_{\max}] = [-11; 8]\,{}^{m}/_{s^2}$. Also, we calculated the entropies for probability distributions with two different standard deviations: we chose $s_x = s_y = 3\sigma$ and $s_x = s_y = 4\sigma$, thus assuming that approximately 99.7 % and 99.99 % of all measurement innovations will lie within the grid, respectively. Please



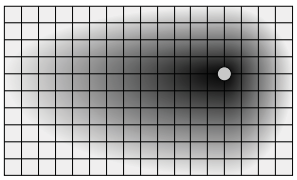
(a) Determine $\mu = \theta_P(m)$.



(b) Scale each grid quadrant to size $[s_x; s_y]$.



(c) Calculate the grid node probabilities p_i.



(d) Normalize probabilities, so $\sum_i p_i = 1$.

Figure 4.6: Skewing of the probability distribution and mapping it to the grid nodes.

note that the entropy, as a predictand, solely depends on the used movement model θ, the alphabet $\mathcal{A}_X$ and the probability distribution $\mathcal{P}_X$ of the random variable X, and not on actual measurements.

It is obvious from the table that even for very high accuracy demands, the expected average information content per symbol is very low: while off-the-shelf GPS receivers provide position measurements as fixed-point numbers with six digital places and thus can be encoded with 57 bits, the expected average information content at an accuracy bound of 0.1 m and for 1.0 Hz measurements, for example, always lies below 12 bits. This would even apply if the probability distribution of the alphabet is considered uniform. According to our model, the entropy becomes smaller with the measurement interval. This is reasonable, because the more information is provided by measurements, the lower is the missing information for a perfect estimation.

			entropy [bits]		
Δt	ϵ	$\lvert\mathcal{A}_X\rvert$	uniform	normal (3σ)	normal (4σ)
0.5 s	0.1 m	199	7.64	6.32	5.55
	0.5 m	16	4.00	2.38	1.77
	1.0 m	7	2.81	1.83	1.61
1.0 s	0.1 m	2761	11.43	10.25	9.47
	0.5 m	136	7.09	5.73	4.97
	1.0 m	41	5.36	3.86	3.20
2.0 s	0.1 m	41580	15.34	14.20	13.42
	0.5 m	1760	10.78	9.59	8.80
	1.0 m	476	8.89	7.65	6.86

Table 4.3: Exemplary alphabet configurations and entropies.

Trained Distributions

Under the assumption that vehicular movements, disregarding the terrain or surrounding, follow a general principle or pattern, we also suppose that there is a generally fitting distribution that can be found, or *learned*. Since the alphabet is discrete and finite, such a distribution can be obtained from a collection of previously observed traces, referred to as the *training set* (S_3^{train}).

To create such a learned distribution, every position measurement in the training set is mapped to its corresponding alphabet symbol using a movement estimator, for instance one of those presented in Subsection 4.5.1, and the grid as described in Subsection 4.5.2. For each symbol $a_i \in \mathcal{A}_X$, a counter n_i is maintained, which is increased upon every occurrence of the respective symbol in the training set. After all training measurements have been processed, the probability p_i can be stated as

$$p_i = \frac{n_i}{\sum_{i=1}^{N} n_i} \ .$$

According to the definition in Subsection 4.4.1, all symbol probabilities need to be nonzero. We therefore define $n'_i := max\{n_i, 1\}$ and p'_i analogously to p_i, resulting in a close approximation of p_i that can be used for the determination of the information content and for the arithmetic coding of trajectories. If each symbol occurs at least once in the training set, $n'_i = n_i$ and thus $p'_i = p_i$.

A Posteriori Distributions

A special case of the trained distributions is the *a posteriori* distribution. For this, one particular trajectory is used to train the distribution, i. e., the result reflects exactly the innovation distribution for this trajectory. With this knowledge, an arithmetic coder will be able to determine an excellent code. Though this kind of distribution can not be used for real-world implementations, because the distribution is obviously unknown unless the complete trajectory has been collected, it can serve as an upper bound approximation for the compression performance for a trajectory, especially to evaluate the compression ratio achieved with other probability distribution settings.

However, even the usage of a posteriori distributions does not guarantee optimal compression results; an optimal result could be achieved, if the next measurement with a discretization to the symbol a_k would be known; in this case,

$$p_i \in \mathcal{P}_X, \quad p_i = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{else} \end{cases} .$$

Then, this particular $\mathcal{P}_X$ would be the Dirac measure for $a_k \in \mathcal{A}_X$.

Contextual Distributions

Next to the probability models that apply the same probability distribution to every observed measurement innovation, it could be beneficial to work with probability distributions that are purpose-made and tailored to a set of distinctive contexts or situations. For example, when a vehicle is measuring its position while standing at a red traffic light, a different probability distribution might fit best than for the situation when the same vehicle drives on an on-ramp to enter a high-speed motorway. Therefore, in the following, we discuss *contextual distributions*: a contextual distribution is, strictly speaking, no self-contained probability distribution but rather a way to combine multiple distributions into a single probability model and thus can be used as an extension instead of a stand-alone and exclusive alternative. Also, different kinds of distributions can be employed within each context in order to assemble the optimal fit for each context.

For the contextual distributions, it can be assumed that the actual distribution of measurement innovations correlates with the measurement vehicles' current movement parameters, i. e., velocity or acceleration.

The Movement Context Model: cntxt_M The simplest regarded contextual distribution differentiates between two situations with a movement filter: there is one trained

distribution each for measured velocities of up to $1\,{}^{m}/_{s}$ and for velocities above this threshold. With this model, we want to make use of the assumptions that at such very low velocities or halts, the probability distribution will be significantly different from those for movement periods with higher velocities and that the movement estimator will likely provide good results at close-to-zero velocities.

The Velocity Context Model: $\mathbf{cntxt}_V$ Going one step further, one could argue that the probability distributions for every velocity or at least velocity range of a certain, regular width differs significantly from the others. To examine this assumption, this contextual distribution contains ten trained probability distributions, the first nine of which cover the increasingly ordered scalar velocity ranges of $3\,{}^{m}/_{s}$ each. The last distribution covers velocities of $27\,{}^{m}/_{s}$ and larger. So, a measurement in the trajectory is assigned to the distribution D_i with

$$i = \min\left(\left\lfloor \frac{|\vec{v}|}{3} \right\rfloor + 1,\ 10\right)$$

where $\vec{v}$ is the previously observed velocity.

The Clustered Velocity Model: $\mathbf{cntxt}_C$ After we had a first look at the probability distributions created with the velocity context model, we recognized that many of them, of course especially the ones for close velocities, were very similar to each other. It was therefore decided to analyze this similarity further and calculated distributions for 40 scalar velocity classes with a coverage of $1\,{}^{m}/_{s}$ each. Again, the last class covers all velocities of $39\,{}^{m}/_{s}$ and larger.

The goal is to find groups of distributions that are similar enough, so that they can be aggregated or clustered. This should reduce the number of distributions so that these can be trained more efficiently. Therefore, after the distributions were created, a *k-means* clustering was applied to the data. With the k-means clustering algorithm, a set of data points can be clustered into k groups, so that all points have a minimal distance to the center points (*centroids*) of the group they belong to, with respect to a particular distance metric. Before the algorithm starts, initial centroids are selected, which are then iteratively updated until a stop criterion, for example after a certain number of iterations, applies. The k-means algorithm is a little problematic, because the initial centroids need to be chosen carefully, because a bad choice could result in empty clusters at the end of the run.

An N-dimensional linear space, $N = |\mathcal{A}_X|$, was set up, whereby each calculated distribution represents a single point in this space and the Euclidean metric can

velocity class index i	k = 2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1
1	1	2	2	2	2	2	2	2	2
2	1	2	2	2	3	3	3	3	3
3	1	2	2	3	3	3	3	4	4
4	1	2	2	3	3	3	4	4	4
5	1	2	2	3	3	3	4	4	4
6	1	2	2	3	3	3	4	4	4
7	1	2	2	3	3	3	4	4	4
8	1	2	2	3	3	3	3	4	4
9	1	2	2	3	3	3	3	4	4
10	1	2	2	3	3	3	3	5	4
11	1	2	2	3	4	3	4	5	5
12	1	2	2	3	4	3	5	5	5
13	1	2	2	3	4	3	5	5	5
14	1	2	3	3	4	4	5	5	5
15	1	2	3	4	5	4	6	6	6
16	1	2	3	4	5	4	6	6	6
17	1	1	3	4	5	4	6	6	6
18	1	1	3	4	5	4	6	6	6
19	1	1	3	4	5	4	6	6	6
20	1	1	3	4	5	4	6	6	6
21	1	1	3	4	5	4	6	6	6
22	1	1	3	4	5	4	6	6	6
23	1	1	3	4	5	4	6	6	6
24	1	1	3	4	5	4	6	6	6
25	1	1	3	4	5	5	6	7	6
26	1	1	3	4	5	5	6	7	6
27	1	1	3	4	5	5	6	7	7
28	1	1	3	4	5	5	6	7	7
29	1	1	3	4	5	5	6	7	6
30	1	1	3	4	5	5	6	7	7
31	1	1	3	4	5	5	7	7	7
32	1	1	3	4	5	5	7	7	7
33	1	1	3	4	5	5	7	7	7
34	1	1	3	4	5	5	7	7	7
35	1	1	3	4	5	5	7	7	7
36	1	1	3	4	5	5	7	7	7
37	1	1	3	4	5	5	7	7	7
38	1	1	3	4	5	5	7	7	7
39	1	2	2	3	4	6	5	4	8

Table 4.4: Results of the distribution clustering for $\epsilon = 0.25\,\mathrm{m}$ and a square-tiled grid.

be employed for the distance calculations in the linear space. For the choice of the initial centroids, we calculate both the mean average μ' over all distributions' means $\mu_j, 1 \leq j \leq 40$—which are all at least very close to the logical grid center, anyway—and the standard deviations σ_j for each distribution. Then, the interval $[\min(\sigma_1, \ldots, \sigma_{40}), \max(\sigma_1, \ldots, \sigma_{40})]$ is segmented evenly from the smallest to the largest standard deviation into $k-1$ subintervals, thereby retrieving k standard deviations $\langle \sigma'_i \rangle_{1 \leq i \leq k}$. Finally, k distributions are constructed by calculating Normal distributions with the mean average μ' and the standard deviations $\langle \sigma' \rangle$, as described on pages 74f. We use these distributions as initial centroids.

We ran the algorithm with 1000 iterations and for $1 < k \leq 10$. Exemplary representative results of the clustering for $\epsilon = 0.25\,\mathrm{m}$ and a square-tiled grid are shown in Table 4.4; the created clusters are highlighted in color. It clearly shows that for $i = 0$

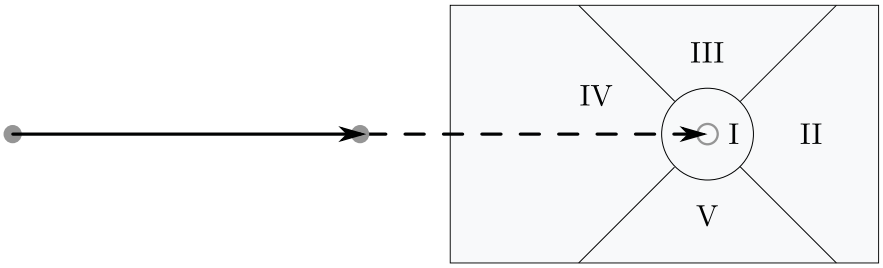


Figure 4.7: Acceleration Context Model: sectors around the logical grid center.

and $i = 39$, i. e., for the classes covering halts and the highest velocities, respectively, separate clusters were identified. Also, for $k > 2$, there is an obvious cluster threshold around $i = 15$. Based on this results, we decided on using the velocity clusters found with $k = 4$; we merely made two minor adaptations, after we had reviewed the distributions of the resulting clusters: first, we created 4 independent clusters and did not use one cluster for two velocity ranges, as the clustering algorithm indicated. Second, we started the third cluster at $i = 15$, because this threshold dominated in the clustering results over all values of k. So, we trained probability distributions for the velocity intervals $[0; 1)$, $[1; 15)$, $[15, 39)$, and $[39, \infty)$ (in $^{m}/_{s}$).

The Acceleration Vector Context Model: cntxt$_A$ The last contextual distribution that we consider does not regard scalar velocities, but acceleration vectors. Technically, the acceleration vectors can be visualized by drawing a vector from the logical grid center (the non-accelerated position estimation) to the actual position measurement within the grid. So, the space around the logical grid center is divided into five sectors, as shown in Figure 4.7. Sector I catches all position measurements with a small derived acceleration, the diameter depends on a scalar threshold, which is set to $0.3\,^{m}/_{s^2}$. For each of the sectors, we maintain a trained probability distribution. Once a position measurement is mapped onto the grid, the distribution into the sector of which the measurement has been mapped, is used for $\mathcal{P}_X$.

Adaptive Distributions

In contrast to predefined distributions, we also regard distributions that evolve over time. Such *adaptive distribution* models start with an initial setup, e. g., a uniform distribution, and evaluate the observed symbol occurrences to converge towards the actual distribution. Of course, more realistic distributions can also be used as initial setups. Common adaptive distribution models regard n-gram relations in the symbol stream, e. g., constructing a unigram probability distribution in the simplest case.

The advantage of this approach is that the resulting distribution is an optimal match for all previously, and at best upcoming symbol occurrences. Of course, this will only pay off for sufficiently long trajectories, i. e., if the ratio of trajectory length to the alphabet size satisfies a particular threshold value; otherwise, the learned distribution reaches a representative version too lately. Then, too few position measurements in the trajectory can benefit from the learned distribution to compensate for the learning phase, during which the distribution may be far from being an acceptable fit.

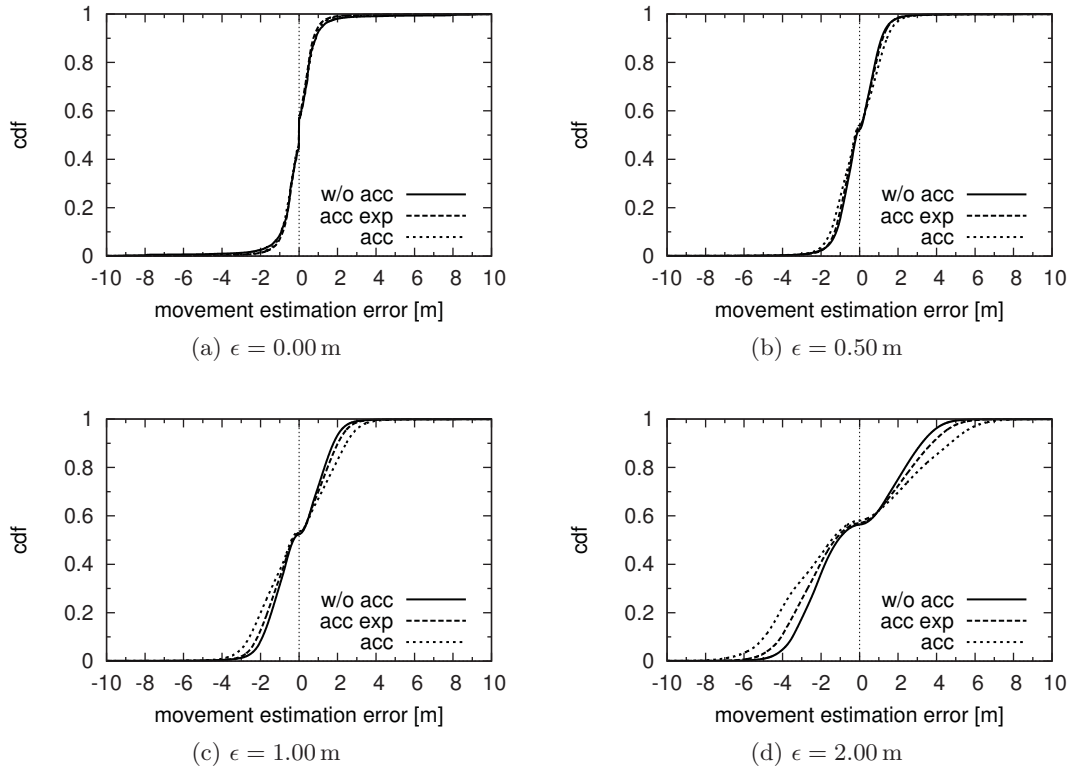
4.5.4 Model Implementation: An Arithmetic Coder

The estimation results presented in Table 4.3 encouraged us to build an arithmetic coder [Abr60] based on our formal model, since compression rates of approximately 90 % could be expected, even at tight accuracy bounds. For the implementation, we used the arithmetic coding project of Bob Carpenter [Car02], version 1.1 as a basis.

The only modification to our formal model lies in the handling of outlier measurements: instead of adding an extra symbol to $\mathcal{A}_X$, the encoding stops upon outlier measurements. This is inevitable, because the mapping of a discretized innovation to a symbol needs to be bijective; this is not fulfilled in case of outliers. Once an innovation can not be mapped to a grid node, it is not possible to retrieve a valid grid node in the decoding process. Therefore, in this case the symbol stream is terminated with the *End Of Stream* symbol, the probability distribution $\mathcal{P}_X$ and the estimator are reset and a new encoding begins. This is an undesirable situation, because at least two measured positions need to be transmitted uncoded; so, even with the mentioned *guard zone*, using a robust estimator is crucial for the compression performance.

4.6 Evaluation

In the previous section, we presented an exemplary implementation for the measurement of the information content for vehicular movement trajectories and also described the realization of an arithmetic coder based on our model. In doing so, we proposed many options for the movement estimator, the alphabet setup, i. e., the grid node arrangement and grid frame shape, and the probability distribution. In this section, we evaluate the model with regard to all these options. We analyze the movement estimator to begin with and find the best one to use in our context. We then focus on the alphabet and examine the influences of the different parameters discussed in Subsection 4.5.2. Finally, we turn on the probability distributions and their impact on the compression performance of the arithmetic coder.



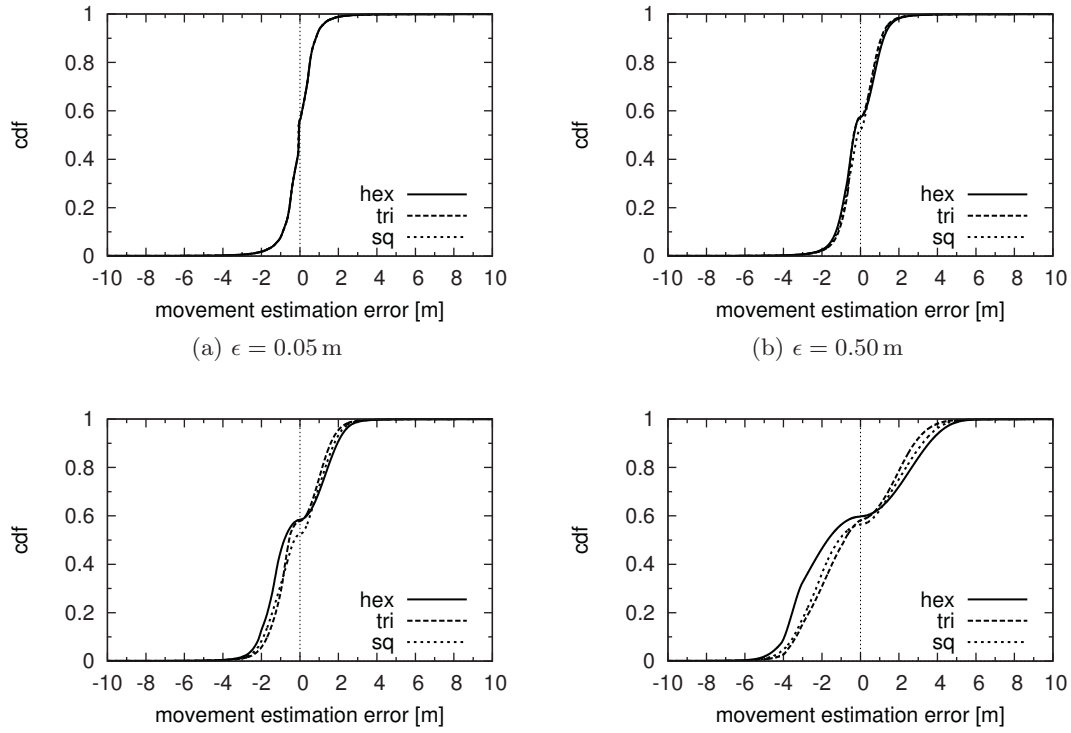
(a) $\epsilon = 0.00\,\mathrm{m}$ (b) $\epsilon = 0.50\,\mathrm{m}$

(c) $\epsilon = 1.00\,\mathrm{m}$ (d) $\epsilon = 2.00\,\mathrm{m}$

Figure 4.8: Movement estimation error analysis for different movement models (high velocity trajectory set S_2^{high}).

4.6.1 Movement Estimation and Discretization

Since it is essential to use an accurate and robust movement model, we performed movement estimations with the non-accelerated and accelerated movement estimators. In particular, we were interested in the influence of the discretization and therefore regard the movement estimation inaccuracies for varying discretization steps; the results are shown as cumulative distribution functions in Figure 4.8. A negative error occurs, when the movement estimation overshoots the actual measurement (i. e., the longitudinal error is negative, as viewed from the estimation). Without discretization (i. e., $\epsilon = 0$), the absolute error is below 2 m in 90-95% of the cases, with slightly worse results for the low velocity topologies, which we left out due to the strong similarity. With increasing discretization tolerance ϵ, the grid gets coarser and the discretized measurements become slightly distorted. This distortion lowers the quality of the estimator's observation vector and has a direct influence on the information content and the compression performance.



(a) $\epsilon = 0.05\,\mathrm{m}$ (b) $\epsilon = 0.50\,\mathrm{m}$

(c) $\epsilon = 1.00\,\mathrm{m}$ (d) $\epsilon = 2.00\,\mathrm{m}$

Figure 4.9: Movement estimation error analysis for non-acc. movements and different grid node alignments (high velocity trajectory set S_2^{high}).

Though the distributions are basically point symmetric, there is a minor bias for all accuracy bounds and estimators, so the inflection points (for $\epsilon = 0.0$) and saddle points (for $\epsilon > 0.0$) of all cumulative distributions functions lie above 0.5. This indicates that the movement estimators tend to overshoot, causing the negative estimation errors to prevail.

Unexpectedly, the non-accelerated estimator (*w/o acc*) provides more accurate results than the accelerated variant (*acc*). The latter is more impaired by the discretization, because it derives acceleration values from distorted velocities. To reduce this effect, we amended the accelerated estimator by smoothing the computed acceleration values exponentially (*acc exp*). According to (4.3), this new estimator can be described as

$$\begin{aligned} \theta_{acc} &= m_{M-1} + \vec{v}_{M-1}\Delta t + \frac{\vec{a'}_{M-1}}{2}\Delta t^2 \ , \\ \vec{a}'_{M-1} &= w \cdot \vec{a'}_{M-2} + (1-w) \cdot \vec{a}_{M-1} \ . \end{aligned} \tag{4.4}$$

For our experiments, we used $w = 0.5$. This improves the situation, as the cdf graph is now denser around an error of 0 m, but does not provide the same robustness as the non-accelerated estimator (cf. Figure 4.8). The movement estimator is a key component within the model and in all of our tests, the worse results for the acceleration-supported estimators directly resulted in lower compression ratios. We therefore only regard the non-accelerated model in the remainder of our evaluation.

The discretization grid node alignment also influences the performance of the movement estimator as discussed in Subsection 4.5.2. Figure 4.9 shows the movement estimation error analysis for the non-accelerated movement estimator and the three discussed tessellation schemes with triangular (tri), square (sq), and hexagonal (hex) grid tessellation. The estimation results are very close to each other, but the differences are still clearly to be seen: the regular triangular tessellation provides the highest node density and thus, the movement estimations are better than for the other two tessellations. For the extreme, the hexagonal tessellation results in the worst estimation accuracy, because of the lowest grid node density.

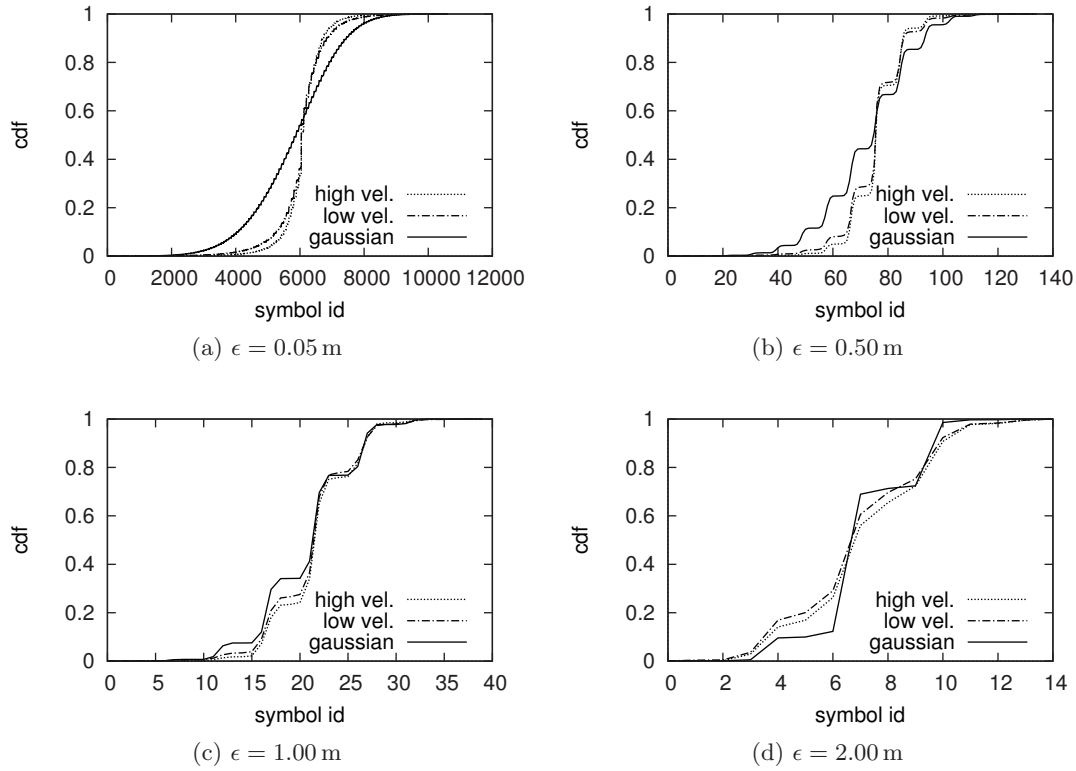
4.6.2 Gaussian Probability Distribution

To evaluate the goodness of fit for the assumed Gaussian probability distribution, we compare it to the actual cumulative distribution functions for both topologies over the respective discrete alphabet in Figure 4.10. To this end, we serialized the two-dimensional distributions over the grid to one-dimensional distributions over the ordered symbol set to gain a better overview: we concatenated the symbols from cross-sections along the lateral grid axis, causing stepped curves, where each "step" refers to one such cross-section. The cdf graphs for the normal distribution resemble the empirically determined ones, though the distributions for both the low and high velocity topologies are denser, especially for lower values of ϵ. This basically confirms our assumption that the Gaussian distribution is a reasonable approximation for $\mathcal{P}_X$, though it is also quite obvious that there is room for improvements. We will evaluate the impact of the other distribution models at the end of the next subsection.

4.6.3 Compression

Basic Configuration and Benchmarks

At first, we evaluate the compression performance of our proposed arithmetic coding scheme in a basic configuration; that is, we use a rectangular shaped discretization grid with a grid node alignment following the square grid cell tessellation. We run the encodings with the uniform and a posteriori distributions to retrieve lower and



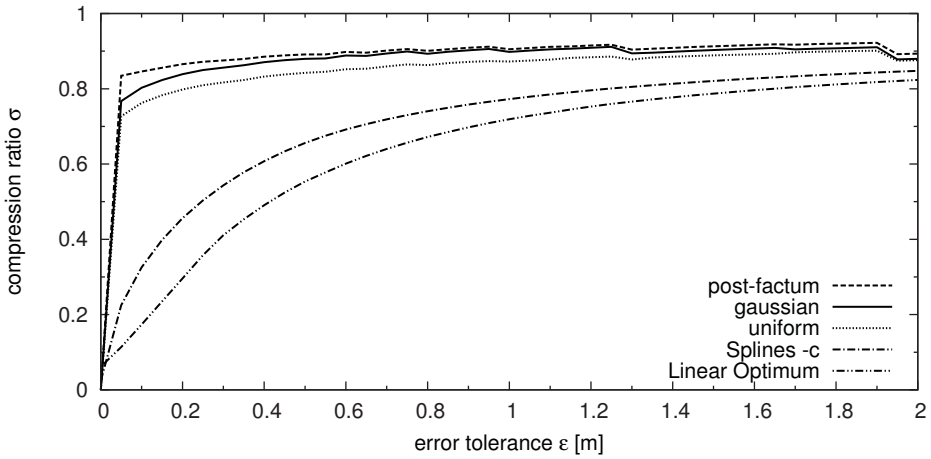
(a) $\epsilon = 0.05\,\mathrm{m}$ (b) $\epsilon = 0.50\,\mathrm{m}$

(c) $\epsilon = 1.00\,\mathrm{m}$ (d) $\epsilon = 2.00\,\mathrm{m}$

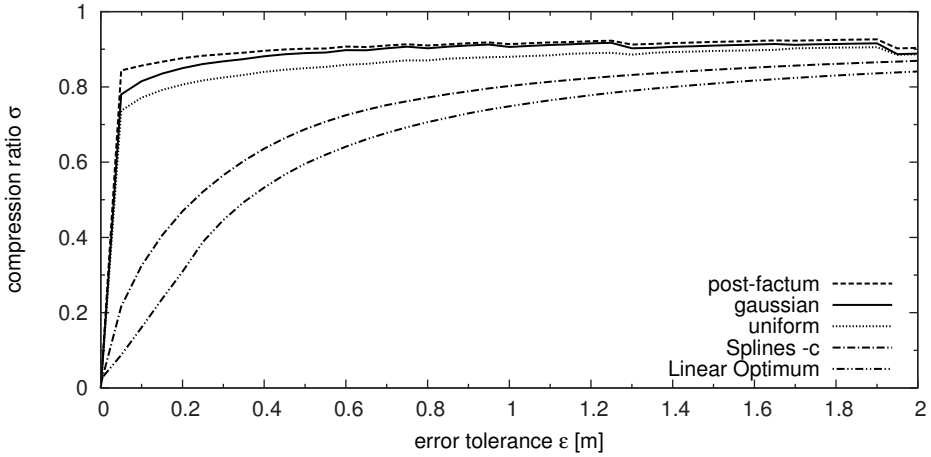
Figure 4.10: Cumulative distribution analysis of the probability distribution $\mathcal{P}_X$.

upper bounds for the compression performance, respectively and use the normal distribution as a more realistic candidate for $\mathcal{P}_X$. With this setup, we compress the vehicular trajectories with 250 elements each that we already used for the evaluation of the geometric compression algorithms. In doing so, we can directly evaluate the compression performances of the arithmetic coder, using the optimal linear and the context-loosened spline approaches from Chapter 3 as benchmarks.

Figure 4.11 shows the compression results for this basic arithmetic setup and for the geometric benchmark results against the discretization threshold ϵ. For both the low and high velocity trajectory sets, the same ranking of compression techniques is visible: the optimal line simplification algorithm performs worst, being outperformed by the cubic spline approach, as we have already seen in Figure 3.8. The arithmetic coding approach performs best, especially for very tight accuracy bounds of $\epsilon < 1.0\,\mathrm{m}$. We can see that even if a uniform probability distribution is used for the probability distribution $\mathcal{P}_X$, the arithmetic coding achieves better compression ratios than the spline-based approach. For the normal distribution, the compression ratios improve significantly once more, with respect to the uniform distribution. The results for



(a) Low velocity trajectories (S_1^{low}).



(b) High velocity trajectories (S_1^{high}).

Figure 4.11: Basic compression ratios of geometric benchmark algorithms.

using a posteriori knowledge are clearly best, especially for $\epsilon < 0.5\,\text{m}$; thereafter, they are almost reached by the compression ratios with the normal distribution assumption. This underlines our findings from the probability distribution analysis that for growing ϵ, the impact of exactly meeting the probability distribution decreases. Because the basic arithmetic coding configuration already outperforms the current state-of-the-art compression schemes, we will use it as benchmark for all enhanced code model configurations in the remainder of this chapter.

In general, the compression ratio graphs are very close to each other, so the following figures only show the σ interval $[0.6; 1.0]$. The figures also show only the results for the high velocity trajectories. This is because all effects that we could find occurred

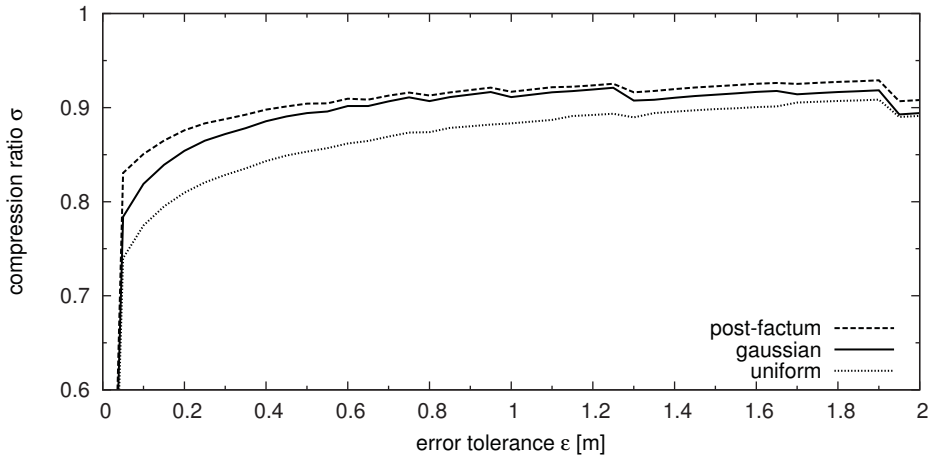


Figure 4.12: Basic compression ratios for uncut high velocity trajectory set (S_2^{high}).

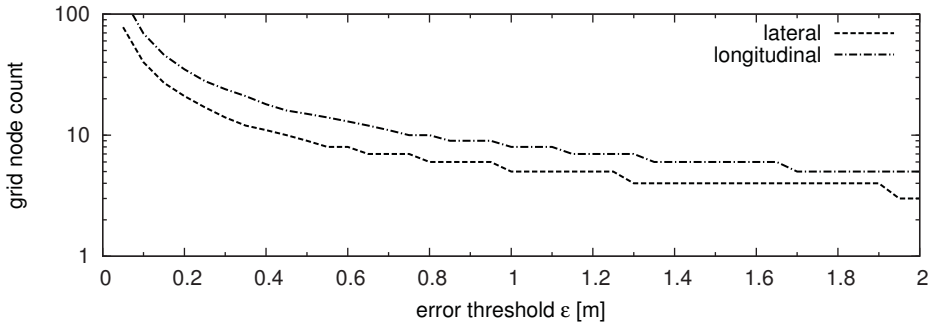


Figure 4.13: Grid node counts over increasing accuracy threshold.

equally for both the low and high velocity settings; the only difference is that for our complete evaluation, the compression results for the low velocity trajectories were slightly worse, by a difference of approximately 1.0 to 1.5 percent. For this reason, we only regard the high velocity trajectory set in our evaluation.

Figure 4.12 shows the compression results for uncut trajectories. The results do not differ considerably, because the input stream length is less relevant for the arithmetic coding, in case a static probability distribution is used. In contrast to that, we have seen in [KKKM11] that the trajectory size does in fact influence the compression performance of geometric schemes.

An interesting effect is the clearly visible drops of the compression ratios for higher ϵ that occur for all distributions and trajectory classes alike. These drops originate from the discretization grid that is getting coarser for growing ϵ. As we have seen in Figure 4.8, this reduces the quality of the movement estimations, which in turn causes larger and noisier predictions. The basic cause can be seen in Figure 4.13

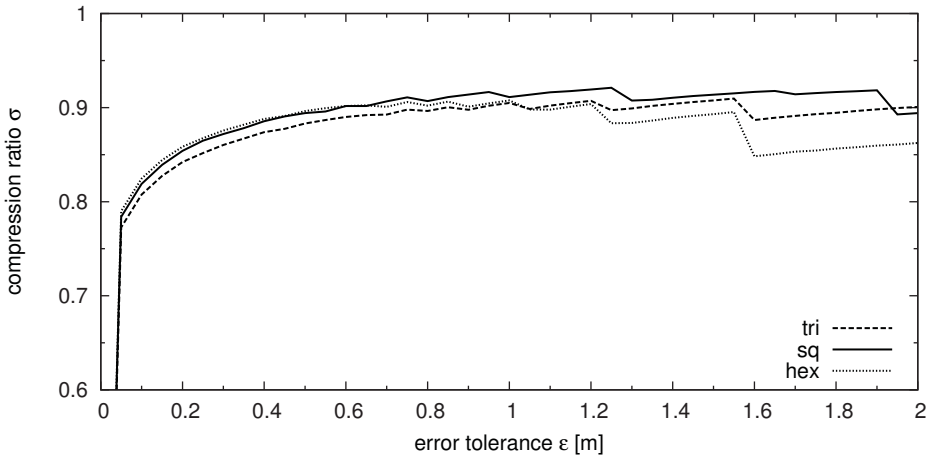


Figure 4.14: Compression ratios for different grid node alignments (high velocity trajectory set S_2^{high}).

that depicts the number of discretization grid cells per dimension over an increasing accuracy threshold.

Impact of Discretization Grid Nodes Alignments

As we have seen in Subsection 4.6.1, the alignment of the grid nodes has a direct effect on the movement estimator in making the estimations less accurate. However, the node density is affected as well, which could balance this effect, so that good compression results could nevertheless be achieved. Therefore, we evaluated the compression performance for the three tessellation schemes using the uncut low and high velocity trajectories, the results are depicted in Figure 4.14.

While the compression ratios are again slightly better for the high velocity trajectories, the compression result ranking is the same for both settings: the hexagonal grid node alignment performs slightly best for $\epsilon \leq 0.6$ and then drops below the performance curve for the square tessellation. This shows that the low node density may cause inaccurate movement estimations, but the lower entropy resulting from the smaller alphabet outbalances this effect. For $\epsilon > 1.0$, the discretization inaccuracies are too immense to be compensated by the node density and as a consequence, the compression ratio drops and is worse than for the other two tessellation schemes. Though the triangular grid node alignment allows for better movement estimations, the node density with this scheme is twice as high as for its hexagonal counterpart. This results in the worst compression ratios for $\epsilon < 1.0$, and the performance of the square gridding can only be exceeded for $\epsilon > 1.9$. Thus, the square tessellation turns

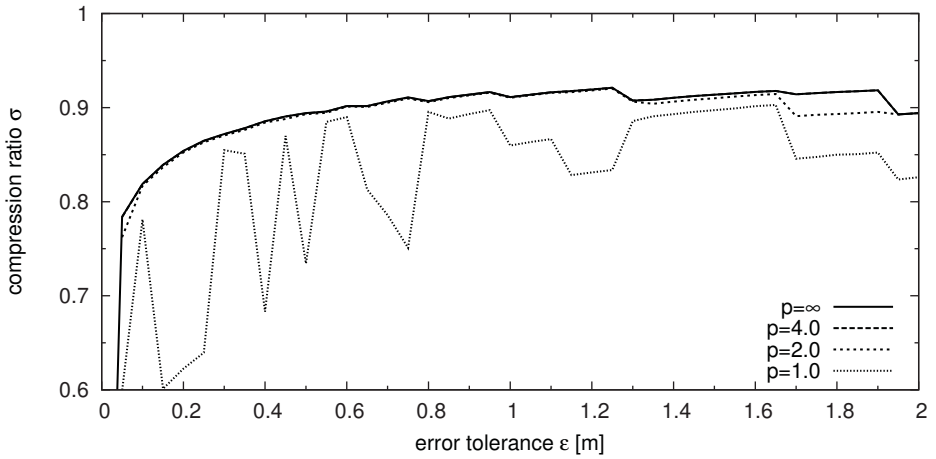


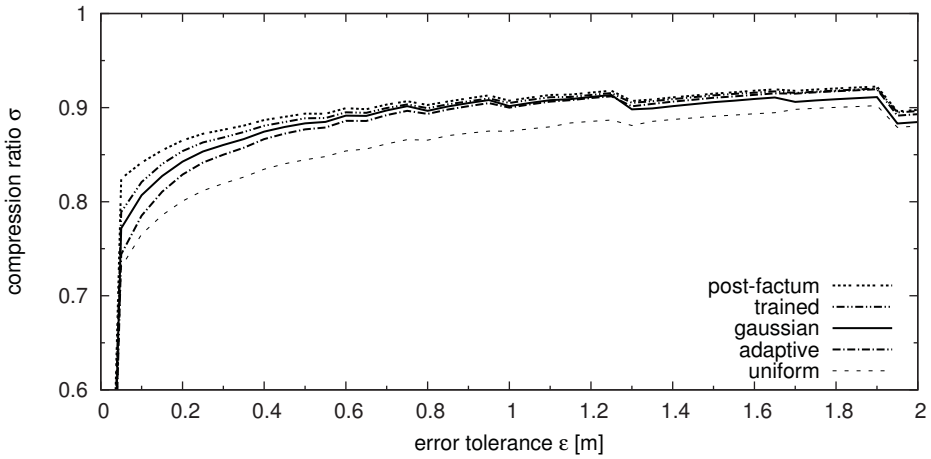
Figure 4.15: Compression ratios for different grid frames (high velocity trajectory set S_2^{high}).

out to be a very good choice overall, as it is a good trade-off between the alphabet size and the estimation accuracy.

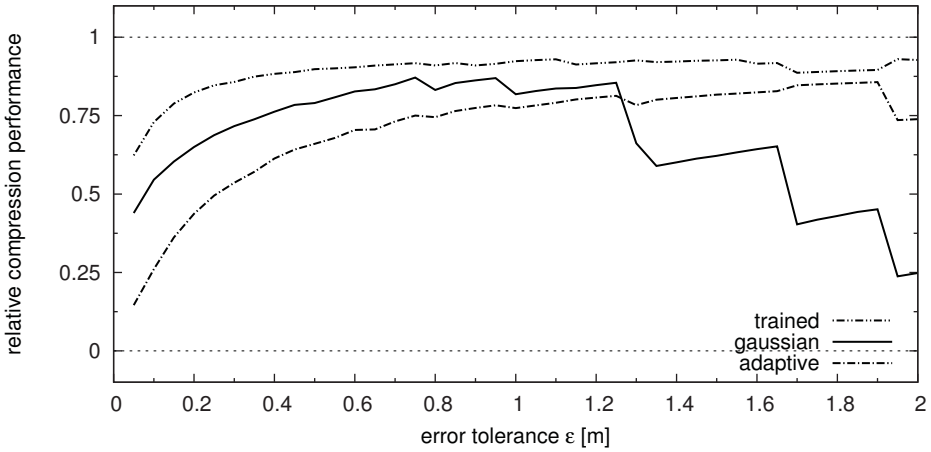
Based on this impact of the discretization node alignments, one can not simply state a generally optimal choice of the discretization node alignments. However, we prefer the square grid tessellation for practical implementations due to its algorithmic simplicity and its good overall performance.

Impact of Discretization Grid Frame Shape

At the end of Subsection 4.5.2, we presented a way to determine a more realistic shape of the discretization grid by using p-vector norms. According to our parameter selection, Figure 4.15 depicts the compression ratios for the grid vector norm frame parameters $p \in \{1.0, 2.0, 4.0, \infty\}$. It can clearly be seen that the compression performance improves with increasing p, i. e., with a grid frame more similar to the rectangular shape of the basic configuration. Since a larger p causes a larger alphabet, this result is a good indicator that a more tolerant choice of the grid frame (and thus the alphabet) size is rather beneficial than disadvantageous. This is an interesting result, because it confirms our assumption that a higher tolerance for outlier measurements is more important to the overall compression performance than an exact determination and dimensioning of the code symbol alphabet and that the effects of an oversize alphabet are outbalanced by a fitting probability distribution.



(a) Absolute results.



(b) Relative results.

Figure 4.16: Compression ratios for non-contextual probability distributions.

Impact of Probability Distributions

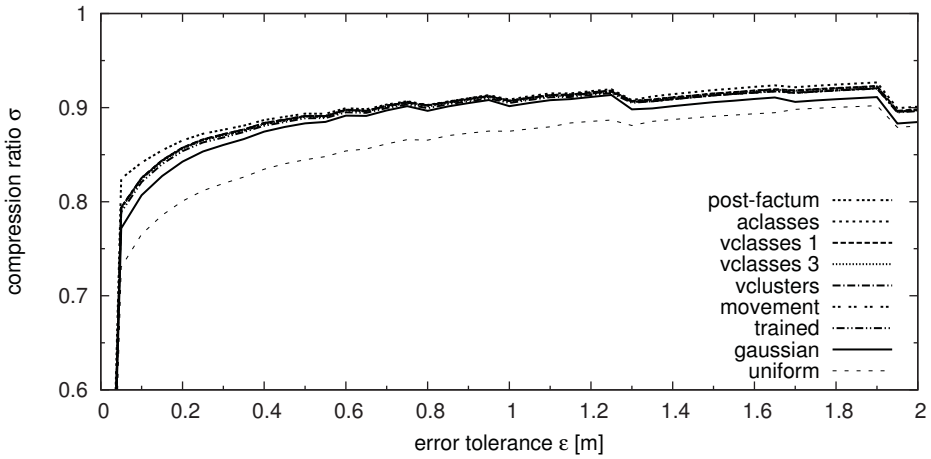
In this section we will analyze the compression performance of the trained, adaptive, and contextual distributions as described above. We trained each distribution with the training set S_3^{train} as described in Section 2.2 and used these trained distributions for the arithmetic coding for all traces in the test set S_3^{test}.

Figure 4.16a shows the compression results of the non-contextual, i. e., the trained and the adaptive, probability distributions compared to the ones obtained with the basic configuration. At first sight, the compression results seem to be quite close to each other, but it can be seen that the trained distribution yields a better compression than the normal distribution and that its result graph quickly converges against the

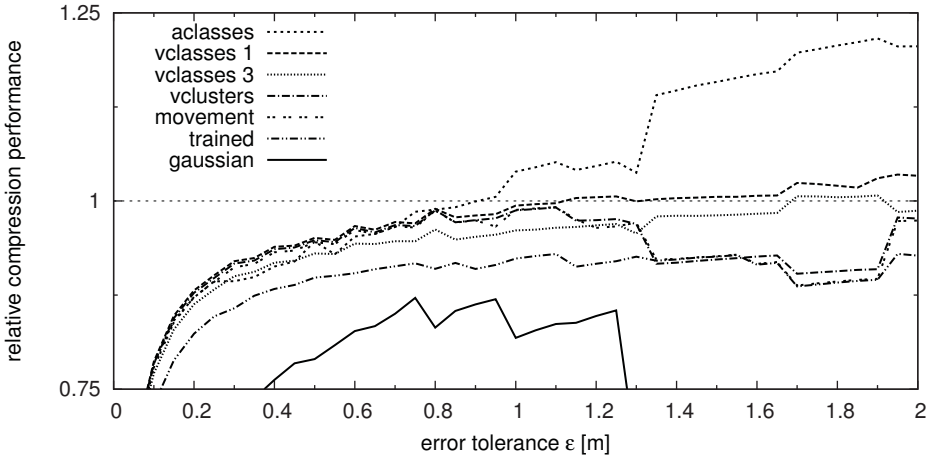
upper bound set up by the post-factum distribution results. This shows that the training of distributions in fact brings a benefit over the assumption that the innovations follow the normal distribution. The adaptive distribution initially performs worse, which is due to the size of the alphabet for small values of ϵ. Naturally, it takes long trajectories to learn the correct distribution for large alphabets and for $\epsilon < 1.3$, the trajectories are not long enough (or the alphabet is too large) to bring a benefit over the Gaussian distribution. After that, the alphabet is small enough so the distribution can be adapted fast enough and the resulting compression ratio exceeds the ration obtained with the normal distribution. Figure 4.16b shows the compression performance in relation to the interval spanned by the lower and upper bounds, given by the a posteriori and uniform distribution results, respectively: in this figure, a zero value means that a compression equals the achievement with a uniform distribution and a value of one means that a compression ratio as good as with a posteriori knowledge could be achieved. In this scaling, the effects described above can be seen more clearly.

The compression results for the contextual distributions can be seen in Figure 4.17, again both absolute and relative to the upper and lower compression bounds. For the relative analysis, we have focused on the ratios from 0.75 to 1.25 to see the effects more clearly. We directly see that all contextual distributions yield better compression results than the normal and the simple trained distribution. Though no result clearly stands out for $\epsilon < 1.0$, the acceleration vector context model (cntxt_A) achieves the best compression rates for $\epsilon > 0.8$. It even exceeds the performance of the compression with post-factum distributions. This is possible, because of the multitude of distributions that are selected depending on the current movement class and that are optimized views on the symbol distributions. Of course, if an a posteriori distribution with acceleration vector classes would have been used, this could not have been exceeded. The figure also shows the compression performances of the contextual distributions based on velocity categorizations and allows for a clear statement: the smaller the velocity intervals, the better the compression ratios that can be achieved. In the group of velocity-based contextual distributions, the distribution with $1\,{}^{m}/_{s}$ steps that we used for the clustering performs best, the distribution with $3\,{}^{m}/_{s}$ steps follows in the ranking. Both of these distribution models exceed the performance achieved with the post-factum distribution. Finally, there is hardly a compression ratio difference for the clustered distribution and the one that merely differentiates between movements and halts. For $\epsilon > 1.3$, these are only minimally better than the results of the trained distribution.

From these results, we learn how to setup the probability distributions for information determination and arithmetic coding: first, we see that assuming a normal



(a) Absolute results.



(b) Relative results.

Figure 4.17: Compression ratios for contextual probability distributions.

distribution of measurement innovations actually provides useful results, if no training set is given. Second, unless small alphabets are used, simple adaptive algorithms do not provide as much of a benefit for the compression ratio as well-trained static distributions. It simply requires very long trajectories to establish a representative distribution, and such lengths are rarely given. Third, contextual distribution models are the best means to be applied. In particular, trained distributions based on the acceleration vector of the moving vehicles provide the most promising results, and in case that velocity contexts should be employed, every aggregation of velocity classes significantly reduces the compression performance of the used coder.

4.7 Conclusion

In the previous chapter, we analyzed several geometric trajectory compression algorithms with respect to the compression ratios they can achieve. We found out that nonlinear compression models are able to achieve better compression ratios than state-of-the art approaches that use linear simplification. However, we could not make any statements on the quality of the regarded algorithms with respect to the best possible compression ratio any algorithm could possibly achieve.

In general, the upper bound for the compression ratio for some data is given by the *information* that is contained within the data; one can not reduce the data to less than the size of the optimal encoding of this information, without losing parts of the information. This concept of an information content was introduced in Claude Shannon's information theory in [Sha48] and is the foundation of the coding theory. We make use of this theoretical background to view the compression of trajectories from another perspective.

In this chapter, we focused on the determination of the actual information content of trajectories and the expected information content of object movements with respect to a prediction model. We introduced a formal model as a fundamental framework for measuring the information content of movement data and described the components that it consists of, namely the movement estimator, the symbol alphabet and the probability distribution. We pointed out the influences of the respective components on the whole system and used these findings to design practical implementations of the components based on this argumentation. In particular, we proposed three different movement estimators and discussed the setup of a discretization grid that is necessary to guarantee a finite and countable symbol alphabet. Finally, we took a close look at static, adaptive and contextual probability distributions that may be used to fit the actual symbol distribution of a regarded trajectory as good as possible. As a result, we assembled an implementation suite of the presented components that can be used in real-world setting for trajectory information content measurements.

Using these findings, we were able to specify an arithmetic coding and compression scheme. We demonstrated the practical applicability of our ideas by using them to compress vehicular trajectories and applied them to a large number of heterogeneous real-world vehicular movement traces. We performed an in-depth evaluation of the whole system and collected a number of interesting findings: first, the results from our evaluation show that our approach is superior to the best existing and state-of-the-art compression scheme for vehicular trajectories that were discussed in the previous chapter. Second, it could be seen that for movement estimation based on

position measurement traces, a simple linear movement extrapolation works best, because the movement parameters that are more detailed than the velocity vector suffer intensively from the original positioning noise. This result fits the findings from the clothoid sketching by means of curvature analysis that we discussed in Section 3.4. Third, a basic square-tessellated discretization grid yields the most stable compression results and, due to noisy position measurements, an adaption of the grid frame to a realistic shape is rather disadvantageous than helpful. Finally, we found out that though simple probability distributions such as the normal distribution result in very good compression ratios, trained distributions that regard a movement context, provide the best compression performances. In particular, distribution schemes regarding movement vectors worked best, even better than models working on velocity intervals.

With the findings from this chapter, we answer the question for an upper compression bound or at least provide an approximation for this bound. We realized that it is very hard to set up an accurate, absolute upper bound, because such would always depend on many factors, e. g., the positioning noise, the individual driving behavior in case of vehicular trajectories. Although, we found out that it can be described using prediction and probability models, instead, that need to be carefully adjusted to the use case.

5

Lossless Trajectory Compression

5.1 Introduction

The contributions on trajectory compressions that we have discussed in the previous chapters only regard lossy compression techniques with a configurable error threshold ϵ. While the geometric compression algorithms are able to compress the trajectories in a lossless manner, i. e., $\epsilon = 0.0$, this is not possible for the arithmetic coding, because this would imply an infinitely large symbol alphabet due to a discretization node distance of 0.0. However, we have also seen that for $\epsilon = 0.0$, the geometric algorithms do not perform very well, as the average compression ratios do not exceed 6 %.

An alternative to the presented algorithms are conventional compression techniques that have been proposed for text or byte string data. In general, those algorithms reduce the redundancy of the data by efficiently encoding repeating byte patterns in the input data. The trajectory data that we have regarded so far consists of geodetic coordination pairs that are encoded as floating point numbers, for instance following the IEEE 754 standard [iee08]. With such encodings, however, numbers with small differences are usually not represented by similar byte sequence, so that the direct use of conventional compression techniques is unlikely to be beneficial.

The next section contains an overview of previous work on lossless trajectory compression. Next to those contributions, conventional compression algorithms can also be considered as promising means to implement lossless trajectory encoding. Therefore, we then briefly present and discuss a selection of such algorithms and evaluate their compression performance, when applied to raw, i. e., non-preprocessed and binary trajectory data. However, the results from this evaluation are far from being satisfying.

For this reason, in the subsequent section, we will describe a way to prepare trajectories and encode them by following a trajectory byte encoding scheme, the output of which is better suitable to the conventional compression approaches. Subsequently, we evaluate the achieved compression performances for the output of our byte coder and use the real-world trajectory test set S_3^{test} for a direct comparability.

5.2 Related Work

In contrast to lossy trajectory compression, comparably little work has been published on lossless trajectory data compression. In [LHB06], the authors discuss the compression of ASCII-coded NMEA-0183 position measurement log files [Ass95] on the basis of separate records. In doing so, they distinguish between stationary and movement estimations: if the single elements of a sequence of position measurements do not differ significantly from a moving average value that is calculated over an initial set of positions, all following position measurements are discarded. If, on the other hand, the positions differ, the data sentences are encoded using the direct binary representations and compressed with conventional algorithms. However, since position measurements are actually dropped, this approach implements merely a *quasi-lossless* compression. Also, neither the binary encoding nor the used compression algorithms are explained, which makes the result hard to be compared to ours.

For high-precision and long term position measurements, the *Receiver Independent Exchange Format (RINEX)* [GE09] is often used as a logging format. With this format no actual positions, but raw GNSS signal information, i. e., information about the signal phase, the pseudorange[1] and the doppler shift, are encoded. Storing raw information enables researchers to derive the measurement device's position post-factum, using different techniques, for instance for evaluation purposes of new developed algorithms. Due to the immense volume of data, the *observational data files* containing the raw data easily become very large and need to be compressed. In [VMR+10], the authors present an approach to compress the raw RINEX data using different prediction models and the *FAPEC* entropy encoder [PV09] for the observables and a conventional dictionary compressor for the composition of observable byte code and the text-formatted headers. Although it appears meaningful to use a tailored preprocessor, the approach presented in this contribution is not applicable to the trajectory data we are handling.

Finally, in [RHS09], Reinhardt et al. propose a compression layer for wireless sensor network nodes that implements a stream encoding for network packets that contain

[1] A pseudorange is a systemic error for the distance estimation between a GNSS receiver and all regarded satellites due to an offset of the receiver's clock to the synchronized clocks of the satellites.

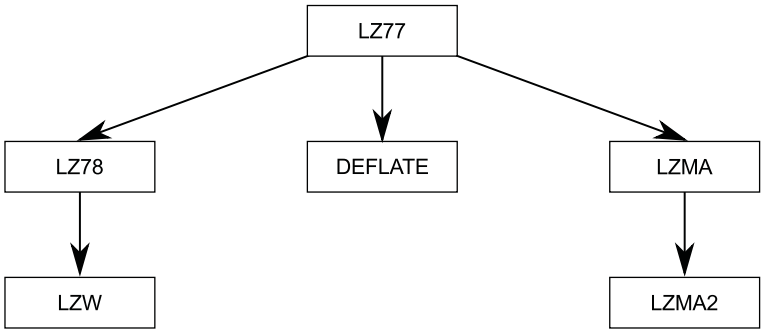


Figure 5.1: A selection of LZ family compression algorithms.

sensor measurements, such as timestamps, positions, and temperature. The authors assume that measurements may change slowly or repeatedly take similar values over time and thus packets may differ only marginally from previously occurred packets. Therefore, they calculate the difference between a packet and the most similar packet in a history table and encode this difference, assuming that the resulting byte stream that can be compressed easily due to a presumably large number of contained zero bytes. As a result, the authors find that significant compressions can be achieved even for simple difference functions being used. We adapt the idea of preprocessing the data to acquire a large number of equal bytes, but instead of working with simple difference functions, we propose a data-specific encoding that makes use of difference vectors of first and second degree to benefit from presumably slowly and continuously changing position measurements.

5.3 Conventional Lossless Compression Algorithms

5.3.1 LZ Algorithm Family

The *LZ* algorithm family compasses the group of compression algorithms that are derivatives of the lossless compression algorithm designed by Abraham Lempel and Jacob Ziv. All of these algorithms use some sort of dictionary and aim at compressing a symbol string by replacing repeating symbol sequences in the string with pointers to the dictionary entry that already holds the respective, previously inserted sequence. Because of this, algorithms from the LZ family are also said to implement *dictionary* or *substitution* compression. In the following, we will go through the LZ algorithm family in the order of derivation that is also visualized in Figure 5.1.

LZ 77

The *LZ77* algorithm, as published in [ZL77], implements the dictionary in the form of a *sliding window* using a fix-sized buffer. This buffer is split into a *prefix* part in the front and an *extension* part in the back. Initially, the symbol string is shifted into the extension part and from then on, the algorithm looks in each iteration for the longest string subsequence in the prefix part that matches the first symbols at the head of the extension part. Once such a subsequence with length $l_i - 1$ and starting at position p_i is found in the i-th iteration, the algorithm produces the compression output $(p_i - 1, l_1 - 1, c_i)$, where c_i is the first symbol in the extension part that could not be matched. The sliding window is then shifted by l_i symbols, i. e., the first l_i symbols from the prefix part are removed and l_i further symbols are shifted into the extension part, and the encoding starts anew. The compression ends, once the last symbol is shifted out of the extension part.

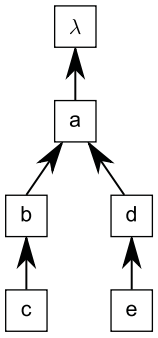
LZ 78

As an extension of LZ77, the *LZ78* [ZL78] algorithm uses a dedicated dictionary instead of the temporary memory based on the sliding window. Basically, the LZ78 algorithm also looks in each step for the longest dictionary entry that is a prefix of the current content of the input buffer. It then constructs a tuple (e_i, c_i), where e_i is a reference to the found dictionary entry and c_i is the first symbol that extends e_i in the buffer, as we know it from the LZ77 algorithm. The algorithm then appends this tuple to the encoding result and also inserts it into the dictionary as a new entry. This way, the dictionary that is initially empty, is filled on the fly during the encoding or decoding.

Technically, the LZ78 dictionary can be understood as a tree-like structure, where each entry is also referable to with an index: the first (root) entry $1 : (0, \lambda)$ contains an invalid reference and the empty symbol λ with length 0. Further entries then create reference chains, at the end of which the root entry is always referred to. The input symbol string “aababcadade”, for example, would be processed in the steps “a–ab–abc–ad–ade” and result in the dictionary shown in Figure 5.2.

LZW

In [Wel84], the author presents the *LZW* algorithm, an extension to LZ78, along with implementation details. As main difference, LZW works with a dictionary that is not initially empty, but is filled with all unigram symbol instances from the input alphabet. For a byte coder, this would mean that the dictionary would initially be filled with the values 0 to 255. This reduces the overhead in the initialization phase that would occur



(a) Tree structure.

index	entry
1	$(0, \lambda)$
2	$(1, a)$
3	$(2, b)$
4	$(3, c)$
5	$(2, d)$
6	$(5, e)$

(b) Table structure.

Figure 5.2: LZ78 dictionary example for the symbol string "aababcadade".

for the LZ78 algorithm. The LZW algorithm is implemented by the UNIX software *compress* in the *ncompress* project [Fry].

DEFLATE

The *DEFLATE* algorithm applies a Huffman encoding (cf. Subsection 4.3.2) to the results of a modified LZ77 encoding. In doing so, characters and symbol strings that may occur often in the LZ77 code are further compressed with a Huffman entropy encoder. The DEFLATE algorithm is implemented in the *gzip* compression program [GA, Deu96].

LZMA/LZMA2

Another algorithm that uses an entropy coder to further compress the output of a dictionary compression is *LZMA* [Pav]. Like the DEFLATE algorithm, it uses the LZ77 coder and further compresses the output of which using a range encoder (cf. Subsection 4.3.4). The range encoder employs an adaptive probability model for the input symbol alphabet that is controlled by an estimator, using a Markov chain.

The second version of LZMA, *LZMA2*, introduces the possibility to split the input sequence in several subsequences that are then encoded separately. While this allows for a parallel subsequence encoding to enhance the runtime performance, a segmentation of the input stream may also be beneficial for the compression ratio: LZMA2 supports segments of uncompressed data that would otherwise be inflated in size if being encoded with the original version of LZMA, due to the encoding overhead.

A more recent LZMA implementation is provided in the *xz utils* collection [Col]. In all of our tests, xz achieved significantly better compression results, although both

programs used the same amount of memory for the dictionary. The implementation difference is not clear to us, but we use the xz software for our evaluation.

5.3.2 bzip2

The *bzip2* compression program [Sewa] employs an algorithm that consists of nine steps—we will only regard the most important steps[2]: first, the code is sorted according to the *Burrows-Wheeler Transformation (BWT)* [BW94] that can be reverted for decoding without any overhead. The purpose of a BWT is to reorder the symbols in such a way that afterwards, there are multiple sequences of repetitive symbols in the string, which is beneficial for later compression steps. The third step already profits from this transformation, as the symbol string is now encoded with the *Move-To-Front (MTF)* algorithm [BSTW86]. In its original version, MTF reads a text and inserts new words at the end of a list, if they are not contained in the list already. The algorithm then encodes the words with the index of the list entry and moves the entry to the front, so that repeating words will have especially low indexes. Also, words are encoded only once in their original form, all following occurrences are encoded with integer list indexes. Instead of using words, the bzip2 algorithm applies MTF on the basis of symbols. If the assumption holds that there is a set—which of course may vary over time—of continuously reoccurring symbols, these will be encoded by a small set of indexes, with no regard of the actual symbol values. Obviously, this would decrease the entropy of the resulting code string. After another *Run Length Encoding (RLE)* phase, the code string is compressed using a Huffman code.

5.3.3 Arithmetic Coding with Prediction by Partial Matching (PPM)

As last compression scheme, we will use an arithmetic coder (cf. Subsection 4.3.3). For the probability model, we will use the *Prediction by Partial Matching (PPM)* model [CW84] that adapts with every read symbol from the input stream. The PPM model regards *conditional occurrence probabilities* of symbol n-grams (i. e., symbol chains of size n) and always tries to encode as long symbol sequences as possible. To this end, the PPM model is initialized with a parameter N (PPM-N) that determines the maximum value for n. The model then maintains N tables, one for each value of n, in which the n-gram frequencies are recorded; let us denote the particular table for n-grams as T_n. Then, if a symbol is read, the PPM model looks up the frequency f of the n-gram that consists of the last N read symbols (in order) in table T_N. If the

[2] We skip the actual first step, namely the preparation with a Run Length Encoding, because the author of bzip2 states on the project website that this first step "is entirely irrelevant" [Sewb], as the Burrows-Wheeler transformation can safely handle repetitions in the symbol string.

algorithm does not find a match in T_N, it repeats the search for the respective n-gram that consists of the last $N-1$ read symbols in T_{N-1}. This procedure is repeated until a frequency f can be retrieved. If even T_1 does not contain the unigram of the last read symbol, the frequency if set to $f = \frac{1}{\mathcal{A}_X}$, with $\mathcal{A}_X$ referring to the input symbol unigram alphabet. The tables are then updated and the symbol is passed to the arithmetic coder, along with the frequency f which is then interpreted as the symbol's probability of occurrence.

5.3.4 Compression Performance for Raw Trajectory Data

We apply the compression algorithms described in this section to raw, i. e., non-preprocessed, binary trajectory data to evaluate the basic compression performance that can be achieved without any prior data preparation. To this end, we encode the trajectories from the test set S_3^{test} that we already used in Chapter 4 by storing the first reference timestamp and the measurement interval once and the position measurements as a sequence of absolute value pairs. As floating point representations may be disadvantageous for the compression, we use two versions for this simple byte code, specifically a floating point and a fixed point raw trajectory format. For the floating point format, we use the IEEE 754 format [iee08] with double precision, so eight bytes are used for each position measurement coordinate. For the fixed point format, we use four or eight bytes per coordinate, depending on the number of digital places of the input data. In the following, we refer to the number of bytes used to encode a single spatial measurement pair as *field width*, or w_0.

The conventional compression algorithms are applied to these simple byte codes and the achieved compression ratios for each algorithm are analyzed. For the determination of the compression ratio, we need to regard the length of the decimal parts of the trajectory coordinates, because these directly influence the minimal necessary field width to store a coordinate (w_0). We refer to the maximum number of decimal places for the longitudinal and the lateral coordinates as $n_{\text{lon}}, n_{\text{lat}}$, respectively. Then, the optimal field width (in bits) for a trajectory element in an uncompressed reference binary file is determined by

$$w_0 = \lceil \log_2 360 \cdot 10^{n_{\text{lon}}} \rceil + \lceil \log_2 180 \cdot 10^{n_{\text{lat}}} \rceil \ . \tag{5.1}$$

Given w_{time} as the number of bytes necessary to encode the reference timestamp and the measurement frequency, the byte size s_{ref} of the reference file holding n_{pos} position

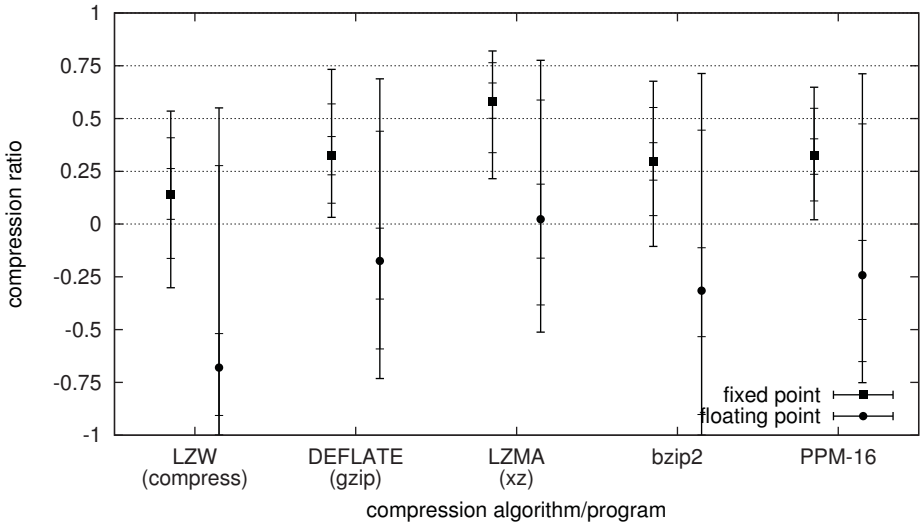


Figure 5.3: Compression ratios for all algorithms and raw data.

measurements can be calculated as

$$s_{\text{ref}} = w_{\text{time}} + \left\lceil \frac{n_{\text{pos}} \cdot w_0}{8} \right\rceil \ . \tag{5.2}$$

Finally, the compression ratio is defined as

$$\sigma = 1 - \frac{s_{\text{c}}}{s_{\text{ref}}} \ , \tag{5.3}$$

where s_{c} is the size of the coded or compressed file. Thus, a negative compression ratio indicates that the respective algorithm's output is larger than the input.

The achieved compression ratios are visualized in Figure 5.3: for each compression algorithm, the figure shows the results for the fixed and floating point format. Next to the minimum, average, and maximum, two thin horizontal lines each between the average and the minimum/maximum mark the 2-, 25-, 75-, and 98-percentile values, as we know it from the box plots in the previous chapters. As expected, the trajectory files in the floating point format could not be compressed as well as the trajectories in the fixed point format. In general, the compression results are unsatisfying, as only the LZMA algorithm achieved a compression ratio of more than 0.5, which is due to the LZMA2 extension, with which uncompressible data chunks can be stored with fewer overhead. Obviously, the conventional algorithms do not provide an appropriate compression performance for raw binary files; we therefore propose to preprocess the trajectory data to make it better compressible.

5.4 Lossless Byte Encoding

In the previous section, we discussed the functioning of conventional compression algorithms. The byte coder that we are presenting in the following aims at converting the input data, namely the spatio-temporal trajectories, into a data format that, applied to which, the discussed compression algorithms can achieve a good compression ratio. Although the positions in the trajectories may appear similar in their original decimal notation, these similarities do not necessarily occur in the respective byte stream representation as well. The information from the trajectories thus needs to be encoded in such a way that its redundancy is reflected by the bytes in the symbol stream. We therefore implement a *delta encoding* scheme, similar to the way that we have already designed the lossy arithmetic coding model in Chapter 4 and that extends the delta encoding proposed in [RHS09].

5.4.1 Algorithmic Idea

The delta encoded byte stream is very likely to allow for a good compression, because in general, position measurements for moving objects do not differ very much over time. Further, these differences are presumably steady by trend, as we still expect object movements to be smooth and regular to a certain degree. Therefore, we use difference vectors of first and second degree, in the following referred to as Δ_1 and Δ_2. Accordingly, we will denote absolute position data sets by Δ_0 vectors.

For the encoding of the trajectory elements, we use a fixed point arithmetic instead of a floating point arithmetic, for multiple reasons: first, with a floating point arithmetic, changes of small difference vectors could more easily affect multiple bytes in the resulting code stream than with a fixed point arithmetic. Second, possible rounding errors with the floating point arithmetic could result in an erroneous coding or decoding of the byte stream. Additionally, the encoder produces an *aligned* byte code, i. e., data are always encoded in full bytes, because all of the presented compression algorithms work on a byte basis. For the fixed-point arithmetic, we basically multiply each position coordinate with a factor 10^d, where d is the number of the decimal places that the encoder should regard. This means that for positioning devices that provide a higher coordinate resolution, the encoder needs to use more bytes per position encoding. The same applies for Δ_1 and Δ_2 vectors, of course: since with fixed point arithmetic, every number can be understood as an integer and as the difference between two positions most likely requires significantly less bits to be represented than the original data, it makes sense to reserve fewer bytes for difference vectors as well. Again, we refer to the number of bytes used to encode a Δ_n vector as the n-th degree *field width*, or w_n.

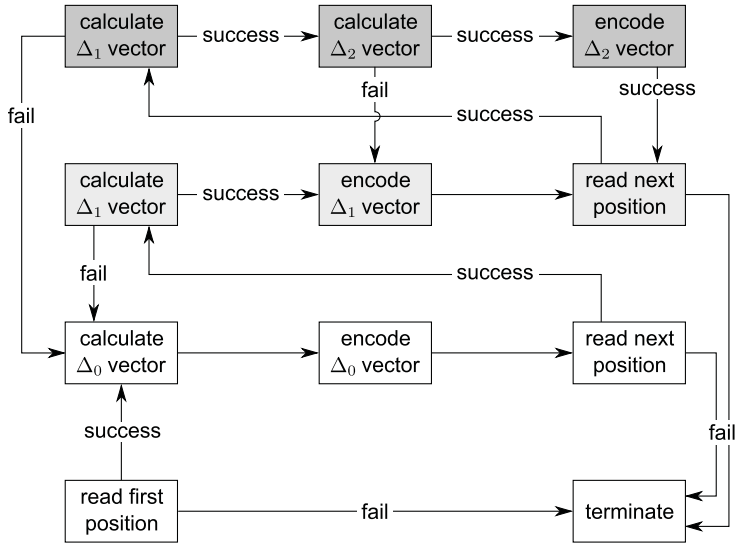


Figure 5.4: Algorithmic overview for the byte encoder.

Figure 5.4 visualizes the algorithm of the byte coder as a control flow diagram: the first position is encoded as Δ_0 vector and serves as reference point. From the second read position on, the encoder then attempts to calculate a Δ_1 vector as the difference of the last two calculated Δ_0 vectors. If successful, the encoder then tries from the following position on to calculate first a Δ_1 and then the respective Δ_2 vector. Whenever an attempt to calculate a Δ_n vector fails, the algorithm continues with the Δ_{n-1} vector. The calculation of a Δ_n vector fails, if the difference between two Δ_{n-1} vectors requires more bytes to be represented correctly than are reserved for the n-th degree. The algorithm terminates once no further position can be read. Since every position in the trajectory is only regarded once, the runtime complexity of the byte coder is $\mathcal{O}(n)$.

The choice of the *field widths* is not as trivial as it occurs. Of course, the simplest way for the encoding is to always use the maximum number of bytes that would be necessary to encode a Δ_0 vector. However, this would very probably cause an overweight of `0x00` or `0xFF` bytes in the code stream, which again would bias and distort the frequency distribution for the Huffman coding, for example. This could artificially increase the byte stream's entropy and lead to a worse compression ratio. Therefore, we discuss the choice of appropriate field widths in the following subsection.

[3] depends on number of decimal places of position measurements
[4] determined by cumulative distribution analysis

Profile	Information	Longitude	Latitude
1	$*$	8	8
2	Δ_0	$4/8^3$	$4/8^3$
	Δ_1	2/4	2/4
	Δ_2	1/2	1/2
3	Δ_0	$1/2/4/8^3$	$1/2/4/8^3$
	Δ_1	$1/2/4/8^4$	$1/2/4/8^4$
	Δ_2	$1/2/4/8^4$	$1/2/4/8^4$

Table 5.1: Field width profile overview.

5.4.2 Field Width Profiles

In this subsection, we present and discuss three field width profiles; Table 5.1 shows a summary of the particular field widths.

Profile 1

The first profile implements the already mentioned, very simple field width policy: the difference vectors for all degrees are encoded with an 8 byte fixed point arithmetic. In doing so, a position's latitude and longitude coordinates can be encoded with up to 17 and 16 decimal places, respectively; such a high positional resolution is unrealistic, as it corresponds to a global maximal positioning error of approximately $1 \cdot 10^{-9}$ mm; in contrast to that, recent DGPS accuracy measurements and improvement estimations work in the magnitude of centimeters or millimeters, at the best [MMH05]. With this profile, every difference vector will fit into the respective fields, with no occurring overflows. Therefore, every trajectory will be encoded with one Δ_0 vector for the first position, one Δ_1 vector for the second positions and with Δ_2 vectors for the remaining trajectory elements.

Profile 2

For the second profile, the encoder checks the trajectory for the positioning resolution: if eight or more decimal places need to be supported, the field width is set to $w_0 = 8$, otherwise $w_0 = 4$. Following the assumption that the value domain of each difference degree decreases significantly, we halve the field widths with respect to the previous degree. Formally, $n = 1, 2 : w_n = \frac{w_{n-1}}{2}$.

Profile 3

The third profile determines the field widths on an empirical basis: for the Δ_0 vectors, the encoder scans the trajectory elements and chooses the best field widths for the longitude and latitude coordinates. For the determination of the field widths w_1 and w_2, all difference vectors of first and second degree are calculated and analyzed by means of their cumulative distribution functions. Then, according to threshold parameters $p_1, p_2 \in \mathbb{R},\ 0 < p_1, p_2 \leq 1$, the field widths w_1, w_2 are set so that the ratio of difference vectors of the particular degree, that are codable with these settings, does not exceed the respective threshold value. For example, if $p_2 = 0.6$, then w_2 is set to the smallest number of bytes that are necessary to encode 60 % of the observed Δ_2 vectors.

Obviously, this profile can not be applied in real-world scenarios, but merely serves as reference for the individually best-fitting profile. We assume that it will provide the best compression ratios and we will therefore use it as an upper performance bound. So, in our evaluation, we can compare the compression ratios for the other profiles with this.

Profile 4

With the results from the analyses performed with profile 3, we derive the fourth profile: we analyze the distribution of the field widths used for the third profile and set w_1, w_2 for the fourth profile to the field widths that appear most promising for the approximation of the compression ratios achieved with the third profile. The fourth profile can therefore be understood as a trained profile that should fit the majority of all trajectories.

5.4.3 Byte Code Structure

To interpret the byte code correctly, the decoder first reads the coder configuration that is transmitted in the byte stream header. This contains the number of decimal places for the longitudinal and lateral coordinate components, the reference time stamp, the measurement frequency, and finally the coding profile. If profile 3 is employed, the header also contains a 2-byte field that holds the field widths for all difference vector stages. Table 5.2 gives a summary over the header fields and the respectively used byte counts.

The structure of the remaining byte code is straightforward, as it merely contains a sequence of Δ_n vectors. We remember that the encoding algorithm follows a greedy philosophy in always attempting to increase the delta degree once the encoding of a difference vector of the current degree has been successful. Therefore, the decoder

header field	field width in bytes
decimal places for longitudinal coordinate	1
decimal places for longitudinal coordinate	1
reference time stamp	8
measurement frequency	1
coding profile	1
field widths (profile 3 only)	2
Total	**14**

Table 5.2: Byte code structure: header information.

also knows that it needs to interpret the next bytes in the byte code from the current position on as a difference vector of the degree that would be used if the accordingly attempted delta encoding step had been successful. However, there needs to be a way to signal the decoder that the calculation of a difference vector has actually failed. Therefore, whenever a range overflow occurs and so the intended difference vector could not be coded, the encoder inserts a control byte into the byte code stream. This byte can hold three different values to determine the degree of the next difference vector.

Using control bytes and thus implementing an irregular and semantic-dependent byte code structure has a positive effect on the entropy of the code, because in the other case, each difference vector would be announced by such a control byte. Especially in cases when an input trajectory is encoded with a majority of difference vectors of a particular degree, such regularly occurring control bytes would tamper the resulting code's entropy, thus causing a worse compression result. The dynamic approach that we suggest inserts the control bytes only when necessary and therefore minimizes the control byte overhead. Even in cases, when every difference vector encoding attempt fails in profiles 2 and 3, the dynamic approach causes one byte less overhead than the static approach, because the first difference vector degree is always 0 and therefore does not need to be announced.

The dynamic approach makes the encoding and decoding steps a little more complex, though: Since the decoder needs to identify the control bytes with certainty, the encoder has to make sure that no ambiguities occur in the byte code. Therefore, in case that no control byte needs to inserted that would clarify the semantic of the following bytes, the encoder needs to analyze the first byte of the each delta vector that is to be encoded. If this first byte equals one of the three reserved control byte values, it inserts a control byte for the current difference degree before appending the

Algorithm	Linux application	Version	Parameters
LZW	`compress` [Fry]	4.2.4.3	
DEFLATE	`gzip` [GA, Deu96]	1.3.12	`-9`
LZMA	`xz` [Col]	5.0.0	`-9`
BWT+MTF+RLE+Huffman	`bzip2` [Sewa]	1.0.5	`-9`
arithmetic coding w/ PPM-N	`arithcode` [Car02]	1.1	`16`

Table 5.3: Lossless compression programs and configurations used for the evaluation.

difference vector. In doing so, the encoder uses the control byte as an *escape sequence* to resolve ambiguities. To reduce the necessity of such situations, we need to choose the reserved control bytes carefully. For instance, small values would be disadvantageous, as we expect many small difference vectors to occur as well. We therefore use the three largest positive numbers as control byte values.

5.5 Evaluation

5.5.1 Methodology

In this section, we evaluate our proposed byte coding algorithm and the profiles that we have implemented. Again, we encode the trajectories from the test set S_3^{test} for this purpose. The byte codes are then compressed with the conventional compression algorithms that we discussed in Section 5.3. We evaluate both the sizes of the compressed byte codes and the steps from the byte encoding process to gain a thorough understanding and use the overall compression ratio as a general quality measure. We use the definition of the compression ratio from Statements (5.1), (5.2), and (5.3).

Table 5.3 shows the lossless compression programs and their configurations, as they are used for this evaluation. We configured the programs *gzip*, *xz*, and *bzip2* with the `-9` parameter to use the largest-possible amount of memory that the implementations provide for the compression process. For the PPM encoding, we set $N = 16$.

5.5.2 Determination of Profile 3 Parameters

First, we turn towards the third profile that we proposed as an upper performance bound estimate. In this profile, the field widths w_1, w_2 for the difference vectors of first and second degree, respectively, are chosen in such a way that a ratio of p_n of all Δ_n vectors, $n = 1, 2$ can be stored. We determined the compression rates for all combinations of $0 < p_1, p_2 \leq 1.0$ in steps of 0.1, resulting in 100 compression perfor-

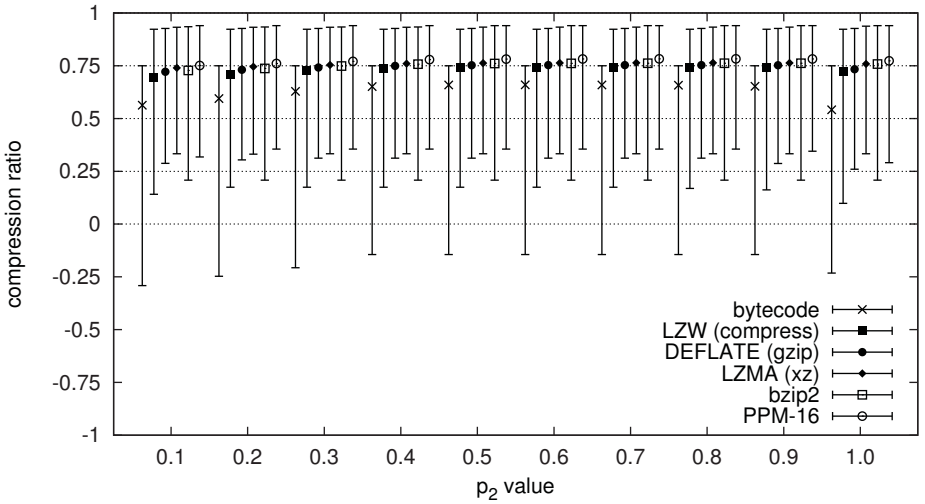
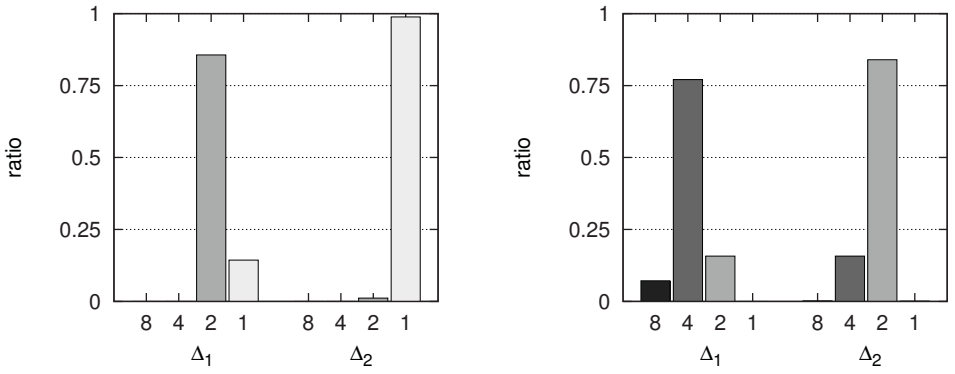


Figure 5.5: Compression ratios for profile 3 with $p_1 = 1.0$ and over varying p_2.

mances for each used program. Without showing all of these results, we discovered that $p_1 = 1.0$ led to the best results, i. e., setting w_1 in such a way that all difference vectors of the first degree can be encoded and only the first position measurement needs to be stored as absolute value.

The determination of the optimal value for p_2 is more complex. Figure 5.5 shows the compression performances of the selected programs for $p_1 = 1.0$ and the complete range of p_2. For each value of p_2, the compression result of a particular program is shown as an average with vertical range bars. At each bar, a particularly shaped point marks the average, and the topmost and lowermost limitations mark the best and worst achieved compression results, respectively. At first glance, the ranking of the compression results within each p_2 value group can be seen: beginning with the delta byte encoder that provides an average compression ratio between 0.54 and 0.66, the LZW, DEFLATE and LZMA algorithms achieve better results, in this order. bzip2 performs slightly worse than LZMA and finally, the arithmetic coding with the PPM-16 model always achieves the best compression.

The second effect is that the compression performances vary over the range of p_2. Instead of continuously increasing, we see that at some point, they reach an optimum and decrease from there on. The byte coder reaches this maximum at $p_2 = 0.6$, LZW, DEFLATE, LZMA, and the arithmetic coder perform best for $p_2 = 0.7$, and bzip2 has the best compression for $p_2 = 0.8$. As the maximum compression ratios of the byte coder and bzip2 differ only marginally from their performances with $p = 0.7$, we select $p_1 = 1.0, p_2 = 0.7$ as the parameter set for the third profile of our byte coder.



(a) $w_0 = 4$ (b) $w_0 = 8$

Figure 5.6: Overview of difference vector degrees used with profile 3.

5.5.3 Determination of Profile 4 Parameters

Based on the results achieved with profile 3, we can now derive the parameters for profile 4. As previously stated, we therefore regard the field widths that were used in the byte coder runs with profile 3. Figure 5.6 shows the usage ratios of the available field widths for the difference vectors and for $w_0 = 4$ (cf. Figure 5.6a) and $w_0 = 8$ (cf. Figure 5.6b). We see that for both cases, the majority of field widths are cut in half with each increasing step of the difference vector degree. This is exactly what we modeled with the field width settings for profile 2, so we can expect that profile 2 already provides a very good approximation of the compression performance achieved by profile 3. However, for $w_0 = 4$, approximately 15.75 % of all trajectories required higher field widths for the Δ_2 vectors. In these cases, profile 2 would need to step down to the respectively lower difference degree, causing a higher data entropy and a worse compression ratio. For this reason, we choose to set $w_2 = w_1 = \frac{w_0}{2}$. w_0 is always set according to the coordinates' number of decimal places in a trajectory. We are aware that, in the case $w_0 = 8$, approximately 7.18 % of the trajectories require field widths of 8 instead of 4 byte for the Δ_1 vectors, but we consciously do not increase w_1 even more, because this would over inflate the byte code with zero bytes which would cause a heavy shift in the symbol frequencies.

5.5.4 Profile Performance Comparison

Figure 5.7 shows the compression ratios of all implemented field width profiles, grouped by the used compression method. The figure shows the distribution details with the horizontal marks at the maximum, minimum, and the 2-, 25-, 75-, and 98-percentile

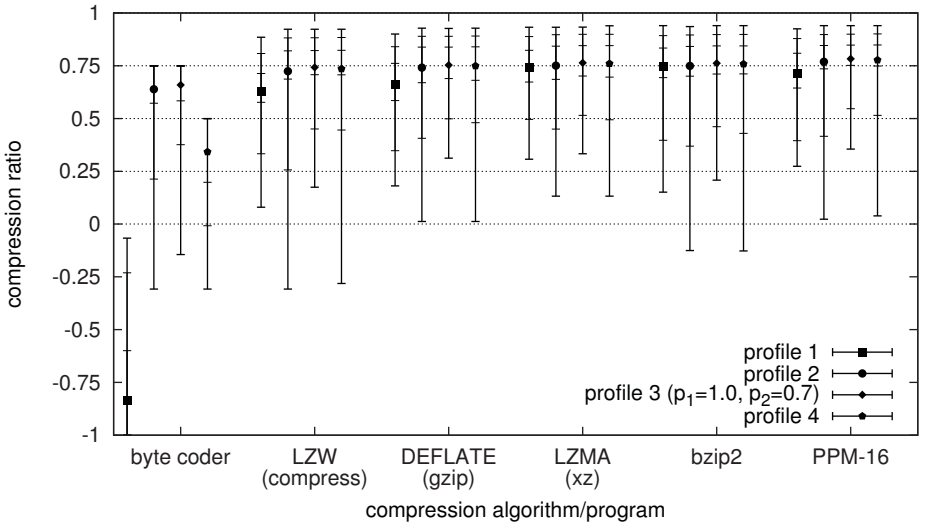


Figure 5.7: Compression ratios for all profiles and algorithms.

values, so it can be read like Figure 5.3. Overall, the compression with the arithmetic coder featuring the PPM-16 probability model performs best, while the xz utils and bzip2 provide a very close compression ratio. For every compression method, the average compression performance increases from the first to the third profile, while the results for the fourth profile always lie between the ones from the second and third. Actually, the difference of the fourth to the third profile is almost negligible for the maximum down to the 25-percentile mark. The 2-percentile marks show recognizable differences and for the fourth profile (as for the second), there are clearly worse minimum outliers, though. The circumstance that these outliers could not be eliminated is due to the relatively large amount of Δ_1 vectors for $w_0 = 8$, that we saw in Figure 5.6. This can also be seen in Figure 5.8 that shows the ratios of difference vectors over all encoded trajectories as box plots. We can see for profile 4, for example, that for 98 % of all trajectories, less than 18 % of the respectively contained positions needed to be encoded as Δ_0 vectors. It can also be seen that Δ_1 vectors are rarely used for the fourth profile, but Δ_2 vectors dominates, instead.

To put the lossless compression results into a global context, we want to point out that for the fourth profile, the arithmetic coder with the PPM-16 model achieves an average compression ratio of approximately 0.7770. This result is very close to the compression ratios achieved with the lossy arithmetic coder presented in the previous chapter, for $\epsilon = 0.05\,\mathrm{m}$. Actually, the lossless coder outperforms the lossy version with a uniform distribution (0.7305) and a Gaussian distribution (0.7717), but lies below the lowest results of the trained distributions (0.7891). Furthermore, it achieves

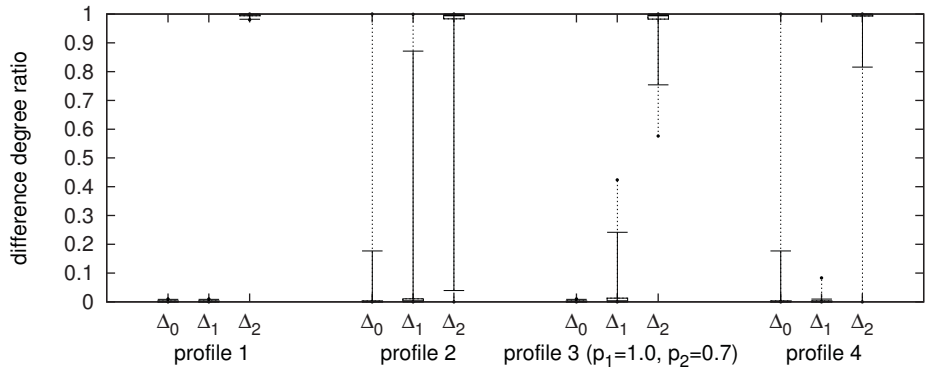


Figure 5.8: Comparison of usage ratio of the difference vector degrees for all profiles.

compression ratios that roughly match those obtained with the spline-based and optimal line simplification algorithms for error thresholds of $\epsilon = 0.65\,\mathrm{m}$ and $\epsilon = 1.00\,\mathrm{m}$, respectively.

5.6 Conclusion

While our previous work deals with compression schemes with adjustable accuracy, we presented a lossless encoding scheme for vehicular trajectories in this chapter. For this scheme, we discussed the suitability of existing, conventional compression algorithms, namely the *LZ77* algorithm and its derivatives, the *bzip2* algorithm, as well as the Huffman and arithmetic entropy encoders. However, these algorithms do not perform well when applied to raw binary trajectory descriptions that contain a mere list of absolute spatio-temporal measurements, because similar measurements do not necessarily result in similar byte representations, especially when they are stored in a floating point format.

We therefore proposed a byte coder that is applied to the trajectory data as a preprocessor. For this byte coder, we use the experience that we gained with the arithmetic coder from the previous chapter: instead of encoding absolute positions, merely the differences of the current positions from the expected values were regarded. For smooth and regular movements, such differences should be small. For the lossless trajectory compression, we realized this approach in the form of a delta encoder that aims at outputting only difference vectors of up to second degree. In doing so, it needs to regard the *field widths* for the difference vectors, i. e., the number of bytes that are available for the encoding of difference vectors of a certain degree. The field widths for higher-degree difference vectors should be smaller to avoid an overweight of unnecessary

leading `0x00` or `0xFF` bytes in the resulting byte code for positive and negative vectors, respectively. Such bytes would cause a bias in the byte occurrence frequency and thus distort determined probability distributions. In such cases, the resulting compression is most likely significantly suboptimal. To examine the influence of different field widths on the compressibility of the byte code, we specified four profiles: the first uses the maximal possible amount of bytes for the fields, the second one halves the field widths with each increment of the difference vector degree, the third analyzes the requirement of a trajectory to dimension the field widths to the optimal values accordingly, and the fourth approximates the performance of the third profile. The output of the byte encoder can directly be fed into conventional compression algorithms.

We evaluated the developed byte coder and the proposed profiles on the basis of the trajectory test set that we already used for the evaluation of the arithmetic coder in Chapter 4. We first found the best set of parameters for the third profile that we then used as an upper bound benchmark for the rest of the evaluation. In the next step, we examined the field widths of the third profile in detail and saw that the design of the second profile almost features the main characteristics of the third profile, promising very good compression results. To improve those even further, we set up the fourth profile with an especially high ratio of second-order difference vectors. In our last step of the evaluation, we compared the achieved compression ratios for the conventional compression programs and algorithms that we had applied to the byte coder's output. We saw that though all algorithms had comparably good performances, the best result was achieved with the fourth byte coder profile and subsequent arithmetic coding with a PPM probability model. The average overall compression ratios that we achieved with the lossless algorithms are close to the ones of the lossy arithmetic coders with trained probability distributions at an approximation error tolerance $\epsilon = 0.05\,\mathrm{m}$ that we have presented in Chapter 4. Also, they roughly match the compression ratios achieved with the spline-based and optimal line simplification algorithms for error thresholds of $\epsilon = 0.65\,\mathrm{m}$ and $\epsilon = 1.00\,\mathrm{m}$, respectively.

The byte coding scheme presented in this chapter is tailored to the needs of the employed conventional compression algorithms. For future work, a coding scheme could be regarded that can be directly integrated into an arithmetic coder, because the arithmetic entropy encoding provides the best results for the lossless and lossy trajectory encoding and is therefore the most promising approach to follow.

6

Conclusions

In this thesis, we focused on the question of how to compress spatio-temporal trajectories in such a way that a maximum compression ratio can be achieved. For various applications or use cases in ubiquitous computing or, for instance, inter-vehicular communications, spatio-temporal trajectories, i. e., sequences of time and position measurements, need to be reported by mobile nodes to a central unit. However, as many applications require a high temporal and spatial measurement resolution, the sizes of uncompressed trajectory report messages easily become very large. Thus, valuable wireless channel resources are wasted or costs may incur, should the transmission take place via cellular mobile communication. To reduce the data load that occurs during transmission, several lossy compression algorithms for trajectories have previously been published. In this thesis, we reviewed and evaluated existing compression algorithms and developed new ones on the basis of a large number of real-world trajectories that we retrieved from the OpenStreetMap project.

As a first step, we reviewed the related work on previously published compression mechanisms for spatio-temporal trajectories in Chapter 3. The majority of these contributions originate from the area of *mobile object databases (MODs)* and employ line simplification algorithms and other geometric and linear models. We evaluated two representative algorithms, specifically the Douglas-Peucker algorithm and an algorithm that calculates the optimal line simplification for a trajectory. We saw that the heuristic Douglas-Peucker algorithm performed 10-15 % worse than the optimum, while having a better runtime complexity. Both algorithms are widely used in the related work and form a de-facto standard; furthermore, the results of the optimal line simplification algorithm denote an upper bound for all linear geometric compression

approaches. For these reasons, we used these compression results as benchmark values throughout the thesis. Although the compression results obtained with the linear algorithms were basically satisfying, we argued that vehicular movements are naturally not necessarily linear, but rather obey the principles of kinematics. In kinematics, object movements are described with smooth curves, so we assumed that, in analogy to that, geometric compression algorithms should be designed with nonlinear models as well. Additionally, we argued that once the underlying geometrical design patterns and components of roadways were known, trajectories could basically be compressed by simply approximating concatenations of these roadway components. In fact, so called clothoid splines, i. e., smooth combinations of line segments, circular arcs and clothoids (transition curves), are used to construct roadways, especially when a joltless movement is desired, such as on highway exits. We therefore attempted to sketch our collected trajectories with clothoid splines and used an algorithm from computer graphics for this purpose. However, even the best currently existing algorithm is not designed to strictly stick to an approximation error bound as we use it, but the accuracy is only adjustable via a penalty cost model. Due to this cost model, the algorithm can not guarantee that the demanded error threshold is not violated and thus achieved worse compression results than even the Douglas-Peucker linear algorithm. Finally, we presented a trajectory compression algorithm based on cubic spline interpolation. Cubic splines have a number of advantageous features that suit our use case: first, the cubic polynomials meet at the supporting knots under the C^2 criterion that ensures special smoothness. Second, the cubic polynomials allow for a good approximation of the steering behavior that is usually modeled using the second derivative, and finally, the complete spline has a minimal oscillation behavior and curvature between the knots, which resembles a smooth and regular movement. Our evaluation strengthens our assumption that vehicular trajectories can indeed be approximated more accurately with nonlinear functions than with linear models: for $\epsilon < 0.1\,\mathrm{m}$, our spline approach performs more than 10 % better than the linear benchmark algorithm; this compression gain, however, comes at the cost of a higher algorithmic complexity of $\mathcal{O}(n^3)$.

The compression results that we achieved with the spline interpolation based approach showed that geometric linear simplification and modeling that is used in literature as de-facto standard compression technology for trajectories, can significantly be outperformed once nonlinear models are employed. However, the results did not allow for the estimation of an upper bound for the compressibility ratio of a trajectory, because first, we are aware that the algorithm merely finds a locally optimal solution, and second, we can not prove that the approach of using cubic splines is optimal itself. Therefore, we approached the trajectory compression from an information-theoretic

point of view in Chapter 3: we gave an introduction into the Shannon information theory and proposed a generic method of measuring the information content of a spatio-temporal trajectory. This method uses a movement estimator to predict the next position measurement and derives the information content of a single measurement from the deviation to the respective estimation. To this end, the measurement is mapped to a symbol alphabet for the symbols of which a probability distribution needs to be given. We then returned to the use case of vehicular movements and presented a complete and detailed implementation example for the movement estimator, the discretization means that we use to map the innovation to the symbol alphabet, and the probability distribution. We proposed several alternatives for each model component, implemented these within an arithmetic coder and evaluated their influences on the compression performance; in doing so, we found out that simple component realizations, e. g., a movement estimator that does not regard the calculated acceleration and a simple square-tiled discretization grid, often provide the best compression ratios and that for the use in practice, trained probability distributions should be used that depend on the respective vehicle's last detected heading angle. While the choice of model parameters can improve the compression performance, even the simplest model configuration enables the arithmetic coder to clearly outperform the geometric compression algorithms discussed in Chapter 3.

While only lossy compression schemes were regarded up to this point, we examined the performances of a selection of conventional lossless algorithms as the final contribution of this thesis. For this selection, the compression algorithms from the LZ family, the *bzip2* algorithm, and the Huffman and arithmetic entropy encoders were regarded. All these algorithms work on byte streams and achieve good compression results if the input contains numerous equal symbols or symbol strings. For this reason, they would not perform well when directly applied to trajectories that were encoded in the traditional binary formats, such as the IEEE 754 floating point representation [iee08], because even if two decimal numbers are within close proximity in the real domain, for example, their binary representations are not necessarily as similar to the same degree. Therefore, we proposed a byte encoding scheme that preprocesses the trajectory data; thereby, it preserves the contained information, but changes the byte content so that afterwards, there is presumably a large amount of equal bytes in the code. To achieve this, it performs a delta encoding: if possible, the encoder does not store the absolute trajectory element, but only a difference vector of first or even second degree. In this context, the preprocessor encodes the difference vectors of each stage with a particular number of bytes: the higher the difference degree, the fewer bytes are presumably necessary and therefore the fewer bytes are used. This is done to reduce the

number of unnecessary `0x00` or `0xFF` bytes that would distort the byte distribution. Therefore, the preprocessor can only encode a trajectory element as difference vector, if the value of the vector can be encoded with the respective number of available bytes. We applied the conventional compression algorithms to the output of the byte encoder and found out that all algorithms achieved comparable results, where the arithmetic coding with a PPM adaptive probability model performed best. Compared to the lossy algorithms that we discussed in the previous chapter, we found out that the lossless compression achieves nearly as good a result as the lossy arithmetic coder at the lower error tolerance, the spline interpolation based geometric approach at an error threshold $\epsilon = 0.65\,\mathrm{m}$ or line simplification algorithms at $\epsilon = 1.00\,\mathrm{m}$.

From the results collected in this thesis, we can deduce some clear statements: if geometric methods are to be used, it makes sense to use nonlinear methods for movement modeling; these provide a better approximation for vehicular movements, thus allowing for higher compression ratios, as we have seen at the spline interpolation results in Chapter 3. Conjunctively, we only proposed a method with a significantly higher runtime complexity than its linear counterparts, so that it can merely be applied on buffered trajectories with less measurements than those being handled by linear techniques. Furthermore, information-theoretic approaches, i. e., entropy coding, provided the best compression results that we could achieve in our evaluations. This came at the cost of a more complex model, the components in which need to be tailored to the particular use case. Also, though our model has a linear asymptotic runtime complexity over the number of trajectory elements, it is slower than, for example, the heuristic Douglas-Peucker algorithm, due to the employed movement estimation and discretization steps. This would result in a higher power consumption for a mobile device's computing unit and requires further research to evaluate the actual impact in real-world environments. It is therefore necessary to optimize the algorithm for the particular use case to reduce the actual power consumption. Finally, even lossless compression techniques can be successfully applied, if the trajectory data is preprocessed, for instance with a delta encoding scheme.

By using arithmetic coding, we could achieve compression results that are optimal for the respective choice of model parameters in Chapter 4. Though this provides only a partial answer to our initial question of how spatio-temporal trajectories can be compressed so that a maximum compression ratio can be achieved, our findings have a strong practical relevance and can be used directly to significantly reduce the channel load for the wireless transmission of trajectories. By means of compression, movement report protocols can be optimized to cut a long movement story short and to transmit only the essence of it.

Bibliography

Own Publications

[KBMS11] Markus Koegel, Daniel Baselt, Martin Mauve, and Björn Scheuermann. A Comparison of Vehicular Trajectory Encoding Techniques. In *MedHocNet '11: Proceedings of the 10th Annual Mediterranean Ad Hoc Networking Workshop*, June 2011.

[KKKM10] Markus Koegel, Wolfgang Kiess, Markus Kerper, and Martin Mauve. Compact Vehicular Trajectory Encoding (extended version). Technical Report TR-2010-002, Computer Science Department, Heinrich Heine University, Düsseldorf, Germany, September 2010.

[KKKM11] Markus Koegel, Wolfgang Kiess, Markus Kerper, and Martin Mauve. Compact Vehicular Trajectory Encoding. In *VTC '11-Spring: Proceedings of the 73rd IEEE Vehicular Technology Conference*, May 2011.

[KM11] Markus Koegel and Martin Mauve. On the Spatio-Temporal Information Content and Arithmetic Coding of Discrete Trajectories. In *MobiQuitous '11: Proceedings of the 8th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking & Services*, December 2011.

[KOKM10] Markus Koegel, Thomas Ogilvie, Wolfgang Kiess, and Martin Mauve. Real-World Evaluation of C2X-Road Side Warning Devices. In *ISWPC '10: IEEE International Symposium on Wireless Pervasive Computing*, pages 180–185, May 2010.

[KRHM12] Markus Koegel, Matthias Radig, Erzen Hyko, and Martin Mauve. A Detailed View on the Spatio-Temporal Information Content and the

Arithmetic Coding of Discrete Trajectories. *Mobile Networks and Applications*, pages 1–16, October 2012.

[RSKM09] Jedrzej Rybicki, Björn Scheuermann, Markus Koegel, and Martin Mauve. PeerTIS - A Peer-to-Peer Traffic Information System. In *VANET '09: Proceedings of the 6th ACM International Workshop on VehiculAr Inter-NETworking*, pages 23–32, September 2009.

Other References

[Abr60] Norman Abramson. *Information Theory and Coding.* McGraw-Hill Book Co., 1960.

[Ass95] National Marine Electronics Association. *NMEA 0183, Standard for Interfacing Marine Electronic Devices: version 2.1.* NMEA National Office, October 1995.

[Baa84] K. G. Baass. The use of clothoid templates in highway design. *Transportation Forum*, 1:47–52, 1984.

[Bar] Ilya Baran. Cornucopia: a clothiod sketching software. Online resource. http://code.google.com/p/cornucopia-lib.

[BD84] Brian A. Barsky and Anthony D. DeRose. Geometric Continuity of Parametric Curves. Technical Report UCB/CSD-84-205, EECS Department, University of California, Berkeley, October 1984.

[BD99] Amiya Bhattacharya and Sajal K. Das. LeZi-update: an information-theoretic approach to track mobile users in PCS networks. In *MobiCom '99: Proceedings of the 5th Annual ACM/IEEE Int'l Conf. on Mobile Computing and Networking*, July 1999.

[BEJS05] Rene Brüntrup, Stefan Edelkamp, Shahid Jabbar, and Björn Scholz. Incremental map generation with gps traces. In *ITSC '05: Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems*, September 2005.

[BLP10] Ilya Baran, Jaakko Lehtinen, and Jovan Popovic. Sketching Clothoid Splines Using Shortest Paths. *Computer Graphics Forum*, pages 655–664, 2010.

[BLvO12] Filip Biljecki, Hugo Ledoux, and Peter van Oosterom. Transportation mode-based segmentation and classification of movement trajectories. *International Journal of Geographical Information Science*, pages 1–23, October 2012.

[Bro98] Eli Brookner. *Tracking and Kalman Filtering Made Easy.* Wiley-Interscience, April 1998.

[BSTW86] Jon Louis Bentley, Daniel Dominic Sleator, Robert Endre Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4):320–330, 1986.

[BW94] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Systems Research Center, 1994.

[Car02] Bob Carpenter. Arithcode project: Compression via arithmetic coding in java, version 1.1, 2002. Online resource: http://www.colloquial.com/ArithmeticCoding/.

[CC92] W.S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. *Proceedings of the 3rd International Symposium on Algorithms and Computation, in: Lecture Notes in Computer Science*, 650:378–387, 1992.

[CE08] Ji-Wung Choi and Gabriel Hugh Elkaim. Bézier curves for trajectory guidance. In *WCECS '08: Proceedings of the World Congress on Engineering and Computer Science*, pages 625–630, October 2008.

[CJP05] Alminas Civilis, Christian S. Jensen, and Stardas Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):698–712, May 2005.

[CKK96] A.R. Cassandra, L.P. Kaelbling, and J.A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *IROS '96: Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, November 1996.

[CLFG+03] José Antonio Cotelo Lema, Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. Algorithms for moving objects databases. *The Computer Journal*, 46(6):680–712, 2003.

[CN04] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD '04: Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 2004.

[Col] Lasse Collin. XZ Utils. Online resource. http://tukaani.org/xz.

[CW84] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[CWT06] Hu Cao, Ouri Wolfson, and Goce Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB Journal*, 15(3):211–228, September 2006.

[DAC+08] N. D'Agostino, A. Avallone, D. Cheloni, E. D'Anastasio, S. Mantenuto, and G. Selvaggi. Active tectonics of the adriatic region from gps and earthquake slip vectors. *Journal of Geophysical Research (Solid Earth)*, 113(B12), December 2008.

[Deu96] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), May 1996.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[DMF+06] Chris Dever, Bernard Mettler, Eric Feron, Jovan Popović, and Marc Mcconley. Nonlinear trajectory generation for autonomous vehicles via parameterized maneuver classes. *Journal of Guidance, Control and Dynamics*, 29:289–302, 2006.

[DP73] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[Fox98] Dieter Fox. *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. PhD thesis, University of Bonn, Germany, 1998.

[Fry] Mike Frysinger. ncompress: a public domain project. Online resource. http://ncompress.sourceforge.net.

[FT84] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:338–346, 1984.

[GA] Jean-Loup Gailly and Mark Adler. The gzip home page. Online resource. http://www.gzip.org.

[Gal78] Robert G. Gallager. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, November 1978.

[GE09] Werner Gurtner and Lou Estey. Rinex – the receiver independent exchange format, version 3.01. Online resource, June 2009. http://igscb.jpl.nasa.gov/igscb/data/format/rinex301.pdf (as of 2012-10-15).

[Gil92] Thomas D. Gillespie. *Fundamentals of Vehicle Dynamics*. SAE International, March 1992.

[GKM+07] Joachim Gudmundsson, Jyrki Katajainen, Damian Merrick, Cahya Ong, and Thomas Wolle. Compressing spatio-temporal trajectories. In *ISAAC '07: Proceedings of the 18th International Symposium on Algorithms and Computation*, pages 763–775, December 2007.

[gpl] GNU General Public License, version 3. http://www.gnu.org/licenses/gpl-3.0.html.

[GS05] Ralf Hartmut Güting and Markus Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[GSTW04] Astrid Gühnemann, Ralf-Peter Schäfer, Kai-Uwe Thiessenhusen, and Peter Wagner. Monitoring Traffic and Emissions by Floating Car Data. Technical Report ITS-WP-04-07, DLR – German Aerospace Centre, Institute of Transport Research, Berlin, 2004.

[HGNMs08] Nicola Hönle, Matthias Großmann, Daniela Nicklas, and Bernhard Mitschang. Preprocessing position data of mobile objects. In *MDM '08: Proceedings of the 9th IEEE International Conference on Mobile Data Management*, April 2008.

[HGRM10] Nicola Hönle, Matthias Großmann, Steffen Reimann, and Bernhard Mitschang. Usability analysis of compression algorithms for position data streams. In *GIS '10: Proceedings of the 18th ACM SIGSPATIAL*

international conference on Advances in Geographic Information Systems, November 2010.

[HLO99] Werner Huber, Michael Lädke, and Rainer Ogger. Extended floating-car data for the acquisition of traffic information. In *ITSWC '99: Proceedings of the 6th World Congress and Exhibition on Intelligent Transportation Systems and Services (ITS)*, November 1999.

[HS92] John Hershberger and Jack Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In *SDH '92: Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 134–143, 1992.

[Huf52] David Huffman. A method for the construction of Minimum-Redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[HWLC97] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: theory and practice*. Springer-Verlag, 1997.

[iee08] IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, August 2008.

[II86] H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31–41, 1986.

[JAW67] E.N. Nilson J.H. Ahlberg and J.L. Walsh. *The Theory of Splines and Their Applications*, volume 38 of *Mathematics in Science and Engineering*. Academic Press (Elsevier), 1967.

[JOW+02] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS '02: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 96–107, 2002.

[Keh07] Steffen Kehl. *Querregelung eines Versuchsfahrzeugs entlang vorgegebener Bahnen*. PhD thesis, University of Stuttgart, Institute for System Dynamics, November 2007. In German language.

[Koe] Markus Koegel. Cornuconsole: A trajectory compression scheme using clothoidal spline curve fitting. Online resource. http://www.cn.uni-duesseldorf.de/staff/koegel/software/cornuconsole.

[KP08] Florian Kranke and Holger Poppe. Traffic Guard - Merging Sensor Data and C2I/C2C Information for proactive, Congestion avoiding Driver Assistance Systems. In *FISITA '08: World Automotive Congress of the Int'l Federation of Automotive Engineering Societies*, September 2008.

[Kro09] Lieuwe Krol. The reconstruction of vehicle trajectories with dynamic macroscopic data. Master's thesis, University of Twente, Enschede, Netherlands, August 2009.

[KSNS07] E. Kalogerakis, P. Simari, D. Nowrouzezahrai, and K. Singh. Robust statistical estimation of curvature on discretized surfaces. In *ES-SGP '07: Proceedings of the Eurographics/ACM Siggraph Symposium on Geometry Processing*, 2007.

[Kuc07] Horst Kuchling. *Taschenbuch der Physik*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 17 edition, August 2007. In German language.

[KWRKM05] S. Y. Kim, K. Wilson-Remmer, A. L. Kun, and W. T. III Miller. Remote fleet management for police cruisers. In *IV '05: Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 30–35, June 2005.

[LDR08] Ralph Lange, Frank Dürr, and Kurt Rothermel. Online trajectory data reduction using connection-preserving dead reckoning. In *MobiQuitous '08: Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking & Services*, July 2008.

[Lee61] C. Y. Lee. An Algorithm for Path Connections and Its Applications. *Electronic Computers, IRE Transactions on*, EC-10(3):346–365, September 1961.

[LFDR09] Ralph Lange, Tobias Farrell, Frank Dürr, and Kurt Rothermel. Remote Real-Time Trajectory Simplification. In *PerCom '09: Proceedings of the 7th IEEE International Conference on Pervasive Computing and Communications*, pages 184–193, March 2009.

[LHB06] Ryan Lever, Annika Hinze, and George Buchanan. Compressing gps data on mobile devices. In *OTM '06: Proceedings of the Conference On*

the Move to Meaningful Internet Systems, volume 4278 of *Lecture Notes in Computer Science*, pages 1944–1947. Springer, November 2006.

[LNRL06] Larissa Labakhua, Urbano Nunes, Rui Rodrigues, and Fátima S. Leite. Smooth trajectory planning for fully automated passengers vehicles — spline and clothoid based methods and its simulation. In *ICINCO '06: Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics*, pages 89–96, August 2006.

[LPFK05] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Building personal maps from gps data. In *MOO '05: Proceedings of the IJCAI Workshop on Modeling Others from Observation*, July 2005.

[LR01] Alexander Leonhardi and Kurt Rothermel. A comparison of protocols for updating location information. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 4(4):355–367, October 2001.

[Mac02] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[Mar79] G. Nigel N. Martin. Range encoding: An algorithm for removing redundancy from a digitised message. July 1979.

[MdB04] Nirvana Meratnia and Rolf A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT '04: Proceedings of the 9th International Conference on Extending Database Technology*, March 2004.

[MMH05] Luis Sardinha Monteiro, Terry Moore, and Chris Hill. What is the accuracy of DGPS? *The Journal of Navigation*, 58:207–225, May 2005.

[MPS02] Jaakko Myllylä and Yrjö Pilli-Sihvola. Floating car road weather monitoring. In *SIRWEC '02: Proceedings of the 11th International Road Weather Congress*, January 2002.

[MS09] James McCrae and Karan Singh. Sketching piecewise clothoid curves. *Computers & Graphics*, 33(4):452–461, 2009.

[MW92] D. S. Meek and D. J. Walton. Clothoid spline transition spirals. *Mathematics of Computation*, 59(199):117–133, July 1992.

[NPB95] Illah Nourbakhsh, Rob Powers, and Stan Birchfield. DERVISH – an office-navigating robot. *The AI magazine*, 16(2):53–60, 1995.

[NR07] Jinfeng Ni and Chinya V. Ravishankar. Indexing Spatio-Temporal Trajectories with Efficient Polynomial Approximations. *IEEE Transactions on Knowledge and Data Engineering*, 19:663–678, May 2007.

[ope] Open Data Commons Open Database License (ODbL). http://opendatacommons.org/licenses/odbl/.

[Pav] Igor Pavlov. 7-zip. Online resource. http://7-zip.org.

[PPS06] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM '06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 275–284, July 2006.

[proa] The *AKTIV* project. http://www.aktiv-online.org.

[prob] The *COMeSafety* project. http://www.comesafety.org.

[proc] The *Network on Wheels* project. http://www.network-on-wheels.de.

[prod] The sim^{TD} project: Safe and Intelligent Mobility – Test Field Germany. http://www.simtd.de.

[proe] The OpenStreetMap Project. Online resource: http://www.openstreetmap.org/.

[PV09] J. Portell and A. G.and Garcia-Berro Villafranca. A resilient and quick data compression method of prediction errors for space missions. In B. Huang, A. J. Plaza, and R. Vitulli, editors, *Satellite Data Compression, Communication, and Processing V*, 2009. vol. 745505.

[RBFT99] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *ICRA '99: Proceedings of the IEEE Int'l Conference on Robotics and Automation*, pages 35–40, August 1999.

[RGRB04] S. Roberts, T. Guilford, I. Rezek, and D. Biro. Positional entropy during pigeon homing i: application of bayesian latent state modelling. *Journal of Theoretical Biology*, 227(1):39–50, 2004.

[RHS09] A. Reinhardt, M. Hollick, and R. Steinmetz. Stream-oriented lossless packet compression in wireless sensor networks. In *SECON '09: Proceedings of the 6rd Annual IEEE Communications Society Conference*

on Sensor and Ad Hoc Communications and Networks, pages 1–9, June 2009.

[RMB+10] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions Sensor Networking*, 6(2):13:1–13:27, March 2010.

[SA09] N. Schüssler and K.W. Axhausen. Processing raw data from global positioning systems without additional information. *Transportation Research Record: Journal of the Transportation Research Board*, 2105:28–36, October 2009.

[SBM+07] Gail Schofield, Charles M. Bishop, Grant MacLean, Peter Brown, Martyn Baker, Kostas A. Katselidis, Panayotis Dimopoulos, John D. Pantis, and Graeme C. Hays. Novel gps tracking of sea turtles as a tool for conservation management. *Journal of Experimental Marine Biology and Ecology*, 347(1–2):58–68, 2007.

[Sewa] Julian Seward. bzip2. Online resource. http://www.bzip.org.

[Sewb] Julian Seward. bzip2 documentation, Section 4.1. Limitations of the compressed file format, version 1.0.5. Online resource, as of 10/25/2012. http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.html#limits.

[SH82] Karl-Heinz Schimmelpfennig and Norbert Hebing. Geschwindigkeiten bei kreisförmiger Kurvenfahrt – Stabilitäts- und Sicherheitsgrenze. *Der Verkehrsunfall*, 20(5):97–99, May 1982. In German language.

[Sha48] Claude Elwood Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[SK95] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *IJCAI '95: Proceedings of the 14th international joint conference on Artificial intelligence – Volume 2*, pages 1080–1087, August 1995.

[SK07] Igor Skrjanc and Gregor Klancar. Cooperative collision avoidance between multiple robots based on bézier curves. In *ITI '07: Proceedings of the 29th International Conference on Information Technology Interfaces*, pages 451–456, June 2007.

[sma] Smartrunner: Marathon, running, sports tracker, gps, trails. Online resource: http://www.smartrunner.com/.

[STBW02] Ralf-Peter Schäfer, Kai-Uwe Thiessenhusen, Elmar Brockfeld, and Peter Wagner. A traffic information system by means of real-time floating-car data. In *ITSWC '02: Proceedings of the 9th World Congress and Exhibition on Intelligent Transportation Systems and Services (ITS)*, October 2002.

[Str95] Jürgen Strobel. *Global-Positioning-System : GPS; Technik und Anwendung der Satellitennavigation; mit 6 Tabellen.* Franzis, Poing, 1995.

[SWR+04] Stefan Schroedl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. Mining GPS Traces for Map Refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, July 2004.

[TCS+06] Goce Trajcevski, Hu Cao, Peter Scheuermann, Ouri Wolfson, and Dennis Vaccaro. Online data reduction and the quality of history in moving objects databases. In *MobiDE '06: Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, June 2006.

[THR07] S.T.S. Thong, Chua Tien Han, and T.A. Rahman. Intelligent fleet management system with concurrent gps gsm real-time positioning technology. In *ITST '07: Proceedings of the 7th International Conference on Intelligent Transportation Systems (ITS) Telecommunications*, pages 1–6, June 2007.

[Tim02] N.H. Timm. *Applied multivariate analysis.* Texts in statistics. Springer, 2002.

[vD98] Frank van Diggelen. GPS Accuracy: Lies, Damn Lies and Statistics. *GPS World*, 9(1):41–45, November 1998.

[VD00] K.L. Van Dyke. The world after sa: benefits to gps integrity. In *Proceedings of the IEEE 2000 Position Location and Navigation Symposium*, pages 387–394, March 2000.

[Vit87] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM*, 34(4):825–845, October 1987.

[VMR+10] Alberto Villafranca, Iu Mora, Patrizia Ruiz, Jordi Portell, and Enrique García-Berro. Optimizing gps data transmission using entropy coding compression. In *SDCCP '10: Proceedings of the 6th SPIE Conference on Satellite Data Compression, Communications, and Processing*, August 2010.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.

[Wel84] Terry A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.

[Woo07] Oliver J. Woodman. An introduction to inertial navigation. Technical Report UCAM-CL-TR-696, University of Cambridge, Computer Laboratory, August 2007.

[WSCY99] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–287, July 1999.

[ZCL+10] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on gps data for web applications. *ACM Transactions on the Web*, 4(1):1:1–1:36, January 2010.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, September 1978.

Index