Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Contribution to the Long Term Prediction of Motion Trajectories

Dissertation zur Erlangung des Doktorgrades der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau

Konrad Schenk

Betreuer:	Prof. Dr. H.-M. Groß
Gutachter:	Prof. Dr. Hans-Joachim Böhme
	Prof. Dr. Christian Wöhler

Die Dissertation wurde am 13.10.2017 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht und am 11.07.2018 verteidigt.

urn:nbn:de:gbv:ilm1-2018000314

Kurzfassung

Viele Anwendungen in der mobilen und kognitiven Robotik erfordern einen Prädiktionsmechanismus, um die zukünftigen Aufenthaltsorte bewegter Objekte zu schätzen. Ein autonomes Auto muss beispielsweise die Absichten der anderen Verkehrsteilnehmer schätzen können, um Kollisionen zu vermeiden und die Verkehrsregeln einzuhalten. Ein Serviceroboter muss hingegen in der Lage sein, die Bewegungsspuren der Personen in seiner Umgebung vorherzusagen, um in einer sozial akzeptablen Art und Weise zu navigieren und die Passanten nicht zu behindern.
Fast alle Prädiktionsalgorithmen, die in der Literatur zu finden sind, beschäftigen sich mit der Kurzzeitprädiktion und sind auf spezielle Problemstellungen angepasst. Die Lösung einer neuen Problemstellung, welche eine Langzeitprädiktion benötigt (z.B. ein personalisierter Shopping-Assistent, oder eine intelligente Stauvorhersage), ist daher oft mit umfangreichem Forschungs- und Entwicklungsaufwand verbunden.
Das Ziel dieser Dissertationsschrift liegt darin, sich dieses Defizits anzunehmen und der wissenschaftlichen Gemeinschaft ein vielseitig einsetzbares Langzeitprädiktionsframework zur Verfügung zu stellen. Das Framework trifft keine Annahmen über das jeweilige System und kann somit auf einfache Art und Weise an die spezifischen Anforderungen der individuellen Problemstellung angepasst werden. Das Framework selbst besteht aus drei Elementen:

- Ein topologisches Modell, welches mit Hilfe eines Clustering Algorithmus anhand von Beobachtungen erstellt wird. Daraus resultiert ein topologischer Graph, welcher den Zustandsraum effizient abbildet und eine praktkable Repräsentation von Trajektorien ermöglicht.

- Ein probabilistisches Modell, welches den topologischen Graphen um Übergangswahrscheinlichkeiten und Wahrscheinlichkeitsverteilungen der Übergangszeiten ergänzt.

- Das eigentliche Prädiktionsframework, welches beide Modelle integriert. Mit Hilfe eines flussbasierten Algorithmus errechnet es für eine gegebene Eingabetrajektorie die zukünftigen Aufenthaltswahrscheinlichkeitsverteilungen über den gesamten Zustandsraum.

Die im Rahmen dieser Arbeit durchgeführten Experimente zeigen, dass das vorgestellte Langzeitprädiktionsframework für Bewegungstrajektorien in der Lage ist, sich mit mehreren State of the Art Algorithmen zu messen, ohne dabei auf problemspezifische Bewegungsmodelle zurückzugreifen, physikalische Gesetze zu beachten, oder einschränkende Annahmen über den Zustandsraum des Systems zu treffen. Weiterhin enthalten die Experimente umfangreiche Auswertungen und Ergebnisse, um einen aussagekräftigen Vergleich mit künftigen Prädiktionsalgorithmen zu ermöglichen.

Abstract

Most applications of mobile and cognitive robotics require a prediction mechanism to estimate the future positions of moving objects. An autonomous car, for example, needs to determine the intentions of other traffic participants to avoid collisions and to obey the traffic rules. A service robot, on the other hand, needs to anticipate the paths of the surrounding pedestrians in order to move in a socially acceptable manner and to avoid awkward situations.

Almost all prediction algorithms presented in literature mainly focus on the short term time horizon and usually give a solution tailored to a specific application. Thus, extensive research and development is necessary if new applications (e.g., a personalized shopping assistant or an intelligent traffic forecast) require a long term prediction mechanism.

The goal of this thesis is to address this deficit and contribute a versatile long term prediction framework to the scientific community. It provides an algorithm which can easily be adapted to the individual task at hand by avoiding system specific assumptions such as motion characteristics, physical properties, or spatial restrictions. The framework consists of three elements:

- A topological model which is based on observations and is created by utilizing a clustering algorithm. It incorporates a topological graph, sampling the state space efficiently and enabling a convenient representation of trajectories.
- The topological model is enriched with a probabilistic model by encoding transitional probabilities and transitional time distributions into the graph.
- Both models are integrated into the main prediction framework. By using a flow based algorithm, it provides the future probability distribution for a given input trajectory over the whole state space as a result.

The experiments in this thesis show that the presented long term motion prediction framework is able to compete with a variety of state of the art algorithms. Furthermore, they include an extensive set of evaluations and results to enable an expressive comparison to future prediction algorithms.

Acknowledgments

Before addressing the matter of long term motion prediction, I would like to thank everyone that helped and supported me on my journey of writing this thesis.
First and foremost, I would like to thank my doctoral advisor, Prof. Dr. Horst-Michael Groß, for giving me the opportunity to conduct my research at the Neuroinformatics and Cognitive Robotics Lab on the exciting topic of motion prediction. Without you, I may never have discovered the fascinating world of robots and neural networks in the first place.
Thank you to the entire departmental staff for your guidance, assistance, vivid discussions, and friendship. You all contributed to the positive atmosphere which made it a pleasure to come to work every morning. I would especially like to acknowledge my former colleagues and teammates Alexander Kolarow and Markus Eisenbach for broadening my perspective and for the good times we shared; Dr. Klaus Debes for keeping as much bureaucratic work off of our shoulders as possible; Ute Schütz, Katja Hamatschek, and Anja Zwetkow-Schilling for meticulously organizing the lab; and Heike Groß and Sabine Schulz for always keeping the IT running.
Furthermore, I am grateful for my beloved wife, Kate, who inspired me to write this thesis in English, diligently checked the grammar and orthography of every new draft of a section or chapter, and who gave me a backrub when I needed one. Thank you, Barbara Zand, for proofreading and correcting the text. You taught me a lot about copy editing.
Finally, I would like to express my gratitude to my parents, Charlotte and Thomas, and the rest of my family for always believing in me. Your love and unwavering support encouraged me to pursue my dream of earning a doctoral degree.

Contents

Chapter 1

Introduction

Information and communication technology have advanced rapidly within the last few years. Not only are new consumer electronics and faster computers announced on a regular basis - vacuum cleaning robots, service robots and autonomous cars have also been grabbing the media's attention recently. The abilities of machines have changed from planning, regulating and inferring capabilities to cognitive, anticipatory and smart services (e.g., [Kolarow et al., 2013, Gross et al., 2014, Schroeter et al., 2013, Stricker et al., 2012]). These aptitudes are necessary for a lot of new developments. An autonomous car, for example, does not only need to be able to determine and follow the best route to the destination, it also needs to recognize traffic signs, pay attention to traffic lights, detect other vehicles on the road, and estimate their intentions. One necessity for some of these tasks is the ability to predict movements - not only just a few seconds into the future to prevent collisions but also several seconds or minutes to determine the right of way in advance [Hermes et al., 2009].

1.1 Contributions

This thesis highlights the long term prediction since literature unjustifiably pays little respect to this issue. The few publications covering this topic only give tailored solutions to specific tasks which cannot be transferred to different problems without hurdles. In order to avoid developing a particular algorithm for every new prediction task, a versatile prediction framework is presented in this thesis. It can be easily adapted to different specific prediction tasks and provides an assertion over the prospective probability distributions of moving objects. Unlike several other prediction algorithms, the probability of finding the object at one place should not be provided for just a single or a few discrete points, but for the whole attainable state space. It enables us to

not only answer the question about the most probable state but also to determine the probability of a specific state or even of a whole interval of states. It is safe to assume that it needs many calculations and, therefore, considerable computational capabilities to provide such a result; but, this thesis will provide a computationally less expensive solution by transferring the problem into the time domain and an even more efficient algorithm using an iterative approach.
The prediction algorithm will basically consist of three elements. At first, a topological state model is needed in order to encode the position of the object or the state of the system to be more general. The state space is not only restricted to Cartesian coordinates. It can also, for example, include the temperature, pressure, and proportions of a chemical process, or the trade volume, revenue, and quotation of a stock which needs to be predicted. The second element of the prediction algorithm provides a probabilistic model representing the transition probabilities and periods between all positions or states. Both models, which are dependent on each other, are merged into the third element - the main prediction framework, which finally calculates the probability distribution.

1.2 Application Scenarios

The foundation for this thesis was laid during the research project "APFel" (Analyse von Personenbewegungen an Flughäfen mittels zeitlich rückwärts- und vorwärtsgerichteter Videodatenströme - Analysis of Person Movement at Airports via Temporal Backward and Forward Video Data Streams) [Kolarow et al., 2013]. The goal of this project was to provide a proof of concept of an intelligent video recorder assisting security personnel at airports in searching through hours of video footage on multiple cameras. After selecting a person in one camera image, the system searches for this person's every appearance in all video recordings. In order to support the involved re-identification and to filter the relevant video data, a prediction algorithm was implemented for calculating the most probable appearances of the person on every camera. Despite the highly specific application, the prediction framework was designed to be as versatile as possible from the beginning in order to be easily adaptable to a variety of other prediction tasks:

- In mobile communications for instance, it is useful to know the next cell tower a user is going to connect to in advance. This knowledge can be used to improve the hand-off delay and optimize the network utilization [Wanalertlak et al., 2011, Verhein and Chawla, 2006].

- The main application of a prediction for autonomous vehicles is collision avoidance. It is crucial for an autonomous car to estimate the future positions of other vehicles and pedestrians in order to avoid accidents and to optimize its own behavior [Wiest et al., 2012].

- Since mobile service robots are just another type of autonomous vehicle, they clearly also need a collision avoidance that is ideally based on the prediction of every other non-static object in its vicinity. Additionally, it enables the robot to utilize an anticipatory situation assessment in order to change its navigational strategies from a purely reactive collision avoidance [Burgard et al., 1999] into a more socially acceptable navigation policy [Gross et al., 2014, Kuderer et al., 2012, Weinrich et al., 2013].
- In most applications of a service robot (e.g., [Gross et al., 2014, Schroeter et al., 2013, Stricker et al., 2012]), it is important to estimate the future as well as the current whereabouts of people. If the robot is not able to observe them directly (e.g., if the robot is in another room or it is unable/not allowed to follow the person of interest), it can estimate their current positions based on previous observations which is equivalent to a prediction [Bennewitz et al., 2005].
- In marketing, several metrics are in existence to describe a possible customer [Bearden and Netemeyer, 1999, Dees et al., 2008]. Extensive research is being conducted to anticipate a customers needs and his profitability [Drengner et al., 2011, Malthouse and Blattberg, 2005]. A prediction algorithm may not only be useful to predict a shoppers spatio-temporal trajectory but also to estimate high-order metrics to describe more abstract properties of a possible customer.

1.3 Publications

Since the the proposed prediction framework was developed as part of a research project, some elements of this thesis were already published at international and national conferences:

- [Schenk et al., 2011] Schenk, Konrad, M. Eisenbach, A. Kolarow and H.-M. Gross (2011). *Comparison of laser-based person tracking at feet and upper-body height.* In *KI 2011: Advances in Artificial Intelligence*, pp. 277–288. Springer Berlin Heidelberg
The proposed tracking mechanism was used to create the Humboldt dataset as presented in Section 6.1. It transforms LIDAR data into Cartesian coordinates and tracks human-shaped point clouds with multiple particle filters.

- [Schenk et al., 2012a] Schenk, Konrad, A. Kolarow, M. Eisenbach, K. Debes and H. Gross (2012a). *Automatic calibration of multiple stationary laser range finders using trajectories.* In *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, pp. 306–312. IEEE

- [Schenk et al., 2012b] Schenk, Konrad, A. Kolarow, M. Eisenbach, K. Debes and H.-M. Gross (2012b). *Automatic calibration of a stationary network of laser range finders by matching movement trajectories.* In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 431–437. IEEE
  The methods presented in [Schenk et al., 2012a] and [Schenk et al., 2012b] were used to automatically align multiple LIDAR sensor units in a Cartesian coordinate system. It helps significantly in reducing the time needed to set up the hardware and enables the tracking algorithm of [Schenk et al., 2011] to cover the whole Humboldt Foyer.

The following publication was co-authored as part of this work:

- [Kolarow et al., 2013] Kolarow, Alexander, K. Schenk, M. Eisenbach, M. Dose, M. Brauckmann, K. Debes and H.-M. Gross (2013). *APFel: The intelligent video analysis and surveillance system for assisting human operators.* In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pp. 195–201. IEEE
  The presented tracking system is able to track people with several non-overlapping cameras using HOG detectos, a fast object tracking, and a person re-identification mechanism. If a person leaves a camera viewport, a precursor of the proposed predcition framework helps in constraining the number of cameras and time periods for the otherwise time consuming search for that person in the video footage.

Unrelated to this thesis, the following publications were also co-authored:

- [Hermes et al., 2009] Hermes, Christoph, C. Wohler, K. Schenk and F. Kummert (2009). *Long-term vehicle motion prediction.* In *Intelligent Vehicles Symposium, 2009 IEEE*, pp. 652–657. IEEE
  A prediction algorithm using a tree search on motion pattern is presented in [Hermes et al., 2009] in order to predict car movements at crossings. It implements a similarity measure which is invariant to transformations and rotations and able to match partial pattern.

- [Eisenbach et al., 2012] Eisenbach, Markus, A. Kolarow, K. Schenk, K. Debes and H. Gross (2012). *View invariant appearance-based person reidentification using fast online feature selection and score level fusion.* In *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, pp. 184–190. IEEE
  A key component used in [Kolarow et al., 2013] is the person re-identification algorithm presented in this publication. It uses a fast online feature selection to reidentify a person regardless of its orientation, occlusions, and lighting.

- [Kolarow et al., 2012] Kolarow, Alexander, M. Brauckmann, M. Eisenbach, K. Schenk, E. Einhorn, K. Debes and H.-M. Gross (2012). *Vision-based hyper-real-time object tracker for robotic applications.* In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2108–2115. IEEE
  A fast vision-based object tracker is presented in this publication. It utilizes a small number of simple features which are drawn from homogeneous regions of the object to be tracked. These features allow for a fast logarithmic search and enable the tracker to perform significantly faster than real-time on an HD video stream using consumer hardware.

- [Eisenbach et al., 2013] Eisenbach, Markus, P. Scheiner, A. Kolarow, K. Schenk, H.-M. Gross and I. Weinreich (2013). *Learning Illumination Maps for Color Constancy in Person Reidentification.* Workshop Farbbildverarbeitung, 19
  Color constancy is a requirement for most re-identification algorithms in order to track people across multiple cameras. This publication presents a method to automatically learn varying lighting situations in a camera viewport in order to compensate for changes in color and improve the performance of re-identification algorithms.

1.4 The Thesis' Outline

This thesis is structured as follows: In Chapter 2, an overview of state-of-the-art prediction algorithms is given followed by a more thorough review of some selected long term prediction methods. In Chapter 3 to 5, the individual parts of the prediction framework are explained. In order to evaluate the performance of the prediction framework, different methods are examined with experiments in Chapter 6, and the best version is compared to the state of the art. The results and the contributions of this thesis are summarized in Chapter 7 and possibilities for future work are depicted.

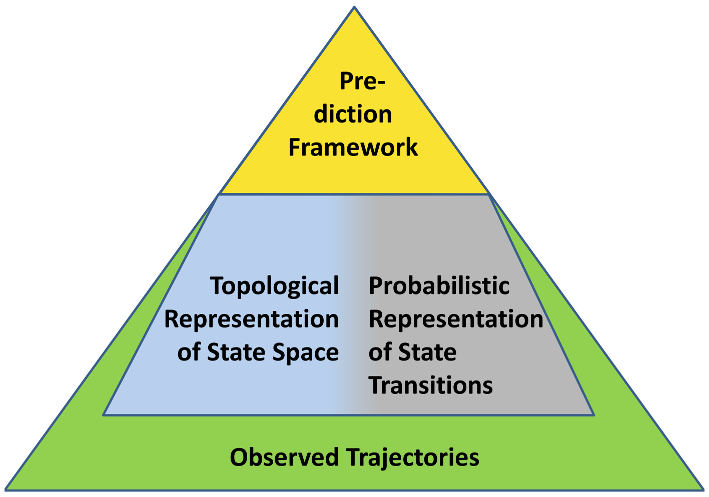


Figure 1.1: *The presented prediction framework basically consists of three elements. The first element incorporates a model which represents the topology of the state space. A second probabilistic model is affiliated with it which describes the transitions between the states encoded into the topological model. Based on those two modules, the prediction framework as the third element can then compute the future probability distribution given an observation. A comprehensive trajectory basis is necessary in order to provide a representative model of the topology and the state transitions.*

The prediction framework itself is assembled as follows: Based on observed trajectories, a topological model, which is illustrated in Chapter 3, is built first. It is used to encode the spatial information about the space in which the prediction task needs to be solved. The topological representations are then extended by a probabilistic model, as described in Chapter 4, in order to reflect the information gained from previously taken observations. Both models are finally combined into the prediction framework, which takes a current observation and calculates the future spatio-temporal probability distributions. The structure and algorithm of the framework itself is presented in Chapter 5. An overview of the prediction framework is given in Figure 1.1.

Chapter 2

Long Term Motion Prediction: State of the Art

In the previous chapter, the long term motion prediction was introduced, examples of its application were given, and the main contributions of this thesis were highlighted. The following chapter gives an overview of the state of the art of long term motion prediction with a main focus on an all-purpose solution.
In order to compare the thesis to the current literature, a categorization is given in Section 2.1. To unveil a common workflow, a broad overview to different approaches was chosen. In Section 2.2, the three most versatile approaches are presented and evaluated to give a motivation for the methods selected in this thesis.

2.1 Overview

In essence, movement prediction can be seen as time series forecasting. Research on this topic dates back several decades ([Kemeny and Snell, 1960, Makhoul, 1975]) but application specific research has been conducted primarily in recent years.
In the data mining community, the main focus is put on mobile services and traffic management. The knowledge of the future whereabouts while driving a car can be used for informing the driver about upcoming gas stations, traffic jams, or points of interest along the route [Jeung et al., 2010, Krumm, 2008, Monreale et al., 2009]. By predicting the movements of not only one, but multiple cars, congestion and traffic jams can be anticipated and traffic management may be optimized significantly [Bachmann et al., 2013].
For pedestrians using smart phones or other wearable devices, movement prediction can be utilized in location based advertising (e.g., restaurants or shops along the road) or for optimizing the cell-handover in wireless networks [Verhein and Chawla, 2006].

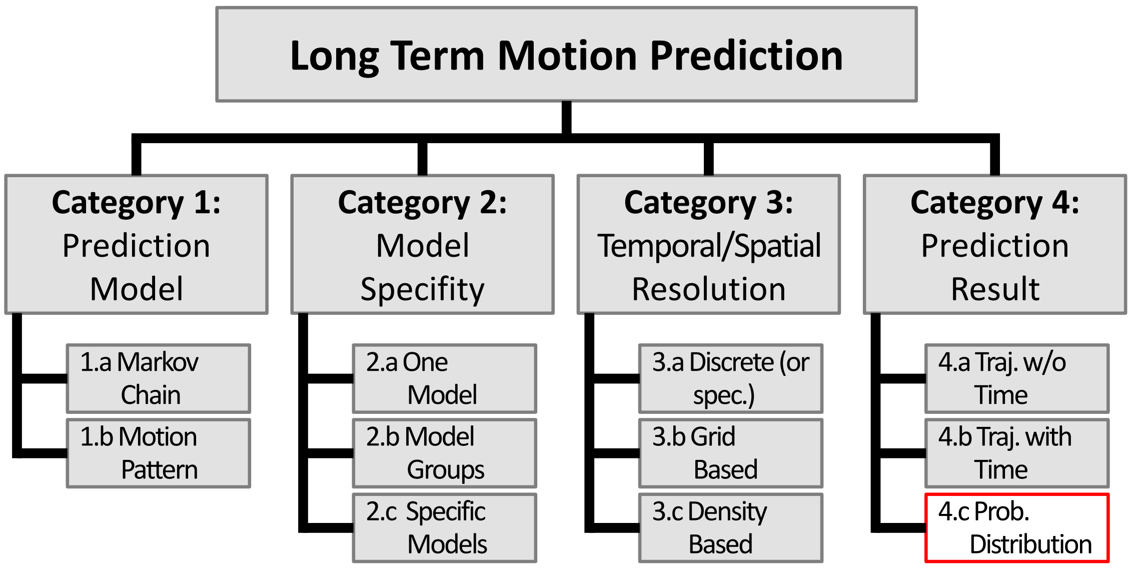


Figure 2.1: *Categories and aspects for comparing prediction algorithms as stated in [*Bachmann *et al., 2013] and described in Section 2.1. The last item in category 4 was added since none of the options available in [*Bachmann *et al., 2013] suited the prediction method presented in this thesis.*

In mobile robotics, movement prediction is a key element in navigation and path planning. Estimating the future positions of moving objects (e.g., other cars for autonomous vehicles or pedestrians for service robots) does not only improve collision avoidance but also helps in avoiding congested areas and finding routes optimal for the robot and other participants [Ikeda et al., 2013, Vasquez Govea, 2007].

Most prediction algorithms found in literature can be compared according to four different categories, as stated in [Bachmann et al., 2013]. Figure 2.1 shows the main categories and their different aspects. A comprehensive comparison of the current literature regarding these aspects is shown in Figure 2.2.

First, the prediction model can be separated between Markov chains or equivalent methods (see 1.a in Figure 2.1 and 2.2) and motion patterns (1.b). While the former is based on the assumption that only the last n observations are relevant for estimating the future positions with probabilistic methods, the latter uses the distance of input trajectories to observed trajectories to give a prediction about the future state.

Another criterion to differentiate between the models is the specificity of their movement models. The most common approach (2.a) is to use one model for all objects. It is often based on the assumption that only one class of objects is present in the state space (e.g., every object on a road is deemed a car - regardless of weather it is a person or a bicycle). On the other hand, a model for every individual object can be learned (2.b). Intermediate approaches with different models for different kinds of

	Asahara et al., 2011	Ashbrook and Starner, 2003	Bennewitz, 2004	Fulgenzi et al., 2009	Giannotti et al., 2007	Hellbach, 2010	Ikeda et al., 2013	Jeung et al., 2008	Jeung et al., 2010	Krumm, 2008	Kuderer et al., 2012	Monreale et al., 2009	Prasad and Agrawal, 2010	Vasquez Goeva, 2007	Verhein and Chawla, 2006	Wanalertlak et al., 2011	Weser et al., 2006	Wiest et al., 2012	Ye et al., 2009	Ying et al., 2011	Ziebart et al., 2009	**this thesis**
1.a Markov chain	x	x		x			x		x	x			x	x		x	x				x	x
1.b Motion pattern			x		x	x		x			x	x			x			x	x	x		
2.a One model	x		x	x	x	x	x	x	x	x	x	x	x	x	x		x	x			x	x
2.b Model groups																x				x		(x)
2.c Spec. models		x								x									x			(x)
3.a Discrete (or spec.)									x	x		x	x			x						(x)
3.b Grid based		x			x	x		x							x				x	x	x	(x)
3.c Density based	x		x	x	x		x				x	x		x	x		x	x				x
4.a Traj. w/o time	x	x								x			x			x	x			x		
4.b Traj. with time			x	x	x	x	x	x	x		x	x		x	x			x	x		x	
4.c Prob. distribution																						x

Figure 2.2: *A comparison of different prediction methods found in literature. The rows represent the different categories for comparing prediction algorithms, as shown in Figure 2.1, and the columns show into which category each algorithm falls. As can be seen, the method presented in this thesis uses a prediction model equivalent to Markov chains and specifies one movement model for all objects. However, it may also use one model for different groups of objects or even one model for every object. The spatial resolution corresponds to the data density but custom designed or grid-based resolutions are also possible. The final result the prediction method presented in this thesis provides is a probability distribution - a result no other method found in literature is able to give as an output.*

objects (e.g., one model for cars and another one for persons) are also possible (2.c). The temporal and spatial resolution may be another distinctive feature. It can be divided between discretized or custom-designed resolutions like street segments, sensor nodes, or cell towers (3.a), equidistant or grid-based representations (3.b), and continuous or density-based subdivisions (3.c).
The last criterion may be the type of prediction result. Some algorithms only provide future trajectories without any temporal information (4.a), whereas other methods give one or few time-discrete sequences of states, which are considered to be the most probable trajectories (4.b).
Aside from a few exceptions, a lot of these algorithms have strong similarities in their workflow. At first, a topological representation of movement trajectories is determined: either by motion pattern or by a discretized state space. Furthermore, a model is learned, which encodes probabilistic state transitions or similarities to observed movements. Both parts are usually dealt with in a learning phase and processed by an algorithm in an on-line phase in order to predict an observation.

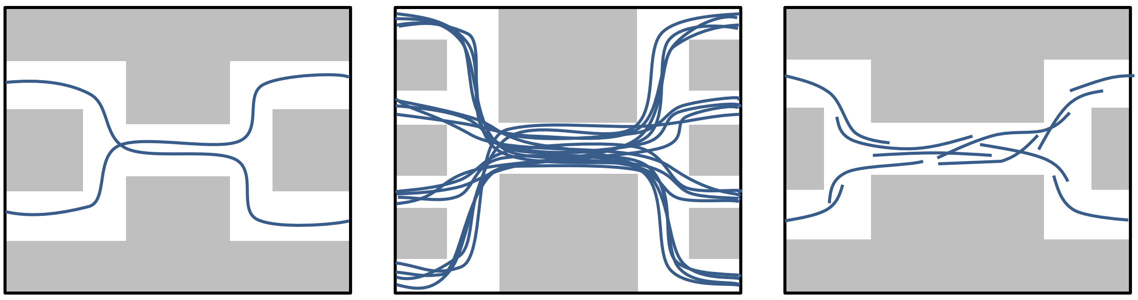
The prediction method presented in this thesis can be categorized under the first criterion as Markov-based. Since only one model is used for every object, the classification under the second criterion is obvious. However, it should be mentioned that specific models for every object or class of objects are not precluded. The spatial resolution can be categorized as density-based. But, grid-based and application specific subdivisions may also be applied. Based on the fourth criterion, the proposed algorithm falls under the time-discrete sequence of states category though, in contrast to comparable publications, it does not predict only one or a few states per time step. Instead, it provides a probability distribution over the entire state space. This achievement enables the computation of probabilities for specific states or intervals in the state space.

2.2 Selected Methods

Most of the above mentioned prediction algorithms are restricted to their specific application and are, therefore, not easily adaptable to other problems. Only a few publications pursue a versatile approach. Promising methods were presented in [Bennewitz, 2004], [Vasquez Govea, 2007], and [Ikeda et al., 2013], and they will now be surveyed in the following paragraphs.

2.2.1 [Bennewitz, 2004]

The thesis written by Bennewitz [Bennewitz, 2004] deals with the path planning for swarms of robots and with the navigation of individual service robots in the presence of people. The main focus lies on motion patterns of people representing typical trajectories between resting places. The robot must learn the pattern based on observations. It obtains the positions of the person with sensors and clusters similar trajectories into motion pattern based on an expectation-maximization-algorithm [McLachlan and Krishnan, 2007]. After procuring the patterns, the robot can use them to perform a socially acceptable navigation or to predict the current positions of people by comparing the previously observed trajectory with all motion patterns and calculating the probabilities that this observation results from each pattern. Based on the probabilities for each motion pattern, a probabilistic statement about the whereabouts of a person can be made.
Although it is not claimed to provide a versatile prediction algorithm, the applied methods lead one to assume that they could also be applied to solve different problems. For example, the creation and utilization of motion pattern does not make any people-specific assumptions and merely follows probabilistic aspects which makes it possible to model different systems in the same manner. Furthermore, the thesis explicitly points out that the motion pattern are not restricted to two dimensions. Thus, the state space



(a) No inference (b) Redundancy (c) Prone to errors

Figure 2.3: *Overview of the drawbacks of motion pattern (MP). The MP are shown in top view as blue lines, and obstacles are depicted in gray. The inability of MP to infer new behaviors is shown in (a). The learned MP are modeling two movements crossing each other in a narrow corridor. A trajectory from the upper left to the upper right is not represented though it could have been modeled due to the overlap in the middle. In (b) the redundancy of MP is shown. Although they are similar, all 16 possible combinations from left to right have their own representation in the middle corridor. The deficits of interrupted observations are shown in (c). If an observation is interrupted, the MP cannot be continued and a new one is created as soon as further observations are available. This results in multiple small MP which could have been connected based on spatial information.*

can be extended with a temporal dimension or additional parameters. Yet despite all of these advantages, the utilization of motion pattern should be questioned in the context of a general purpose prediction algorithm. Since they are goal-oriented, an intention in the movement is implicated and stochastic or reactive movements cannot be modeled. Additionally they are only able to predict already observed trajectories and cannot infer new movements by combining different observations (see Fig. 2.3(a)). Another disadvantage is the redundancy in motion pattern partially covering the same areas. Regions in which several motion patterns are similar to each other are encoded in each individual pattern. This can result in a tremendous amount of redundant data, as shown in Fig. 2.3(b). As depicted in Fig. 2.3(c), a database with several broken tracks will result in an unusable set of motion patterns. If an observation is interrupted, the corresponding motion pattern will also be interrupted and a new, unrelated one is created as soon as new observations are available. Due to these drawbacks, motion patterns are not pursued in this thesis and a more suitable approach will be utilized in order to implement a versatile prediction algorithm.

2.2.2 [Vasquez Govea, 2007]

Dizan Vasques brought similar arguments forward against the use of motion pattern in his thesis [Vasquez Govea, 2007] and introduced an alternative approach for predicting the movements of people and vehicles. In order to implement an iteratively expendable prediction framework, a modification of the Hidden Markov Model (HMM - see [Rabiner, 1989]), called Growing Hidden Markov Model (GHMM) was developed. This model is able to adapt its topology with the help of an Instantaneous Topological Map (published in [Jockusch and Ritter, 1999]) according to new observations without the need to recalculate the whole topology in a batch. The parameters of GHMM are obtained with an incremental version of the Baum-Welch-Algorithm [Baum et al., 1970] exactly like a regular HMM. Also similar to HMM, a prediction based on GHMM is made by using observations to calculate its belief and propagating it into the future by the desired time horizon.

The thesis by Vasquez calls for the prediction framework not to be restricted to people and vehicles, but to be versatile due to the use of GHMM which can be learned iteratively. However, the use of HMM or one of its modifications as a multi-purpose approach can be questioned. Justifiably, the wide usage of HMM is given as an advantage. A lot of optimized methods for learning the parameters exist in literature, and HMMs were successfully applied for classification (e.g., [Koller-Meier and Van Gool, 2002, Makris and Ellis, 2002, Oliver et al., 2000]) and prediction (i.e., [Bennewitz et al., 2005, Fulgenzi et al., 2009, Prasad and Agrawal, 2010]) several times. But, the basic assumption of Hidden Markov Models is that the states of the underlying system are not directly observable (i.e., hidden) and they can only be estimated by indirect observations. But, in the proposed prediction framework, HMM is not suitable since the 'hidden' states are directly observable and ordinary Markov chains [Kemeny and Snell, 1960] can be utilized. If different object states, indirectly resulting from observations, are used as hidden states, system specific knowledge (e.g., the object dynamics, the intended goal, the influence of the environment, etc.) is needed and a versatile prediction method is precluded.

A further disadvantage of HMMs is their assumption of a first order Markov Process (i.e., the current state is only dependent on the previous state). It restricts their ability to learn more complex dynamics resulting in a mediocre prediction accuracy on challenging problems. This disadvantage was circumvented in [Vasquez Govea, 2007] by extending the state space with a dimension representing the intended goal. Unfortunately, this approach severely restricts the applicability since the intended goal is usually not known beforehand or the database for learning GHMM may not always yield this information due to gaps and interruptions in the trajectories. Due to these reasons, HMMs were not shortlisted in this thesis and another approach as in [Vasquez Govea, 2007] was taken which bears a resemblance to the following method.

2.2.3 [Ikeda et al., 2013]

In [Ikeda et al., 2013], a prediction mechanism was presented enabling a service robot to navigate around a shopping mall in a socially acceptable manner and to approach people in order to offer them services. The algorithm is based on the concept of sub-goals which are points in space people tend to approach frequently. They are determined by observations and enriched by a transition model incorporating the transition probabilities between the sub-goals. In order to predict a currently observed trajectory, the sequence of already approached sub-goals is determined and the current velocity is calculated. Both sets of information are combined into the transition model, similar to Markov chains, which infers the future positions of the observed person.
Using such a transition model is feasible since it only considers probabilistic aspects and no further assumptions about the system are made except that the future state is only dependent from the last n states (in [Ikeda et al., 2013], $n = 6$ was chosen, resulting in a sixth order Markov chain). Unfortunately, there was a strong focus on person movements which is reflected in the sub-goals and their topology. For instance, if a person approaches the current sub-goal, it is assumed that the person changes towards the next sub-goal as soon as it is visible. Therefore, explicit knowledge about the environment is needed in order to calculate the lines of sight. Furthermore, the algorithm for calculating the sub-goals needs to be provided with a suitable number of nodes which, in turn, also requires knowledge about the environment. As mentioned in the previous paragraph, the prediction assumes a constant velocity for the person disregarding influences of the surroundings like different surfaces, slopes, or narrows. Thus, only the probabilistic transition model is suited for a versatile prediction framework and different methods for the overall prediction algorithm and for finding a topology need to be chosen.

2.3 Conclusion

The state of the art is currently focusing on long term prediction in specific applications and is, therefore, using tailored algorithms, which cannot be transferred to different problems without hurdles. This thesis takes this deficit and presents a generic algorithm which can be adapted to different problems dealing with long term prediction. A suitable topological model that does not make any assumptions about the state space will be presented, and a proper probabilistic transition model capable of inferring new behaviors out of old observations and providing a probability distribution over the whole state space will also be developed. Both models are described in detail in the next two chapters.

Chapter 3

Topological Representation of State Space

An overview of the state of the art in long term motion prediction and the common workflow in most publications was presented in the previous chapter. Three promising prediction methods were evaluated concerning their versatility showing some insufficiencies for providing a comprehensive long term motion prediction. A topological representation of the state space, a probabilistic representation of state transitions, and an algorithm for processing both representations on-line were identified as the three main elements of a prediction framework.
This chapter focuses on the first element: the topological representation of the state space. Its characteristics and use are explained in detail in Section 3.1, leading to a representation based on a graph. A simple solution for such a graph is given in Section 3.2 followed by a complex but more suitable method for representing the state space in Section 3.3. Since it is based on clustering observations, different clustering algorithms are examined and finally a recommendation is given in Section 3.5.

3.1 Topological Graph

The main use of a topological model for a prediction task is to provide a suitable sampling of the state space. Since most state spaces are continuous (e.g., the two-dimensional positions of cars and pedestrians), a discretized representation needs to be found which enables a computer to process it efficiently by only predicting the system for a few representative states. Furthermore, the topological model has to encode and reflect all possible transitions of the system. The most evident solution to a topological model with these properties is a topological graph to be described in detail in the following paragraphs.

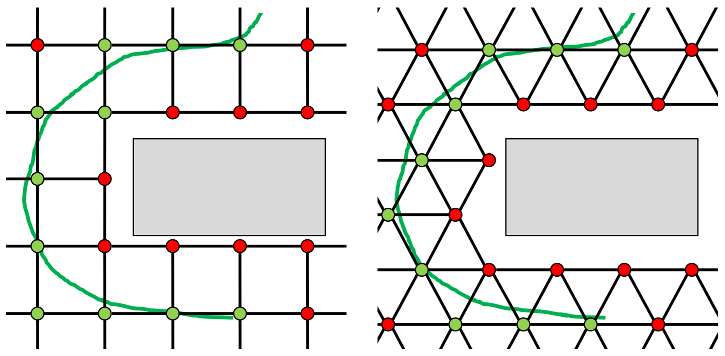
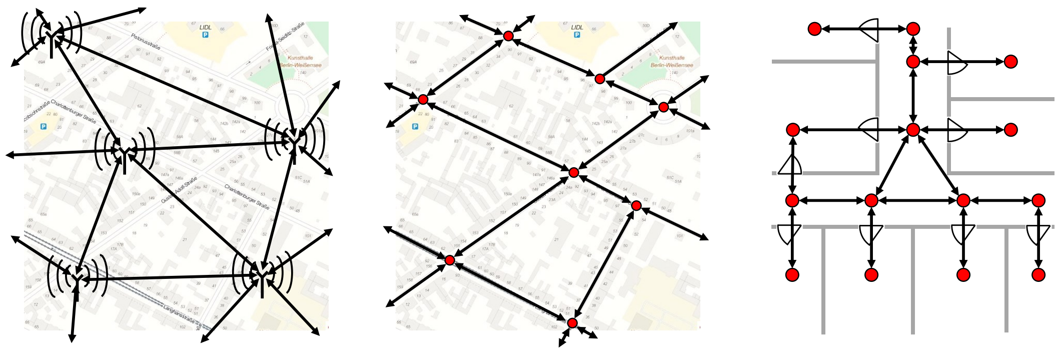

(a) Rectangular grid (b) Sixfold neighborhood

Figure 3.1: *Example of different grid representations. The nodes are depicted as red and green dots, and the connecting edges are shown as black lines. An unreachable region in state space (e.g., an obstacle in the Euclidean space) is plotted as a gray rectangle. It is encoded into the graph by omitting nodes at the corresponding positions. An observed movement trajectory (shown as a green line) can be encoded as a sequence of nodes (colored green). It is apparent in both figures that a sixfold neighborhood provides a better representation of the trajectory in an R^2-space with a lower discretization error.*

For the topological graph, only one obvious assumption is made: instead of randomly alternating between distant states by skipping its intermediate states, the system needs to change continuously, which is the case in most applications. The dimension of the state space does not need to be restricted but, for the sake of simplicity, it is assumed to be a two-dimensional spatial space (e.g., GPS trajectories) for the rest of this thesis. In general, a topological graph $\mathbf{G} = (\mathbf{N}, \mathbf{E})$ consists of nodes $\mathbf{N} = \{\mathbf{n}_i, i = 1, \dots, N\}$ and edges $\mathbf{E} = \{\mathbf{e}_i, i = 1, \dots, E\}$. Each node $\mathbf{n}_i$ represents a discrete state $\mathbf{n}_i = (x_i, y_i)$, and an edge $\mathbf{e}_i = (a, b)$ represents a directed connection from node $\mathbf{n}_a$ to $\mathbf{n}_b$. An observed trajectory can then be represented as a sequence of connected nodes (see Fig. 3.1).

In several applications, a topological graph can be explicitly designed. For predicting the next base station of a user in a mobile network, the layout of the graph is straightforward. The base stations themselves can be encoded as the nodes of the graph, whereas the edges are placed between neighboring base stations (see Fig. 3.2(a)). In the example of traffic prediction, crossroads and turns can be represented by nodes, whereas the connecting roads can be represented by edges (one for each direction in the case of directed edges). Such an example is shown in Fig. 3.2(b). If a coarse spatial resolution is sufficient, a similar approach can be applied for predicting the whereabouts of people in buildings (see Fig. 3.2(c)). But, if more precise information about the position is required or the state space is enormous, modeling the graph by hand may prove impractical and an automated method would be more suitable. Automating the process of graph construction also adds to the versatility of the presented



(a) Wireless network (b) Road net (c) Building

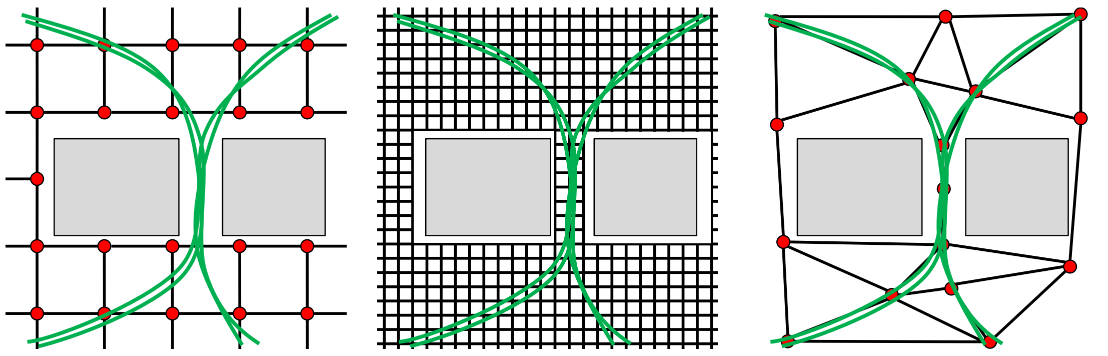
Figure 3.2: *Exemplary two-dimensional topological graphs with the nodes shown as red dots (or as Wi-Fi-icons in Fig. 3.2(a)) and the transitional edges shown as black arrows. In all three examples, the graph can be explicitly constructed due to the knowledge of the state space. It should be noted, that the topology of the graph is strongly related to the problem. If the task is the prediction of the next base station in a mobile network, a graph similar to the one shown in Fig. 3.2(a) would be feasible. It might prove inefficient for predicting car movements, whereas the graph in Fig. 3.2(b) is more suitable. By using directed edges, it is also possible to represent one-way roads. In the case of a person movement prediction in buildings, the floor plan can be used to construct a graph by hand, like in Fig. 3.2(c).*[1]

prediction framework since no prior knowledge of the state space is required. But, it should be noted, that a manual design is not prohibited as long as it is feasible for the problem at hand.

3.2 Grid-based Representation

A basic approach for constructing a topological graph for representing a continuous space is to discretize it by superimposing a regular grid. Each grid cell is covered by a node, and edges are created between nodes that share a border. A rectangular grid, defined by its base length b, is the simplest approach (see Fig. 3.1(a)). A sixfold neighborhood grid adds more complexity in order to minimize the error of the trajectories encoded into the respective topological graph (see Fig. 3.1(b)) and also pays respect to the neurobiological representation of spatial maps [Hafting et al., 2005].
The benefit of such a simple topology is its convenience in construction since the positions of the nodes and their connecting edges are easily computed. It can also deal with obstacles (such as walls or buildings) by omitting the nodes at the corresponding grid cells. Furthermore, the approximation error is guaranteed to be below a certain

[1]The map in the background of Fig. 3.2(a) and Fig. 3.2(b) was taken from openstreetmap.org "© OpenStreetMap contributors"



(a) Big grid with 21 nodes (b) Small grid with 457 nodes (c) Data driven graph with 16 nodes

Figure 3.3: *A comparison of grid-based graphs with different sizes and a topological graph created by a clustering algorithm. The observed trajectories are shown as green lines, and obstacles are shown as gray rectangles. If the chosen grid is too coarse, the transitions through the narrow between the obstacles cannot be modeled as can be seen in Fig. 3.3(a). A smaller grid size can cover the narrow in Fig. 3.3(b) but it overrepresents the regions in which no observations were made. Placing the nodes at observations while sparing empty spaces (as in Fig. 3.3(c)) would provide a memory-efficient and even more precise representation of the state space.*

threshold (for example below $b/\sqrt{2}$ for a rectangular grid with a base length of b). One problem which is accompanied by such a space decomposition is the uniform distribution of nodes over the whole state space: densely covered manifolds of the state are underrepresented with just a few nodes, and sparsely populated areas are overrepresented by many nodes. This sacrifices precision in regions of interest and wastes memory in uninformative parts of the state space. By changing the grid size, one can only mitigate the problem of a precision that is too coarse by increasing the memory needs at the same time or vice versa (see Fig. 3.3(a) and Fig. 3.3(b)). A more suitable discretization of the state space would try to approximate the manifold in which states evolve according to observed data (see Fig. 3.3(c)). Such a topological graph can be found by clustering algorithms as described in the next sections.

3.3 Cluster-based Representation

As shown in Fig. 3.3(c), a topology which considers the observed states should be preferred for a topological graph. Manifolds in the state space in which observations are frequent can be construed as dense regions while regions with only few or no observations can be regarded as sparse. In order to pay respect to the information content, the density of sampling must be proportional to the density in the state space. Such a sampling can be achieved by clustering algorithms, of which a vast variety can be found in literature [Xu et al., 2005]. Almost every algorithm takes a set of points in combination with specific parameters and returns a set of clusters. Since observed states also represent points in state space, the observations can be directly used for input and the resulting cluster-centers can be taken as nodes for the topological graph. Four promising methods for clustering point sets are now contemplated in the following subsections. Although they are not restricted to a two-dimensional space, it helps in grasping the main idea of these methods, if a Euclidean state space (like the positions of people on a floor plan) is assumed.

3.3.1 K-Means Decomposition

K-means [Lloyd, 1982] is a common algorithm for clustering point clouds. Aside from the dataset, it only needs one parameter: the number n of clusters which should be found. On a given point set the algorithm iterates two steps in order to find a suitable partitioning. First, before the iteration starts, the algorithm randomly picks n samples from the point set and defines them as the preliminary cluster centers. In the first iteration step, it assigns every data point to their nearest cluster based on its Euclidean distance. In the second step, it recalculates the center of each cluster as the mean of all assigned data points. Those two steps are repeated until a stop criterion is met (e.g., a maximum number of iterations) or the cluster centers are stable.
The benefits of this algorithm lie in its simplicity and requirement of just one parameter. Unfortunately, those two properties are also its greatest disadvantage. The number of clusters must be known beforehand, which is only given for a few problems. Advanced implementations avoiding this parameter are known [Pelleg et al., 2000] but they do not compensate for the second drawback: it usually finds spherical clusters instead of paying respect to the density distribution of the point set. Furthermore, it is prone to outliers and noise, which often give counterintuitive or even incorrect results.
Due to these drawbacks, the K-means algorithm is not well suited for generating a topological graph. The necessity of knowing the number of clusters threatens the thesis' intention of providing a versatile prediction algorithm. Instead of assuming a spherical partition, a clustering algorithm that considers the density distribution of the given point set should be favored.

3.3.2 DBSCAN Partition

The "Density-Based Spatial Clustering of Applications with Noise" (DBSCAN) [Ester et al., 1996] is a widely used clustering algorithm in data mining. The key idea of this algorithm is that a point in a cluster has to have at least a minimum number *minPts* of neighbors in its neighborhood radius ϵ. These points of a cluster can be found by the concept of "density reachability". A point p is "directly density-reachable" from point q if their distance is below ϵ and if q has more than $minPts - 1$ neighbors in its ϵ-neighborhood. Hence, a point p is density-reachable from q if a chain of directly density-reachable points between p and q exists. By following the tree of density-reachable points from every point of the data set, all of the clusters can then be calculated.

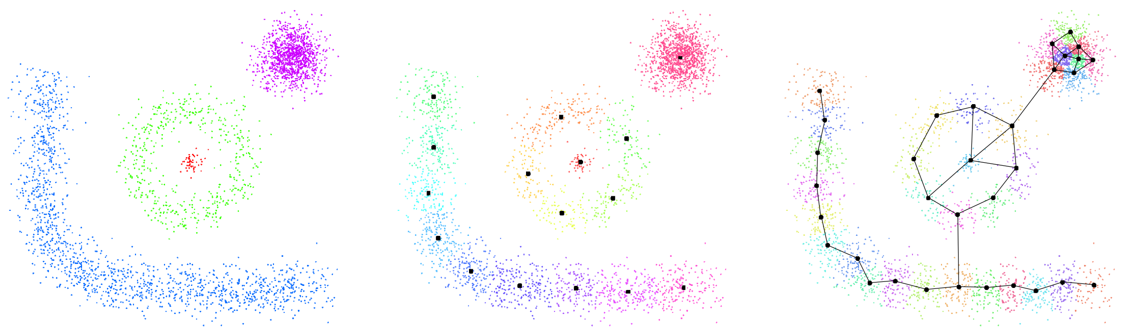
One reason why the algorithm obtains wide acceptance is that it disregards noise and outliers due to the restriction introduced by *minPts*. Furthermore, it is capable of finding concave and elongated clusters (see Fig. 3.4(a)). It is a well-suited clustering algorithm to find objects in point sets (for example cars in radar scans [Kellner et al., 2012]).

At first glance, the algorithm might be opportune for generating a topology graph, but since it puts very elongated point groups into one cluster, it can cover states that are very different from each other (see Fig. 3.4(a)). Furthermore, if two clusters intersect each other, they are merged into one. Considering that the goal of the topological graph is to represent the state space, it is counterproductive to encode a long, continuous series of points into one cluster and, therefore, into one node. Instead, a fragmentation into equally sized clusters would be more expedient.

3.3.3 Growing Neural Gas

In the fields of neural networking, a once common but outmoded clustering algorithm is Growing Neural Gas (GNG) [Fritzke et al., 1995]. It can be seen as an extension to Self-Organizing Map [Kohonen, 1990] and Neural Gas [Martinetz et al., 1991]. Like its predecessors, GNG has a set of nodes which is iteratively modified. On a given input signal, the nearest node and its connected neighbors are moved towards it and, if the signal lies between two unconnected nodes, an edge is placed between them. Edges of nodes which have not been modified for a certain number of nearby inputs are deleted and, if a node is no longer connected to another one, it is removed as well. In order to balance the deletion of nodes, the algorithm also has a mechanism for adding nodes in areas with a high discretization error resulting in an unsupervised adaption of the number of nodes.

The GNG algorithm provides all necessary properties for creating a topological graph, like the ability to adjust the number of nodes and distribution to the density of the input data or the inherent mechanism for connecting neighboring nodes with edges (see Fig. 3.4(c)). Furthermore, its iterative approach enables a life-long learning and an on-line adjustment to a changing topology. Outliers also pose no serious problem due to the deletion of unused edges. The fact that GNG needs at least six parameters in



(a) DBSCAN (b) Mean-Shift (c) Growing Neural Gas

Figure 3.4: *An example of clustering a point set with DBSCAN (Fig. 3.4(a)), Mean-Shift (Fig. 3.4(b)) and GNG (Fig. 3.4(c)). Points with the same color belong to the same cluster. DBSCAN is able to separate all point clouds in an intuitive manner as seen in Fig. 3.4(a). But, in terms of a discrete representation of the state space, such a clustering result is not suitable. For example, the center of the green cluster (and, therefore, the position which refers to the region in state space which is covered by the green point cloud) is almost in the center of the red point cloud and, thus, does not provide a correct representation of the state space. The Mean-Shift algorithm does not provide a correct result in terms of clustering since it splits all contiguous point clouds into smaller clusters (the center of each cluster is shown as a black dot) in Fig. 3.4(b). But, it provides a better partitioning of the state space since widespread point clouds are split up into a sequence of clusters. GNG is able to cluster the pointset according to its underlying density distribution as can be seen in Fig. 3.4(c). The dense pointcloud in the upper right is clustered tightly, whereas the sparse circular pointset is covered by fewer clusters. Due to the mechanics in the GNG algorithm, the clusters are already connected by edges. Unfortunately, it is not guaranteed to provide an extensive and optimal linking (please refer to Section 3.4 for an explanation of 'optimal') as can be seen a number of times in Fig. 3.4(c) (e.g., in the upper right with one missing edge in each of the two four-sided structures or the missing edges between the center, middle and outer point cloud).*

order to work properly is its main disadvantage. Since detailed information about the state space and characteristics of the observations are needed to tune them accurately, GNG may not be the best choice for generating a topology graph in an unknown setting. If the prediction framework presented in this thesis should be tailored to a specific application and the environment is known in advance, GNG is definitely a well suited algorithm for modeling the state space but, for the sake of a versatile prediction framework, another method for generating a topology graph would be more appropriate.

```
Input
  1  $\mathbf{p}_i, i = 1, \ldots, N$                                  //N observations
  2  $G'(x)$          //Derivative of Mean-Shift kernel, e.g., $G'(x) = -x \cdot e^{-\frac{x^2}{2}}$
  3  $\gamma$                                                          //kernel size
Algorithm
  4  $\mathbf{p}'_i = \sum_{j=1}^{N} \mathbf{p}_i \cdot G'\left(\left\|\frac{\mathbf{p}_i - \mathbf{p}_j}{\gamma}\right\|\right) / \sum_{j=1}^{N} G'\left(\left\|\frac{\mathbf{p}_i - \mathbf{p}_j}{\gamma}\right\|\right)$;   //move the points along the gradient
  5  $d = \max_i \|\mathbf{p}_i - \mathbf{p}'_i\|$;                    //find the biggest movement
  6  $\mathbf{p}_i \leftarrow \mathbf{p}'_i$;                          //update the point set
  7  if $d > \frac{\gamma}{100}$, then                //are there still significant movements?
  8    goto 4;
Return
  9  $\mathbf{p}_i, i = 1, \ldots, N$                                  //condensed points
```

Figure 3.5: *Mean-Shift algorithm*

3.3.4 Mean-Shift Clustering

The Mean-Shift clustering algorithm was first introduced in [Fukunaga and Hostetler, 1975] and is based on the idea that each point is a sample of a local density distribution which is represented by a kernel. It aims to find the clusters in the form of local maxima in the overall density distribution by an iterative gradient ascend. A kernel (usually a Gaussian function) is placed at each point in the dataset, and the overall density distribution is formed by the sum of all kernels. The points are then iteratively moved along the gradient, and the density estimation is recalculated. The algorithm is shown in Fig. 3.5.

The final clusters are determined by assigning all of the points that have gathered at the same position to the same cluster.

This algorithm has the benefit of only having one parameter γ (see Fig. 3.5) which determines the width of the kernel function. A wide kernel results in large clusters and a small kernel provides us with clusters of a smaller extent (see Fig. 3.6). It can be used to configure the resolution of the resulting graph and, therefore, the resolution of the final prediction result.

Furthermore, elongated clusters with a roughly uniform density distribution along their dilation are split into smaller clusters, which helps in the task to find a good approximation of the state space (see Fig. 3.4(b)).

It should be noted that this algorithm does not distribute the clusters onto the state space according to the density of the observations like GNG, which would give an optimal sampling of the state space. Nevertheless, it provides us with a representative discretization (a good example can be seen in Fig. 3.6(c)) with an adjustable resolution and no need to configure additional parameters. Unfortunately, as can be seen in

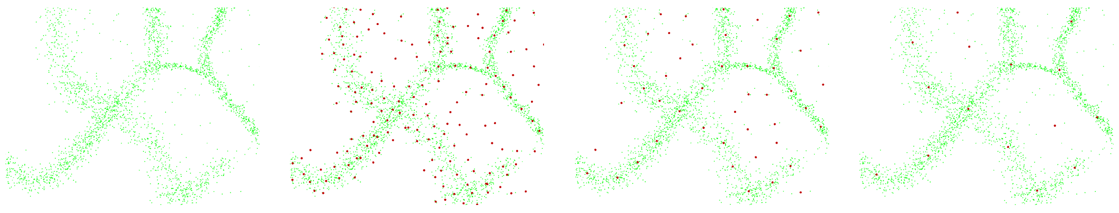


Figure 3.6: *Clustering the point set in Fig. 3.6(a) with Mean-Shift and different kernel sizes. The observed points (sampled from GPS trajectories) are colored green, and the centers of the found clusters are shown as red dots. It is apparent from these figures that Mean-Shift provides a good coverage of areas with a lot of observations. Another important property of Mean-Shift can be seen at regions where frequent trajectories cross each other: due to the higher point density, clusters are placed at these crossings. It improves the quality of the final graph by minimizing the mean discretization error and results in a more intuitive topology.*

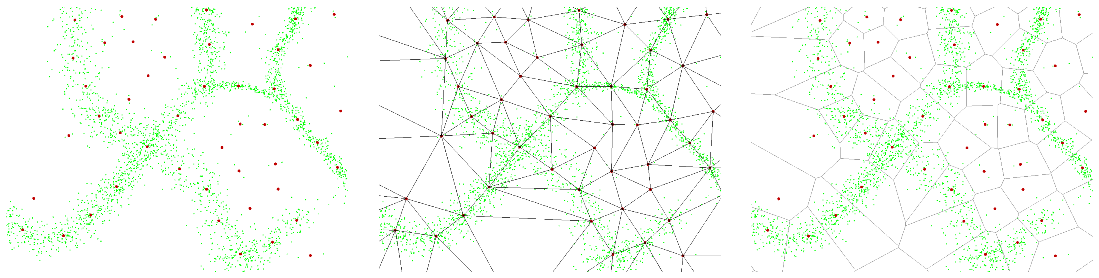
Fig. 3.6, it does not deal with outliers like DBSCAN and GNG but a similar noise reduction can be achieved by omitting nodes which were created by only a small number of observations.

One might argue that K-Means gives similar results and also has only one parameter to choose. However, finding a suitable number of clusters is far more crucial than configuring a proper kernel width. Small changes in the number of clusters often gives entirely different and even incorrect results with K-Means, whereas Mean-Shift provides proper results over a broad range of kernel widths, as shown in Fig. 3.6. Therefore, the Mean-Shift algorithm was chosen to calculate the position of the nodes $\mathbf{N}$ of the topological graph $\mathbf{G} = (\mathbf{N}, \mathbf{E})$. Now the connecting edges $\mathbf{E}$ between neighboring nodes must be calculated in order to complete the graph.

3.4 Edge Creation

The previously mentioned clustering algorithms (except GNG) only provide a node set with no connectivity information. Therefore, the edge creation has to be dealt with separately.

The input data of the clustering algorithm is usually provided in the form of a time series, which would enable us to use the temporal information for associating every pair of successive points in the input data to each other. Since the clustering relates every input point to a cluster (or as in our case to a node), we can use this information to create edges between nodes which are connected by such a pair of associated input points. The advantage of such a procedure would be the direct incorporation of observations into the process of creating edges. Unfortunately, this also means



(a) Clustering result (b) Delaunay triangulation (c) Voronoi regions

Figure 3.7: *Graph creation on a point set taken from GPS trajectories. The points are colored green, and the nodes resulting from the Mean-Shift algorithm are shown as red dots. In order to connect these nodes, a Delaunay triangulation (as shown in Fig. 3.7(b)) is performed. It results in a dual graph to Voronoi parceling, which is shown in Fig. 3.7(c). The Delaunay triangulation connects nodes whose Voronoi regions share a border. It can be seen that the graph is able to cover the main trajectories with its nodes and edges.*

that outliers and noisy data result in a noisy topological graph. Furthermore, such a method strongly relies on a set of trajectories or sequential observations, which are not always given, and the implementation of a versatile prediction framework is thwarted.

Another possible procedure would be to simply connect neighboring nodes to each other. It provides a valid state space representation since states change continuously and do not skip intermediate states (which was the only assumption about the system made in Section 3.1). Therefore, if the state of the system changes, it can only change from the node representing the current state to one of its neighbors. A suitable algorithm for creating edges in such a manner is the Delaunay triangulation [Delaunay, 1934]. It takes a point set and creates a triangle mesh such that no point is in the circumscribed circle of any triangle. Several algorithms for calculating the Delaunay triangulation have been developed [Su and Drysdale, 1995], and the fastest implementation is able to compute the triangulation in $O\,(n\ log\ log\ n)$ [Dwyer, 1987]. An algorithm for performing a triangulation in higher state spaces also exists [Cignoni et al., 1998].

The Delaunay triangulation corresponds to the dual graph of a Voronoi diagram [Aurenhammer, 1991]. The main property of a Voronoi diagram is that it divides the space into regions based on a set of points (called sites) in such a way that every point in a region has the site corresponding to that region as its closest one (see Fig. 3.7(c)). In our case, these sites are the nodes created by a clustering algorithm and their regions are those states in the state space which get assigned to that node based on its Euclidean distance. Since the Delaunay triangulation places edges between nodes which share a border in their Voronoi diagram, it provides the optimal solution for connecting the nodes (see Fig. 3.7(b)) because it is guaranteed that after mapping a continuously changing state onto the graph, it can only change nodes along the connected edges since it can only transit from one region to a neighboring, thus, connected region.

3.5 Conclusion

The sum and substance of Chapter 3 is that a topological graph $\mathbf{G} = (\mathbf{N}, \mathbf{E})$ can be created based on a set of observed states (e.g., sequences of GPS positions) by first determining the nodes $\mathbf{N} = \{\mathbf{n}_i, i = 1, \ldots, N\}$ with the Mean-Shift algorithm. Afterward, they can be connected with edges $\mathbf{E} = \{\mathbf{e}_i, i = 1, \ldots, E\}$ by applying a Delaunay triangulation onto these nodes. The resulting graph can be used to encode states in the continuous space into a discretized representation which pays respect to observations and their coverage in the state space. The graph can be created with Mean-Shift by providing observed states and only one parameter γ (see Fig. 3.5) which configures the resolution of the topological graph. Unfortunately, the graph creation can only be done batch-wise (but that only poses a problem, if the environment is not static).
If the topology changes or if information about the environment is available to provide the necessary parameters, Growing Neural Gas is also a suitable method for representing the topology. It has the advantage of placing the nodes according to the underlying density distribution and enables the implementation of an adaptive, iterative, and life-long learning. Although it connects the nodes with edges, they do not provide an optimal linking as can be seen in Fig. 3.4(c). Therefore, the additional step of reconnecting the nodes as described in Section 3.4 is advised. The performance of GNG is dependent on six parameters and their adjustment is only feasible if the characteristics of the state space and the observed trajectories are previously assessable, thus making GNG the second method of choice.
A topological graph without any additional information can only provide spatial information about the system and encoded representations of spatial observations. In order to predict trajectories, not only is a suitable representation of the state space required but conditional probabilities of transitions between states are also vital. The next chapter shows an elegant way of including these probabilities into the topological graph.

Chapter 4

Representation of Transitional Probabilities of Moving Objects

The last chapter presented a method for creating a topological representation of the state space based on the Mean-Shift algorithm and a Delaunay triangulation. It provides us with a spatial description of observations better suited for processing than a simple sequence of states.

Since this thesis claims to provide a comprehensive statement about all possible future occurrences of a given observation, the entire event tree emerging from a current observation needs to be taken into account. Therefor, the use of a prediction method similar to wavefront based algorithms [Lengyel et al., 1990] would be appropriate. The main prediction method will be explained in Chapter 5 but several basic elements enriching the topological representation with probabilistic information need to be described beforehand. First, the transitional probability is introduced in Section 4.1 in order to encode observations into the topological map and to enable a calculation of spatial probabilities. To provide information about the prior route of a flow, the Markov-tree is presented in Section 4.2. Both elements do not take temporal information into account so transition times between the nodes of the topological graph must be dealt with separately, as described in Section 4.3.

All three elements can then be used together with the topological graph to predict the future states of a system by providing the probability $o_t(n)$ for every node n and every time step t, denoting that the system is in the respective state at the respective time step. The character o was chosen as the variable for this special probability instead of the usual p for two main reasons: first, to provide a visual delimitation to the transitional probabilities in intermediate calculations and to emphasize it as the main result of the prediction framework - the observational probability. The second reason is due to the initial application of the prediction framework for predicting human movements at airports, as mentioned in Section 1.2. The goal was to estimate future

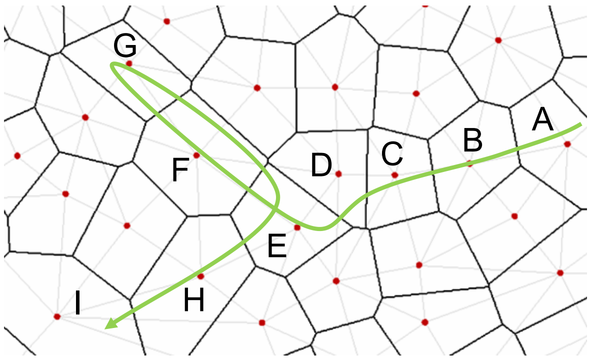


Figure 4.1: *Representing a trajectory (green line) as a sequence of nodes (red dots) by simply collecting the indices of nodes whose Voronoi regions are traversed by the trajectory. The sequence can be obtained by determining the nearest node for each data point of the trajectory. Each time the nearest node (e.g., A) changes to another one (B), the index of the last node (A) is stored in the sequence. In case of noisy data, the application of a smoothing algorithm on both the input trajectory and the output sequence can help in improving the result.*
The depicted trajectory would be encoded as A-B-C-D-E-F-G-F-E-H-I.

<u>o</u>ccurences of a person of interest or the probability that the person is <u>o</u>ccupying a certain area; hence, the $o_t(n)$. It helps in understanding the following chapter to use the example of predicting the movement of a person on a two-dimensional floor plan but, as stated several times in the previous chapters, the presented solutions are not restricted in their application to human motions or movements in spatial state spaces. They can also be used to predict most other systems as well (see Section 1.2 and 3.1).

4.1 Transitional Probabilities

Since the nodes $\mathbf{N} = \{\mathbf{n}_i, i = 1, \ldots, N\}$ of the topological graph represent discretized regions in the state space, an observed sequence of states $\mathbf{S} = \{(x_i, y_i), i = 1, \ldots, S\}$ of a continuous trajectory through the state space can be encoded as a sequence of nodes $\mathbf{N_S} = \{\mathbf{n}_{i_s}, s = 1, \ldots, N_S\}$ (see Fig. 4.1). Due to the edge creation by a Delauny triangulation, it is ensured that every node in $\mathbf{N_S}$ is connected to the successive node by an edge (as explained in Section 3.4). By representing every trajectory as a sequence of nodes, statistics of the transitions from one node to another node can be calculated in a learning phase and updated in the application phase.
In the simplest approach, the statistics for every node $\mathbf{n}_n$ may consist of a counter $f_{n,m}$ for every neighboring node $\mathbf{n}_m$. Every time a transition from node m to n was observed, the counter $f_{n,m}$ is incremented. After obtaining enough observations, the probability to move to node n under the condition that node m was currently observed can be

approximated by the relative frequency $p\,(n|m) \approx f_{n,m}/f_m$ with $f_m = \sum_n f_{n,m}$ being the total number of observed transitions from node m to any other node ([Von Mises, 1928]).

Using such an approach for calculating the transition probabilities, the next state only depends on the current state resulting in a first order Markov chain [Kemeny and Snell, 1960]. The prediction using this approach can be conducted by setting the probability for the node of the last known observation to one and letting it flow through the graph based on the previously calculated probabilities like a wavefront.

A computationally less demanding method is the application of a Monte Carlo method (introduced in [Metropolis and Ulam, 1949]) to obtain the probabilities for each node: a set of H hypotheses is placed at the last known node and each hypothesis traverses the graph by choosing the next node from the current node m randomly with respect to $p\,(n|m)$. For each node n, a counter h_n is incremented if it was visited by a hypothesis at least once. The final probability for each node can then be approximated by the relative frequency which is obtained by just dividing its counter with the total number of hypotheses $o\,(n) \approx h_n/H$.

Using a first order Markov chain provides an easy approach since no information about the previous path needs to be stored. For example, if a person is walking along a corridor and reaches a crossing, it is only important to know its current node in the topological graph in order to estimate the subsequent node. Clearly, the lack of memory is also a severe downside to this approach since it yields more imprecise results than the consideration of the previous path the person has taken. For example, if half of the observed persons at a crossing have taken the route to the exit, one quarter has taken the turn to the escalator, and the remaining quarter has taken the route to the staircase, the same distribution would be the result of the motion prediction for every person approaching this crossing. Even if the person was coming from the exit, the first order Markov approach would estimate that the person will go back to the exit with a probability of 50%. If the transitional probability also pays respect to the previous node, a second order Markov chain is utilized. The first order probability $p\,(n|m)$ becomes $p\,(n|m,l) \approx f_{n,m,l}/f_{m,l}$ with l being the index of the node observed previous to m. If a first order Markov chain is depicted as knowing the current position, the second order Markov chain can be seen as knowing the current position and momentum.

In the previous example, only observations coming from the exit to the crossing would be used for the calculation and the probabilities for the escalator and staircase presumably would be higher than the probability that the person reverts to the exit. Following this thought leads to using the complete previous sequence of visited nodes $\mathbf{c} = \{c_1, \ldots, c_j\}$ as the condition for the transitional probabilities $p\,(n|\mathbf{c})$. Unfortunately, the combinations for $\mathbf{c}$ (and, therefore, the number of counters for each node)

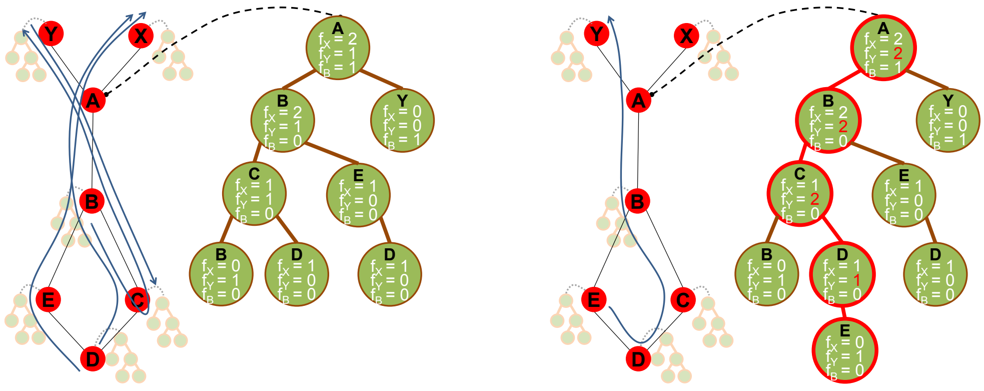
```
Input
    $\mathbf{c} = \{n, m, c_1, \ldots, c_{k-1}\}$          //Observed sequence from current node $m$ to node $n$
    $k$                                                    //Length of observed node sequence up to node $m$
    $v_m$                                                  //Root vertex of Markov-tree $M_m$ for node $m$
Algorithm
    $v_x = v_m$;                                           //Set vertex for recursion
    $l = 0$;                                               //Initialize level for recursion
Label 1:                                                                      recursion
    increment counter $f_n$ of vertex $v_x$;
    if $l >= (k-1)$, then
      terminate;
    find child $v_c$ of $v_x$ with index $c_{l+1}$;
    if no child found, then
      create child $v_c$ with index $c_{l+1}$;
    set $v_x$ to $v_c$;
    increment $l$;
    goto Label 1;
```

Figure 4.2: *Updating the Markov-tree on a given observation*

are infinite if no restrictions to its length are given. But, even if the length is restricted to a number l (resulting in an l-th order Markov chain), the complexity may be too high to provide accurate estimations for the transitional probabilities with a given set of observations. For instance, if $\mathbf{c}$ is restricted to the last ten nodes and every node has four neighbors, at least four million ($4 \cdot 4^{10}$) observations are needed for every node to cover each possible condition for each transition at least once. But, in order to estimate transitional probabilities, more than one sample is needed for each condition. It becomes clear that a Markov chain of a high order requires an enormous database and a Markov chain of a low order may waste useful information. A solution which provides a dynamic trade-off between available information and accuracy will be presented in the next section.

4.2 Markov-Tree

As stated in the previous section, the order of the Markov chain should be chosen neither too high, nor too low. It would also be a waste of memory if, for example, a 10-th order Markov chain is assumed and the necessary counters are stored for each node but no observation has covered more than 5 nodes. The Markov-tree, a novel method using a tree-like data structure to store observations and to calculate the transitional probabilities for the nodes, was developed for this thesis in order to deal with such problems.


Y
X
A
B
E
C
D
A $f_X = 2$ $f_Y = 1$ $f_B = 1$
B $f_X = 2$ $f_Y = 1$ $f_B = 0$
Y $f_X = 0$ $f_Y = 0$ $f_B = 1$
C $f_X = 1$ $f_Y = 1$ $f_B = 0$
E $f_X = 1$ $f_Y = 0$ $f_B = 0$
B $f_X = 0$ $f_Y = 1$ $f_B = 0$
D $f_X = 1$ $f_Y = 0$ $f_B = 0$
D $f_X = 1$ $f_Y = 0$ $f_B = 0$
Y
X
A
B
E
C
D
A $f_X = 2$ $f_Y = 2$ $f_B = 1$
B $f_X = 2$ $f_Y = 2$ $f_B = 0$
Y $f_X = 0$ $f_Y = 0$ $f_B = 1$
C $f_X = 1$ $f_Y = 2$ $f_B = 0$
E $f_X = 1$ $f_Y = 0$ $f_B = 0$
B $f_X = 0$ $f_Y = 1$ $f_B = 0$
D $f_X = 1$ $f_Y = 1$ $f_B = 0$
D $f_X = 1$ $f_Y = 0$ $f_B = 0$
E $f_X = 0$ $f_Y = 1$ $f_B = 0$

(a) Previous observations (b) New observation

Figure 4.3: *Illustration of a Markov-tree for an exemplary graph and observations. The seven nodes of the topological graph are shown as red circles with black edges between them, and observed trajectories are depicted as blue lines with an arrow representing their direction. The Markov-tree of node A is shown in (a) after enriching it with four observations with the algorithm in Fig. 4.2. The vertices of the Markov-tree are depicted as green circles and connected by brown lines. They are labeled with their index as a black capital letter (e.g., the root vertex has the index A since it represents the Markov-tree for node A) and their individual counter f_X, f_Y, and f_B for the number of observed transitions to the neighboring nodes X, Y, and B in white. In (b), a fifth observation (the single blue path going from node E to D, C, B, A, and finally Y) is inserted into the tree of node A. The descent through the tree and changes in the counter variables (i.e., f_Y since the observed trajectory headed to node Y after visiting node A) are highlighted in red. The algorithm for updating the Markov-tree with new observations is described in Section 4.2 and Fig. 4.2 in more detail.*

The basic idea of the Markov-tree is that every possible sequence of previously visited nodes (i.e., the reachability) can be encoded into a tree structure. The Markov-tree M_m consists of vertices which correspond to nodes of the topological graph according to their index c_l. Each vertex of the tree branches into vertices representing the neighbors of the node to which it corresponds. The root vertex of the tree of a node m has the index $c_l = m$. On the example of the graph in Fig. 4.3, node $m = A$ can only be directly reached from node B, Y and X. Thus, its Markov-tree has the root vertex with index A which branches into the vertices B, Y, and X. These branches are also ramifying to the neighbors of their corresponding nodes (e.g., A, C, and E for the vertex of node B or just A for the vertex of node Y). Each vertex of the Markov-tree of node m does not only contain the index c_l of its corresponding node but also a counter f_n for every direct neighbor n of node m. As depicted as pseudocode in Fig. 4.2, information of a given observation $\mathbf{c} = \{n, m, c_1, \ldots, c_{k-1}\}$ can be encoded into the Markov-tree of node m by incrementing the counter f_n of all touched vertices while descending into the tree by matching the indices to the visited nodes: starting at the root vertex m, continuing to the next vertex c_1, and ending after $k - 1$ branches in the vertex with the label c_{k-1}. Using the example in Fig. 4.3(b), the observation went from node A to node Y after passing the nodes B, C, D, and E. Thus, the Markov-tree of node A gets traversed along the vertices of node A, B, C, D, and finally E with the counter f_Y incremented by one on every vertex to incorporate the information that the observation headed towards node Y, after visiting the current node A. In order to save memory, the branches of an initially empty Markov-tree are only created if a corresponding observation is made. It is unfeasible to initialize the tree to its full extend since it might be of infinite size if the topological graph contains loops. Furthermore, vertices with all counter f_n set to zero and their branches do not add any useful information to the tree.

By learning the Markov-trees of all nodes of the topological graph, it becomes easy to estimate the transitional probabilities for an arbitrary observation $\mathbf{c} = \{n, m, c_1, \ldots, c_j\}$ arriving at a node m following the algorithm as shown in Fig. 4.4 as pseudocode. The tree is traversed, starting at the root vertex, by descending into the branch with the index c_1, followed by the vertex with the index c_2, and so on until either no further branch is available or the vertex for the last index c_j is reached. Once the final vertex in the tree M_m is ascertained, its counters f_i can be used to estimate the transitional probability from node m to every neighboring node n as the relative frequency $f_n / \sum_i f_i$.

A good assessment for the quality of the probability estimation is given by Chebyshev's inequality [Chebyshev, 1867], shown in Eq. 4.1.

$$P\left[\left|\frac{f_n}{\sum_i f_i} - p\right| < \epsilon\right] \geq 1 - \frac{1}{4s\epsilon^2} \tag{4.1}$$

For a random experiment, it states that probability P, that the relative frequency $f_n / \sum_i f_i$ is within the margin ϵ around the expectation p, is bigger than the right hand side of Eq. 4.1. If for example $s = 100$ observations contributed to the current vertex of the Markov-tree of a node, we can assume that the estimation of the transitional probability is within a margin of $\epsilon = 10\%$ with a probability greater than 75% ($0.75 = 1 - 1/(4 \cdot 100 \cdot 0.1^2)$).

Eq. 4.1 can be rewritten as Eq. 4.2 to provide the minimal number s of samples needed to be sure that the estimated transitional probability is within an error margin of ϵ with the probability P.

$$s \geq \frac{1}{4(1-P)\epsilon^2} \tag{4.2}$$

Since the leaf vertices of a Markov-tree tend to represent only a few samples, Eq. 4.2 can be used to define a further stop criterion for traversing the tree based on a current observation (see line 13 to 15 in Fig. 4.4). Once a minimal number of s samples is

```
Input
 1   $\mathbf{c} = \{m, c_1, \ldots, c_{k-1}\}$            //Observed sequence to current node $m$
 2   $k$                           //Length of observed node sequence up to node $m$
 3   $v_m$                                //Root vertex of Markov-tree for node $m$
 4   $n$               //Target node for which the transitional prob. should be calculated
 5   $s$                     //Minimal desired number of samples according to Eq. 4.2
Algorithm
 6   $v_x = v_m$;                                          //Set vertex for recursion
 7   $l = 0$;                                       //Initialize level for recursion
Label 1:                                                                 recursion
 8   if $l >= (k-1)$, then
 9     return;
10   find child $v_c$ of $v_x$ with index $c_{l+1}$;
11   if no child found, then
12     return;
13   set $a$ to the sum $\sum_i f_i$ of all counter of $v_c$;
14   if $a < s$, then
15     return;
16   set $v_x$ to $v_c$;
17   increment $l$;
18   goto Label 1;
Return
19   $p(n|\mathbf{c}) = f_n / \sum_i f_i$ using the counter of vertex $v_x$
```

Figure 4.4: *Calculating the transitional probability on a given observation*

chosen (e.g., $s = 100$ would provide a 75% security to be within a 10% margin of the real transitional probability and $s = 1000$ gives a 97.5% security), the last vertex which provides enough samples (i.e., $\sum_i f_i \geq s$) is used to estimate the transitional probability as stated above.

Using an individual Markov-tree for every node of the topological graph provides an easy and adaptive mechanism for encoding observations and calculating transitional probabilities. Its accuracy can be assessed with Eq. 4.1 and, in order to improve the accuracy, the Markov-tree enables an on-line updating to incorporate new observations. In Chapter 4, it was claimed that the prediction framework will provide the probability distribution $o_t(n)$ of the system for every node n and every time step t. With the topological graph and the transitional probabilities, it is possible to provide the probability distribution for every node but without temporal information. In order to incorporate the time domain into the prediction result, the transitional time between every node needs to be taken into account, too. The following section will present two possible methods of extending the Markov-trees with the temporal information of observations.

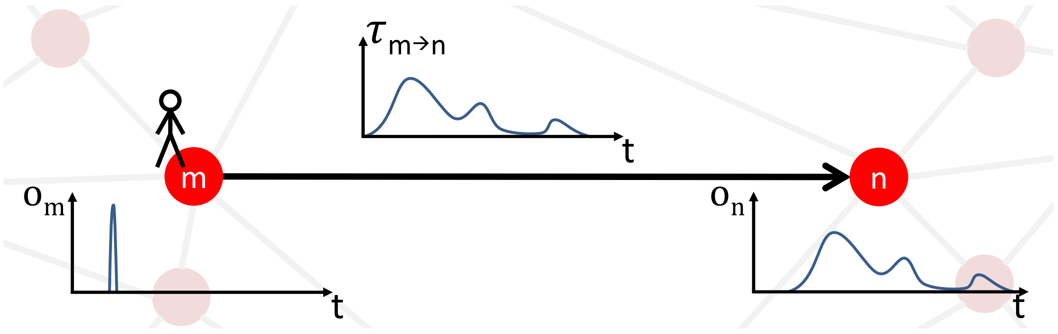
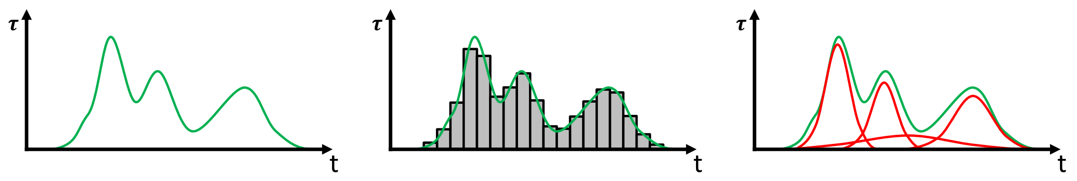


Figure 4.5: *The temporal distribution τ is used to compute the observational probability o_n for node n, based on the probability o_m of node m. In the depicted example, the person was directly observed at node m and, thus, the corresponding observational probability is a single peak at the respective point in time. In combination with the given distribution of the transitional time, an observational probability at node n can be computed. It is apparent that it has a shape similar to the transitional time but with an offset. As will be explained in Chapter 5, it results from the convolution of o_m and τ.*

4.3 Representation of Time

Until now, only spatial information was used to construct the prediction framework. But, in order to predict when a system might be in a specific state, when a person may be observed at a certain location, or where a car can be found at a specific time in the future, the temporal information of previous observations needs to be taken into account.

In the process of computing the sequence of visited nodes of an observation, as described in Section 4.1, the transitional time between each pair of nodes can also be extracted easily. The sequence of nodes $\mathbf{N_S} = \{\mathbf{n}_{i_s}, s = 1, \ldots, N_S\}$ becomes a sequence of tuples $\mathbf{N_S} = \{(\mathbf{n}_{i_s}, t_s), s = 1, \ldots, N_S\}$ with t_s being the transitional time from node $\mathbf{n}_{i_s}$ to $\mathbf{n}_{i_{s+1}}$. While learning the transitional probabilities for every node m by updating its Markov-tree M_m as explained in the previous section, the transitional time also needs to be stored in the respective vertices of the tree. Similar to the counter f_n of a vertex, additional data is stored in each vertex. It contains the temporal information $\tau_t(n|m)$ resembling a probability distribution over time, giving for each transitional time t a probability τ that such duration is needed to get from node m to node n (see Fig. 4.5). Two options for representing the temporal distribution are described in detail in the following sections.



(a) Original distribution (b) Histogram-based (c) Kernel-based

Figure 4.6: *Two methods for representing a temporal distribution, as shown in (a). The non-parametric approximation depicted in (b) has an inherent discretization error, which can only be countered by using bins with a smaller width. In the shown example,* 22 *values (bin width and* 21 *bins, starting from* $t = 0$*) need to be stored in order to represent the underlying distribution. The non-parametric distribution can easily be computed by counting observations. By approximating the distribution with a sum of Gaussian kernels as shown in (c), a smaller error can be achieved and only* 12 *values need to be stored (four times the mean, variance and height). Unfortunately, the computation of these parameters are more complex.*

4.3.1 Non-Parametric Kernel-based Distribution

A simple method to represent a temporal distribution on a computer is to discretize the time scale into a data array with bins of a fixed width w and map the probabilities onto these bins (see Fig. 4.6(b)) like a histogram. Each bin $\hat{\tau}_i\,(n|m)$ with $i = \lfloor t/w \rfloor$ represents a counter for the occurrences of the respective transitional time (or time interval to be precise) and the probability for a given time t can be computed as the relative frequency $\tau_t\,(n|m) \approx \hat{\tau}_{\lfloor t/w \rfloor}\,(n|m) \,/ \sum_i \hat{\tau}_i\,(n|m)$. Since observations are usually obtained by sampling with a fixed frequency and therefore discretized in time, it is advisable to set the bin width to the sampling rate (or to a multitude for cases with an extremely high sampling rate). This way, the number s of observed samples of the current observation between the nodes m and n can directly be used as the index for determining the corresponding bin $i = s$ in the data array. For each new observation, the transitional time distribution is updated by simply incrementing the corresponding bin, enabling a computationally inexpensive and lifelong learning.

In the case of only a few available observations and long transitional durations, the data array may be sparse. If, for example, a bin width of $w = 1s$ was chosen and only two observations with $t = 9.2s$ and $t = 999.8s$ are stored in the array, a total of $1,000$ bins are needed and only two have a value bigger than zero. It also implies that the transitional time between $t = 9.0s$ and $t = 10.0s$ (the time interval the tenth bin represents) has a probability of 50%, whereas a transitional time of $t = 10.1s$ is never expected to occur. That would be a rather bold assumption considering that only a small number of samples were used to estimate the distribution. This problem can be mitigated by not incrementing the value of the respective bin but to diffuse the observation according to a kernel function. This procedure resembles a non-parametric

Kernel Density Estimation (KDE) [Silverman, 1986]. One popular option for a kernel is the Gaussian function but the binomial distribution (shown in Eq. 4.3 with k being the kernel width and r the index of the center bin of the kernel) is better suited for a discretized representation.

$$\Delta\hat{\tau}_{r-k+i} = \binom{2k}{i} \cdot 0.5^{2k} \qquad i = 0, \cdots, 2k \tag{4.3}$$

Since the binomial coefficient is expensive in calculation, the iterative version in Eq. 4.4 should be favored.

$$\begin{aligned} \Delta\hat{\tau}_{r-k} &= 0.5^{2k} && \text{as initialization} \\ \Delta\hat{\tau}_{r-k+i} &= \Delta\hat{\tau}_{r-k+i-1} \cdot \frac{2k-i+1}{i} && i = 1, \cdots, 2k \end{aligned} \tag{4.4}$$

The choice of the kernel width k depends on the underlying distribution. If it is chosen too wide, a multimodal distribution may only be represented as unimodal. If it is chosen too narrow and only a few observations are available, the data array will be sparse. In order to avoid a manual choice, a suitable width can be estimated based on a set of observations. According to Silverman's rule of thumb [Silverman, 1986], the optimal kernel width k_g for a Gaussian kernel is

$$k_g = \left(\frac{4}{3}\right)^{\frac{1}{5}} \sigma_s s^{-\frac{1}{5}} \approx 1.06 \cdot \sigma_s s^{-\frac{1}{5}}. \tag{4.5}$$

with σ_s being the standard deviation of the samples and s the total number of samples. Since the width k_g of a Gaussian kernel is defined as its standard deviation σ_g and the binomial distribution converges to the normal distribution for large widths k with $k_g = \sigma_g = 0.25 \cdot k \cdot w$ and mean $\mu = 0.5 \cdot k$, we can use Eq. 4.5 to calculate the optimal width k of the binomial kernel with Eq. 4.6.

$$k = \left\lfloor \alpha \cdot 4 \cdot \frac{k_g}{w} \right\rfloor = \left\lfloor \alpha \cdot \frac{4.24 \cdot \sigma_s}{\sqrt[5]{s} \cdot w} \right\rfloor \tag{4.6}$$

The parameter α can be used to stretch or compress the kernel in case of additional knowledge about the transitional time (e.g., if it is almost constant, α can be set close to zero) but a value of $\alpha = 1$ has proven to be convenient.
After learning the distribution with the initial set of observations and the above mentioned binomial kernel, it can be iteratively improved with new observations.

4.3.2 Parametric Kernel-based Distribution

A similar approach to the non-parametric Kernel Density Estimation, as explained in Section 4.3.1, is to approximate the underlying distribution with a set of functions

in their parametric form. Extensive research was conducted in the fields of computer vision where a parametric representation of a color distribution is often needed for background subtraction [Piccardi, 2004]. In order to decide if a pixel belongs to background or foreground, its color and intensity is compared to a previously learned color distribution of the background. Different lighting situations, shadows, and reflections usually result in distributions with more than one mode, which need to be reproduced adequately by the approximation. Non-parametric representations (e.g., multidimensional histograms) are not feasible since a distribution needs to be learned and stored for each pixel resulting in extensive memory requirements. Several different methods for solving the background subtraction with a parametric representation exist like Gaussian Mixture Models (GMM) [Zivkovic, 2004] and Kernel Density Approximation (KDA) [Han et al., 2004]. GMM usually uses a fixed number of Gaussians or needs a sensitive adjustment of parameters, whereas KDA is able to cope with more complex distributions and adjusts the number of kernels dynamically.
The method presented in [Han et al., 2004] takes an initial set of samples and applies a variable bandwidth Mean-Shift algorithm [Comaniciu et al., 2001] to find the main modes of the underlying distribution. At each mode, a Gaussian kernel is placed and its covariance is computed by curvature fitting using the Hessian matrix at that mode. Afterwards, new observations are used to sequentially update the set of kernels on-line by applying the variable bandwidth Mean-Shift to them plus an additional Gaussian kernel representing the new observation. If multiple Gaussians are distributing to the same mode, they are merged into one kernel and the covariance is recomputed. As stated in [Han et al., 2004], KDA is robust in finding the modes of the distribution with only few memory requirements but it yields a higher approximation error than the non-parametric Kernel Density Estimation.

4.3.3 Discussion

The main advantage of representing a temporal distribution with a parametric method (like GMM or KDA) over a non-parametric method (like KDE) lies in its modest memory requirements (see Fig. 4.6). Despite the extensive computations needed while learning the distribution, GMM and KDA enable a faster prediction since the prediction framework relies on multiple convolutions of temporal distributions (as will be explained in Chapter 5), which are much faster to compute on two small sets of Gaussian functions than on two long data arrays. But, since the main memory and the computing power of nowadays personal computers (as of 2015) are more than sufficient for processing the non-parametric representation even for large-scale state spaces (e.g., the Beijing scenario in Chapter 6), the non-parametric KDE is recommended for representing the temporal probability distribution $\tau_t(n|m)$. As stated in [Han et al., 2004] and implied in [Piccardi, 2004], GMM and KDA follow the assumption that the underlying distribution consists of well-separated Gaussians. Therefore, they are tuned to accurately model the main modes of the distribution while paying less attention to minimizing the overall approximation error, which is reasonable for background subtraction or object tracking. But, unfortunately, such an assumption may not be true for a distribution of transitional times. Additionally, they may resemble functions which are not easily approximated by Gaussians, such as uniform, χ^2, or more complex distributions.

The transitional time distribution was stated as $\tau_t(n|m)$ denoting that the distribution is only dependent on the current node m and the target node n, disregarding any history. It would assign one distribution to every edge. But, following the same reasoning as in Section 4.1, it is advantageous to account for all the previously visited nodes $\mathbf{c}$, resulting in $\tau_t(n|\mathbf{c})$. It enables concepts like momentum, which makes the the transitional time (or speed) dependent on the previously taken path. For example, a car at a crossing will most likely need less time to drive across it than to take a turn left for which it needs to decelerate first. Also, influences of traffic lights or preference roads can be taken into account intrinsically. The distribution for every variant of $\mathbf{c}$ can be stored into the respective Markov-tree using the same storing and updating mechanisms of the transitional probabilities $p(n|\mathbf{c})$.

But in case of limited memory, it is advisable to restrict the transitional time distribution $\tau_t(n|\mathbf{c})$ for each pair of nodes to a l-th order Markov chain (e.g., with $l = 2$) by only taking the last l observed nodes $\mathbf{c} = \{c_1, \ldots, c_l\}$ into account. This would result in a Markov-tree having a distribution only stored in the vertices up to a level of $l - 1$ instead of every vertex. Only if the memory requirements are still too high, KDA may provide a reasonable alternative to the non-parametric representation of a temporal distribution as the last resort.

4.4 Conclusion

This chapter presented a novel procedure to enrich a topological graph with transitional probabilities and temporal information. The transitional probabilities for each node are extracted from observations by storing the relative frequencies of transitions to neighboring nodes into structures specially designed for this task and introduced as Markov-trees. They enable a lifelong learning and account for the complete history of every observation with very few memory requirements. Since the transitional probabilities are estimates based on a finite number of observations, a method to assess and tune the confidence of the estimation based on Chebyshev's inequality was introduced. For instance, if a high confidence is desired, more observations with a shorter history

are used for calculation resulting in a more diffuse prediction result. If a small confidence is allowed, the few observations with a longer history are used and a spatially more distinct but statistically less certain result can be given.
In order to provide temporal information, the Markov-trees are extended by transitional time distributions for each pair of nodes. A non-parametric representation of the distribution, similar to KDE, was presented and compared to parametric representations like GMM and KDA. The non-parametric version is recommended because it does not make any assumptions about the underlying distribution and no parameters need to be tuned by hand. After an initial learning phase, the parameters can be updated iteratively with new observations. In order to save memory, the calculation of transitional time distributions can be limited to the topmost vertices of the Markov-tree by only accounting for the last few nodes of observed trajectories.
A representation of the state space was introduced in Chapter 3, and a method for calculating the probabilities of transitions between states was defined in Section 4.1. The time needed to transit from one state to another can be estimated, as explained in Section 4.3 and all of this information can be easily stored and accessed by using Markov-trees, as introduced in Section 4.2. The following chapter will present the main prediction framework, which processes all of the transitional probabilities and transitional time distributions to provide a spatio-temporal probability distribution for a given observation.

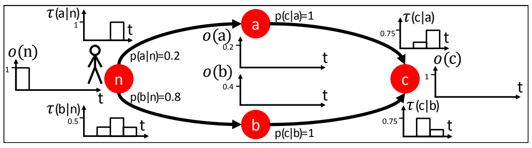
Chapter 5

Prediction Framework

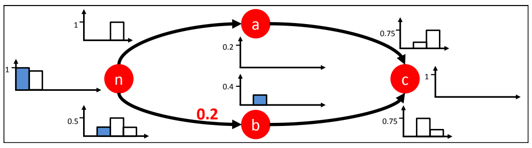
Three main concepts were presented in Chapters 1 through 4. First, the state space is discretized with a graph $\mathbf{G} = (\mathbf{N}, \mathbf{E})$ in order to represent it adequately for further processing. Observed trajectories were then used to enhance the graph with conditional transitional probabilities $p\left(n|\mathbf{c}\right)$ between each ordered pair of connected nodes (n, c_1), encoding the spatial information of observations. Furthermore, the same observed trajectories contribute to conditional transitional time distributions $\tau_t\left(n|\mathbf{c}\right)$, which are linked to the respective transitional probabilities by their condition $\mathbf{c}$. Whereas the transitional probability states the likeliness of a transition to node n under the condition $\mathbf{c}$, the transitional time distribution describes the time needed for such a transit.

This chapter will present an algorithm for processing the learned transitional probabilities and transitional time distributions into a spatio-temporal probability distribution for a given observation of an object. For every state represented by the topological graph and for each time step, it provides the probability for observing the object at that specific spatio-temporal point.

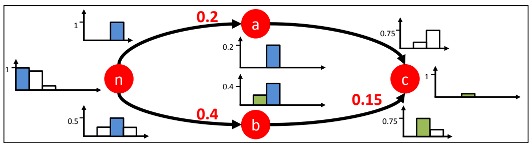
First, the motivation and central idea of the prediction algorithm will be presented in Section 5.1. Its formal definition for a continuous timescale will be given in Section 5.1.1 followed by a practical implementation in Section 5.1.2. The most time consuming part of the algorithm is the computation of multiple convolutions. In order to circumvent this complex operation, an alternative and much faster implementation using the frequency domain instead of the time domain is described in Section 5.1.3. A more effective implementation, using an iterative approach on a discretized timescale, is presented in Section 5.1.4. The proposed procedure to incorporate new data into the topological and probabilistic model is given in Section 5.2.



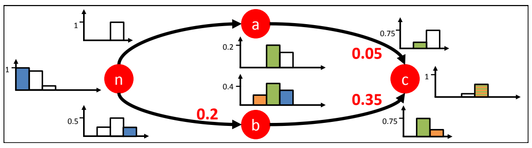
(a) t=0



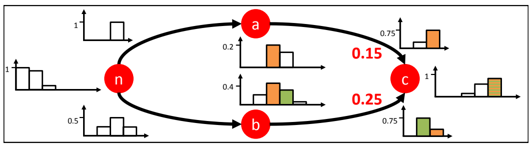
(b) t=1



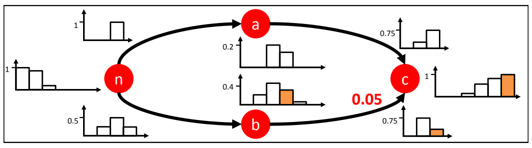
(c) t=2



(d) t=3



(e) t=4



(f) t=5

Figure 5.1: *Example showing the flow of the occupancies on a simple graph. In (a), the involved parameters and the initial state are depicted with a person observed at node n. In (b) to (f), the flow along the edges is shown in red and the bins contributing to them are colored blue, green, and orange according to the individual fractions involved in the current time step (i.e., the flow caused by a τ colored blue involves occupancies which are also colored blue). The occupancies o at time step t are resulting from the previous occupancies at time step $t-1$ plus the the inflow minus the outflow along the edges. A more detailed description is given in Section 5.1, and the method used for calculating the flows is presented in Section 5.1.4*

5.1 Probabilistic Flow

Inspired by several path planning algorithms (like Dijkstra's algorithm [Dijkstra, 1959], A* [Hart et al., 1968], D* [Stentz, 1994], E* [Philippsen, 2004] and many more [LaValle, 2006]), the prediction algorithm was developed based on the idea of spreading an initial belief from one node throughout the entire graph like a wavefront. The basic concept in its iterative approach is depicted with an example in Fig. 5.1. The last observation of the person was made at node n and, therefore, the probability to observe the person (or the occupancy) $o(n)$ at node n for time step $t = 0$ is one. This occupancy is now spilling into neighboring nodes with a dispersal relative to the transitional probabilities p and a speed according to the transitional time distribution τ. In other words, the transitional probability determines how much of the occupancy gets distributed to each neighboring node, whereas the time distribution defines how this proportion is spread across the time scale.

In order to enable the distribution of occupancies across the graph, a new concept

must be introduced: the probabilistic flow $f_t\left(n|\mathbf{c}\right)$. The flow traverses the graph, diverges at nodes, and gets delayed at edges. It can be described as the derivative of the occupancy which is raised by an inflow and decreased by an outflow over time. Thus, the in- and outflows are always positive. Another viewpoint is to regard the flow as a means to "transport" the occupancy from one node to another. Since the transitional probabilities are dependent on the sequence of previously visited nodes $\mathbf{c}$, the origins of each fraction of a flow need to be known, also. In the example of Fig. 5.1, the person was observed at node n with no prior route $\mathbf{c}$. It results in an inflow of $f_{t=0}\left(n\right)=1$ at node n which in turn results in an occupancy of $o_{t=0}\left(n\right)=1$. Two fractions ($f\left(a|n\right)$ and $f\left(b|n\right)$) of the inflow now disperse to node a and b with a proportion according to the transitional probabilities $p\left(a|n\right)=0.2$ and $p\left(b|n\right)=0.8$ and with a delay according to $\tau\left(a|n\right)$ and $\tau\left(b|n\right)$. Those two fractions are outflows of node n and become inflows to node a and b, at which they are once again dispersed in the same fashion. The occupancies at each node result from integrating over the difference of their in- and outflows.
The next sections give formal definitions of this procedure under different aspects of their implementation. An overview of the variables used in this chapter is given in Table 5.1. Only the following section slightly deviates from these definitions by using continuous timescales $f\left(n|\mathbf{c}\right)\left(t\right)$, $o\left(n\right)\left(t\right)$, and $\tau\left(n|\mathbf{c}\right)\left(t\right)$ in contrast to the discretized formalization $f_t\left(n|\mathbf{c}\right)$, $o_t\left(n\right)$, and $\tau_t\left(n|\mathbf{c}\right)$, as used in the table.

Table 5.1: *Definitions*

Symbol	**Description**
n, m	Indices of **nodes**
t	Discrete point in **time** $t=0,\ldots,t_{max}$
$\mathbf{c}$	Sequence of visited nodes (**condition**, **chain**) $\mathbf{c}=\{c_1,\ldots,c_j\}$ with c_j as the first and c_1 as the most recently visited node
$o_t\left(n\right)$	**Occupancy** or **observational probability** at n for time t
$f_t\left(n|\mathbf{c}\right)$	**Flow** from $\mathbf{c}$ to n at time t
$p\left(n|\mathbf{c}\right)$	**Transitional probability** from $\mathbf{c}$ to node n
$\tau_t\left(n|\mathbf{c}\right)$	**Transitional time distribution** from $\mathbf{c}$ to n; defines the probability that a transit is going to happen at time t

5.1.1 Continuous Timescale

The probabilistic flow $f\left(n\right)\left(t\right)$ describes the rate at which the observational probability (or occupancy) $o\left(n\right)\left(t\right)$ at node n changes over time t. Since a flow into one node can originate from multiple nodes along different paths $\mathbf{c}$, the inflow $f_{\text{in}}(n)(t)$ at node n results from the sum of all these flows according to Eq. 5.1.

$$f_{\text{in}}(n)(t) = \sum_{\mathbf{c}} f(n|\mathbf{c})(t) \tag{5.1}$$

Similarly, the outflow $f_{\text{out}}(n)(t)$ of a node n is the sum of all the inflows of its neighbors m with n as the last node in the sequence of visited nodes, as shown in Eq. 5.2.

$$f_{\text{out}}(n)(t) = \sum_{m}\sum_{\mathbf{c}} f(m|\{n,\mathbf{c}\})(t) \tag{5.2}$$

The occupancy $o(n)(t)$ of a node n at time t can then be regarded as the residual of the inflow until time t which did not yet leave the node via the outflow. Thus, it is the integral over the difference of the inflow and outflow:

$$\begin{aligned} o(n)(t) &= \int_0^t f_{\text{in}}(n)(x) - f_{\text{out}}(n)(x)\,dx \\ &= \int_0^t \sum_{\mathbf{c}} f(n|\mathbf{c})(x) - \sum_{m}\sum_{\mathbf{c}} f(m|\{n,\mathbf{c}\})(x)\,dx \end{aligned} \tag{5.3}$$

As previously mentioned, each inflow $f(n|\mathbf{c})(t)$ at node n results in several outflows, which in turn cause inflows at the neighboring nodes m. The fraction $\hat{f}(m|\{n,\mathbf{c}\})(t)$ of the inflow $f(n|\mathbf{c})(t)$, which gets routed towards node m, is determined by the transitional probability $p(m|\{n,\mathbf{c}\})$ (see Section 4.1) according to Equation 5.4.

$$\hat{f}(m|\{n,\mathbf{c}\})(t) = f(n|\mathbf{c})(t) \cdot p(m|\{n,\mathbf{c}\}) \tag{5.4}$$

This fraction also gets delayed towards node m according to the transitional time distribution $\tau(m|\{n,\mathbf{c}\})(t_d)$. As explained in Section 4.3, the time distribution represents the probability for each point in time t_d, that this time is needed to transit from node n to m. The time distribution for time t_d determines how much of the flow $\hat{f}(m|\{n,\mathbf{c}\})(t)$ arrives at node m at time $t + t_d$. To examine it from the viewpoint of node m, its inflow at time t is the sum of the partial flow $\hat{f}(m|\{n,\mathbf{c}\})(t - t_d)$ multiplied by $\tau(m|\{n,\mathbf{c}\})(t_d)$ for every t_d, as shown in Eq. 5.5.

$$f(m|\{n,\mathbf{c}\})(t) = \int_0^\infty \tau(m|\{n,\mathbf{c}\})(t_d) \cdot \hat{f}(m|\{n,\mathbf{c}\})(t - t_d)\,dt_d \tag{5.5}$$

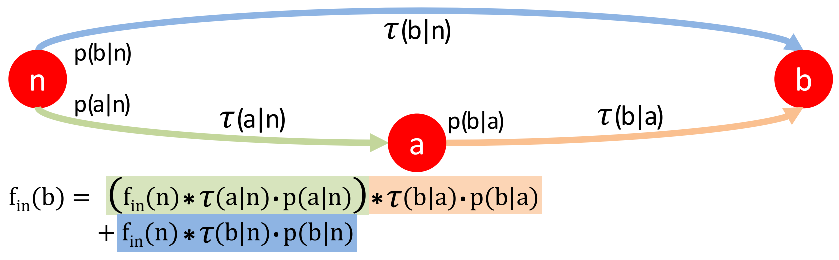
Since the integral resembles a convolution of $\hat{f}$ and τ and the transitional probability p is a constant factor, Equation 5.4 and 5.5 can be rewritten as Eq. 5.6.

$$f\left(m|\left\{n,\mathbf{c}\right\}\right)(t) = p\left(m|\left\{n,\mathbf{c}\right\}\right)\cdot\left(\tau\left(m|\left\{n,\mathbf{c}\right\}\right) * f\left(n|\mathbf{c}\right)\right)(t) \tag{5.6}$$

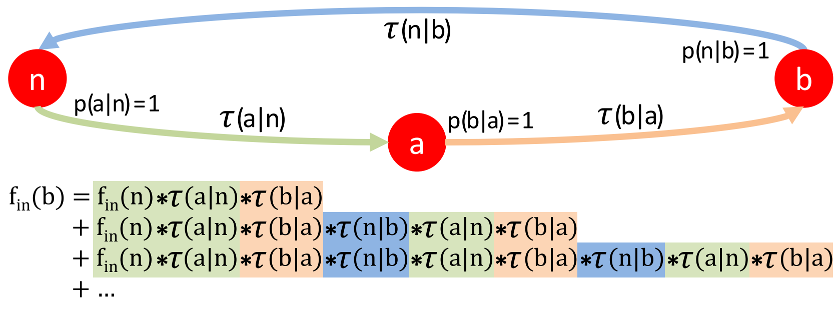
It helps in understanding the prediction method to not interpret Eq. 5.6 in the usual way that a flow $f\left(m|\left\{n,\mathbf{c}\right\}\right)$ is *dependent* on another flow $f\left(n|\mathbf{c}\right)$. A more suitable interpretation would be that an inflow $f\left(n|\mathbf{c}\right)$ to node n *creates* new outflows $f\left(m|\left\{n,\mathbf{c}\right\}\right)$ to its neighboring nodes m.
Using these equations, an observed movement can now be predicted by inducing an initial inflow of one for time $t = 0$ at the node of the last observation and calculating every flow according to Eq. 5.6. After obtaining all flows, the occupancies can finally be calculated according to Eq. 5.3.

5.1.2 Approximative Solution

Although the definition of the prediction algorithm according to Eq. 5.3 and 5.6 sufficiently describes the necessary computations of the main prediction algorithm, there is a severe drawback in its implementation: a flow may be dependent on itself. On an acyclic directed graph (like in Fig. 5.1 and 5.2(a)), the flow of each node can be computed sequentially starting from the node of the last observation, followed by its neighbors, then by their neighbors, and so on. But, as soon as the graph contains a circle or an anti-parallel edge, the inflow of a node can recursively depend on its own outflow (as shown in Fig. 5.2(b)).
Typically, the condition $\mathbf{c} = \{c_1, \ldots, c_i\}$ of a flow $f\left(n|\mathbf{c}\right)$ grows while traversing through the graph since it always gets extended by the last visited node (c_1 in the current example). Since the flow passes through the node n, a transitional probability $p\left(m|\left\{n,\mathbf{c}\right\}\right)$ and a time distribution $\tau\left(m|\left\{n,\mathbf{c}\right\}\right)$ with a matching condition $\mathbf{c}$ needs to be provided by the nodes Markov-tree in order to calculate the resulting outgoing flows. But, unfortunately, the Markov-tree is only able to provide entries with a matching condition if it was observed in the learning phase. If no match is found, the only solution to the problem is to truncate the condition $\mathbf{c}$ of the flow until a corresponding transitional probability and a time distribution is available. It would also be disadvantageous not to reduce the condition since the following nodes m will not use any of this additional information: if there were any observations with a sequence of nodes which would result in a Markov-tree of node m with an entry fully matching the condition $\{c_1, \ldots, c_i\}$, it would also force a matching entry in the previous node c_1 due to the fact that the observations causing such an entry contains the sequence $\{c_2, \ldots, c_i\}$.


τ(b|n)
p(b|n)
n
b
p(a|n)
τ(a|n)
a
p(b|a)
τ(b|a)
f_in(b) = (f_in(n) ∗ τ(a|n)·p(a|n)) ∗ τ(b|a)·p(b|a)
+ f_in(n) ∗ τ(b|n)·p(b|n)

(a) Acyclic graph


τ(n|b)
p(n|b)=1
n
b
p(a|n)=1
τ(a|n)
a
p(b|a)=1
τ(b|a)
f_in(b) = f_in(n)∗τ(a|n)∗τ(b|a)
+ f_in(n)∗τ(a|n)∗τ(b|a)∗τ(n|b)∗τ(a|n)∗τ(b|a)
+ f_in(n)∗τ(a|n)∗τ(b|a)∗τ(n|b)∗τ(a|n)∗τ(b|a)∗τ(n|b)∗τ(a|n)∗τ(b|a)
+ ...

(b) Cyclic graph

Figure 5.2: *Computation of an inflow at one node of an acyclic graph (a) and a cyclic graph (b). As can be seen in (a), the inflow of node b can easily be expressed as a formula. The influence of each edge is highlighted with its corresponding color. The inflow of node n is divided into two outflows with the ratio of $p\,(a|n)$ and $p\,(b|n)$. Each outflow becomes the inflow to node a and b respectively by convolving them with the corresponding transitional time distributions $\tau\,(a|n)$ and $\tau\,(b|n)$. The inflow to node a passes through to node b by again convolving it with $\tau\,(b|a)$. So, the inflow of node b finally results from a one-time convolution plus a twofold convolution of the inflow to node n. If a graph has a loop as in (b), an infinite recursion occurs. The inflow of node b is dependent on the inflow of node n, which in turn is dependent on the inflow of node b. The result is an infinite sum with infinite convolutions, which is obviously unfeasible to compute.*

As a result of the truncation, a flow $f\,(m|\mathbf{c})$ may pass through the graph and arrive at the same node again as an altered version $\tilde{f}\,(m|\mathbf{c})$ but with the same condition $\mathbf{c}$. The same alteration is now applied to $\tilde{f}\,(m|\mathbf{c})$ resulting in another inflow at the same node with the same condition. The consequence is an infinite loop of sequentially dependent inflows, which thwarts an effective computation.

Fortunately, the only assumption about the system made in Section 3.1 demands that the system changes continuously and does not skip intermediate states. It implies that the system needs a certain time to transit from one node to another one; hence, the soonest a neighboring node can be reached is the next time step. This fact enables an approximative computation of the flows since if the flow computed up to the current

```
Input
  1  $f^{t=0}(n|\mathbf{c})$                              //Initial inflow caused by the observation
  2  $l$                                                  //Max time horizon for prediction
  3  $p(n|\mathbf{c})$                                    //All learned transitional probabilities
  4  $\tau(n|\mathbf{c})$                                 //All learned transitional time distributions
Algorithm
  5    $\mathbf{F} = \{f^{t=0}(n|\mathbf{c})\}$;          //Initialize set of all known flows
Label 1:                                                  Approximation
  6    for $t = 1, \ldots, l$;                            //Iterate through time horizon
  7      $\mathbf{E} = \{f^{t=0}(n|\mathbf{c})\}$;        //Create temporary set of flows
  8      for every $f(n|\mathbf{c})$ in $\mathbf{F}$;     //Iterate through all known flows
  9        for every neighbor $m$ of $n$;                 //Iterate through all neighboring nodes
 10          truncate $\mathbf{c}$ until a $p(m|\{n,\mathbf{c}\})$ and $\tau(m|\{n,\mathbf{c}\})$ is available;
 11          $\hat{f}(m|\{n,\mathbf{c}\}) = p(m|\{n,\mathbf{c}\}) \cdot \tau(m|\{n,\mathbf{c}\}) * f(n|\mathbf{c})$;   //Calculate new flows based
                                                          //on previous approximation
 12          add $\hat{f}(m|\{n,\mathbf{c}\})$ to $\mathbf{E}$;   //Update temporary flow set
 13      set $\mathbf{F}$ to $\mathbf{E}$;                //Set current set of known flows to newly calculated flows
Label 2:                                                  Calculate result
 14    for $t = 1, \ldots, l$;                            //Iterate through time horizon
 15      $o_t(n) = \sum_{x=0}^{t}\left(\sum_{\mathbf{c}} f_x(n|\mathbf{c}) - \sum_m \sum_{\mathbf{c}} f_x(m|\{n,\mathbf{c}\})\right)$;   //Calculate observational
                                                          //probabilities
Return
 16  $o_t(n)$,   $t = 1, \ldots, l$,   $n = 0, \ldots, N$  //Observational probabilities
                                                          //for all nodes
```

Figure 5.3: *Approximative prediction method*

time step is considered to be correct, it can only have an influence on itself for the future time steps. The best approach is to change the calculation of the flows from a spatial perspective to a temporal viewpoint.

A flow at time step t can only be dependent on flows up to the time step $t-1$. Therefore, an inflow at a given time step can only add values to itself for future time steps. Since both the in- and outflows are always positive per definition (only the difference $f(n)$ of both can be negative), a self inflicted decrease for further time steps is impossible. It enables an approximation of the flows by iteratively applying Eq. 5.7 on all flows. The algorithmic implementation of the approximation is shown in Fig. 5.3.

$$f^{t+1}(m|\{n,\mathbf{c}\}) = p(m|\{n,\mathbf{c}\}) \cdot \tau(m|\{n,\mathbf{c}\}) * f^{t}(n|\mathbf{c}) \tag{5.7}$$

As the base case, every $f^{t=0}$ is initially given since it represents the last observable state on which the whole prediction is based. As previously mentioned, a flow at time $t = k$ can only be influenced by other flows for time steps $t < k$. Therefore,

after calculating every $f^{t=1}$ based on $f^{t=0}$, the flows yield the correct results up to the time step $t = 1$. The next iteration extends the correct part of the flows to $t \leq 2$. According to mathematical induction, the approximated flows are valid for every $t \leq l$ after calculating l iterations.
The induction can be depicted by the example of a person walking. The initial state is represented by an inflow of one at the node at which the person was currently observed. Every other inflow is zero since a person can not be at different places simultaneously. By calculating one iteration of Eq. 5.7, only the inflows to the neighboring nodes get updated. Since the person can not go beyond those neighbors in one time step, they can also not follow a circular structure and arrive at the neighboring nodes via another route. Hence, the flows will not change anymore for time steps $t \leq 1$ after the first iteration of Eq. 5.7. The earliest situation possible for the person to arrive at a node they have already visited occurs at time step $t = 2$ if the person reverts to the initial node. But, this change in the inflow of the initial node can under no circumstances cause changes in the inflow of other nodes for time steps $t \leq 2$. Thus, after the second iteration of Eq. 5.7, every flow is now static for $t \leq 2$.
Usually, the flows do not only yield the correct results up to time step $t \leq l$ but also provide the better approximation for $t > l$ the more iterations are calculated. This is due to the fact that after each iteration the magnitude of each circular flow gets lower (since the transitional probability is ≤ 1 and thus can only decrease a flow) and more spread across the time due to the convolution with the transitional time distribution.
The prediction method was developed for a real-time application as introduced in Section 1.2, but the approximative computation was not able to meet the necessary speed requirements on large graphs due to a much too high workload on the processor. A similar but much faster strategy to circumvent the expensive computation will be presented in the following section.

5.1.3 Frequency Domain

Since the convolutions of the flows with the transitional time distributions takes up most of the computational time, it is only logical to focus on this operation. The convolution theorem states that the Fourier transformation of a convolution is the pointwise product of both Fourier transformed operands, as shown in Eq. 5.8.

$$\mathcal{F}\left(\tau\left(m|\left\{n,\mathbf{c}\right\}\right) * f\left(n|\mathbf{c}\right)\right) = T\left(m|\left\{n,\mathbf{c}\right\}\right) \cdot F\left(n|\mathbf{c}\right) \tag{5.8}$$

with

$$F\left(n|\mathbf{c}\right) = \mathcal{F}\left(f\left(n|\mathbf{c}\right)\right)$$

and

$$T\left(m|\left\{n,\mathbf{c}\right\}\right)=\mathcal{F}\left(\tau\left(m|\left\{n,\mathbf{c}\right\}\right)\right)$$

Using this theorem, the complexity of the convolution is reduced from approximately $O\left(n^2\right)$ to $O\left(n\right)$ (omitting the computations needed for the Fourier transformation). The transformation only needs to be done once on the transitional time distributions and for the initial inflow to start the prediction. Afterwards, every intermediate computation can be conducted in frequency domain according to Eq. 5.9.

$$F\left(m|\left\{n,\mathbf{c}\right\}\right)=p\left(m|\left\{n,\mathbf{c}\right\}\right)\cdot T\left(m|\left\{n,\mathbf{c}\right\}\right)\cdot F\left(n|\mathbf{c}\right) \tag{5.9}$$

A second set of Fourier transformations is necessary for the final calculation of the occupancies in order to revert to the time domain. Therefore, Equation 5.3 needs to be slightly adjusted to Eq. 5.10.

$$o_t\left(n\right)=\int_0^{-t}\mathcal{F}\left(F\left(n\right)\right)\left(x\right)dx \tag{5.10}$$

with

$$F\left(n\right)=\sum_{\mathbf{c}}F\left(n|\mathbf{c}\right)-\sum_{m}\sum_{\mathbf{c}}F\left(m|\left\{n,\mathbf{c}\right\}\right)$$

Unfortunately, using this approach yields the same problem as the method presented in Section 5.1.2: cyclic structures in the topological graph are causing flows which are dependent on themselves. The approximation in the time domain according to Eq. 5.7 in Section 5.1.2 can easily be transferred to the frequency domain in order to circumvent the convolution resulting in Eq. 5.11.

$$F^{t+1}\left(m|\left\{n,\mathbf{c}\right\}\right)=p\left(m|\left\{n,\mathbf{c}\right\}\right)\cdot T\left(m|\left\{n,\mathbf{c}\right\}\right)\cdot F^{t}\left(n|\mathbf{c}\right) \tag{5.11}$$

After initially transferring all transitional time distributions $\tau\left(m|\mathbf{c}\right)$ into the frequency domain, every flow can be approximated by iteratively applying Eq. 5.11 l times in order to get the correct result up to time step l. Next, Eq. 5.10 can be applied to calculate the final observational probability $o_t\left(n\right)$ for every node n and every time step t. This procedure is depicted in Fig. 5.4. The approximative method, presented in Section 5.1.2, also needs l iterations for each flow to yield the correct result up to time step l, but in each iteration a convolution needs to be performed, whereas Eq. 5.11 only utilizes multiplications. Therefore, calculating the flows in frequency domain

```
Input
  1   $f^{t=0}(n|\mathbf{c})$                              //Initial inflow caused by the observation
  2   $l$                                                   //Max time horizon for prediction
  3   $p(n|\mathbf{c})$                                     //All learned transitional probabilities
  4   $\tau(n|\mathbf{c})$                                  //All learned transitional time distributions
Algorithm
  5   $T(m|\{n,\mathbf{c}\}) = \mathcal{F}(\tau(m|\{n,\mathbf{c}\}))$;    //Transform time distributions into freq. domain
  6   $F^{t=0}(n|\mathbf{c}) = \mathcal{F}(f^{t=0}(n|\mathbf{c}))$;       //Transform initial inflow into freq. domain
  7   $\mathbf{F} = \{F^{t=0}(n|\mathbf{c})\}$;                           //Initialize set of all known flows
Label 1:                                                                   Iteration
  8   for $t = 1,\dots,l$;                                                //Iterate through time horizon
  9     $\mathbf{E} = \{F^{t=0}(n|\mathbf{c})\}$;                         //Create temporary set of flows
 10     for every $F(n|\mathbf{c})$ in $\mathbf{F}$;                      //Iterate through all known flows
 11       for every neighbor $m$ of $n$;                                  //Iterate through all neighboring nodes
 12         truncate $\mathbf{c}$ until a $p(m|\{n,\mathbf{c}\})$ and $T(m|\{n,\mathbf{c}\})$ is available;
 13         $\hat{F}(m|\{n,\mathbf{c}\}) = p(m|\{n,\mathbf{c}\}) \cdot T(m|\{n,\mathbf{c}\}) \cdot F(n|\mathbf{c})$;   //Calculate new flows based
                                                                          //on previous approximation
 14         add $\hat{F}(m|\{n,\mathbf{c}\})$ to $\mathbf{E}$;            //Update temporary flow set
 15     set $\mathbf{F}$ to $\mathbf{E}$;                 //Set current set of known flows to newly calculated flows
Label 2:                                                                   Calculate result
 16   for every $F(n|\mathbf{c})$ in $\mathbf{F}$;                        //Iterate through all flows
 17     $f(n|\mathbf{c}) = \mathcal{F}^{-1}(F(n|\mathbf{c}))$;            //Transform all flows back to time domain
 18   for $t = 1,\dots,l$;                                                //Iterate through time horizon
 19     $o_t(n) = o_{t-1}(n) + \sum_{\mathbf{c}} f_t(n|\mathbf{c}) - \sum_m \sum_{\mathbf{c}} f_t(m|\{n,\mathbf{c}\})$;   //Calculate observational
                                                                          //probabilities
Return
 20   $o_t(n)$,   $t = 1,\dots,l$,   $n = 0,\dots,N$                      //Observational probabilities
                                                                          //for all nodes
```

Figure 5.4: *Prediction in frequency domain*

gives similar results to the approximative method with less computations involved. It is worth noting that the observational probabilities may provide good estimates for $t > l$ too since the iteration of Eq. 5.11 is only necessary to resolve circular structures. If the topological graph is acyclic, each flow only has to be computed once by starting at the initial node and calculating the flows to its neighbors, then calculating the flows to their neighbors, and so on until the entire graph has been traversed.

But, unfortunately, the approximation in frequency domain yields a major problem: the computationally most feasible method for transforming a signal into the frequency domain is to apply a Discrete Fourier Transform (DFT). It transforms a vector in time domain of length l into a complex vector in frequency domain of length l. Since all the resulting flows in Eq. 5.10 are fixed to this length, the occupancy can only be calculated for $t \leq l$. Thus, the prediction horizon is limited to size l of the transformation. If a prediction up to a further point in time is desired, l needs to be increased accordingly. Unfortunately, the memory requirements and computational complexity increase with l since the multiplication in Eq. 5.9 implements an element-wise multiplication of the two vectors T and F of size l. A more flexible and computationally less demanding method is presented in the next section.

5.1.4 Iterative Solution

The main problem of calculating a flow $f\left(m|\left\{n,\mathbf{c}\right\}\right)(t)$ according to Eq. 5.6 is that all other flows $f\left(n|\mathbf{c}\right)(t)$ need to be known beforehand. The approach of the two previous sections circumvent this problem by approximating them. But, the fact that all the flows are known for the initial time step $t = 0$ (due to the last observation) enables an easier and faster solution.
For a given node n and a given time step t, the flow to its neighboring node m results (according to Eq. 5.6) from the product of the corresponding transitional probability $p\left(m|\left\{n,\mathbf{c}\right\}\right)$, the fragment responsible for the current time step of the convolution of the transitional time distribution $\tau\left(m|\left\{n,\mathbf{c}\right\}\right)$, and the inflow $f\left(n|\mathbf{c}\right)$ as shown in Eq. 5.12.

$$f_t\left(m|\left\{n,\mathbf{c}\right\}\right) = p\left(m|\left\{n,\mathbf{c}\right\}\right) \cdot \sum_i \left(\tau_i\left(m|\left\{n,\mathbf{c}\right\}\right) \cdot f_{t-i}\left(n|\mathbf{c}\right)\right) \tag{5.12}$$

It is worth noting that the transitional time distribution $\tau_i\left(m|\left\{n,\mathbf{c}\right\}\right)$ for $i = 0$ is zero due to the assumption of a continuous system. Thus, only inflows of the previous time steps contribute to the current calculation. The iteration starts at $t = 1$ and updates all available flows for the current time step based on the previous time steps in each iteration. Similar to Eq. 5.6 in Section 5.1.1, Eq. 5.12 should be interpreted as that the previous set of flows (right side of the equation) get dispersed at the nodes they are arriving at and thus causing new flows (left side of the equation) for the current time step.
The occupancy of node n for the current time step t results from the previous occupancy enhanced by every momentary inflow and decreased by every momentary outflow to all of its neighbors m as given in Eq. 5.13.

$$o_t\left(n\right) = o_{t-1}\left(n\right) + \sum_{\mathbf{c}} f_t\left(n|\mathbf{c}\right) - \sum_m \sum_{\mathbf{c}} f_t\left(m|\left\{n,\mathbf{c}\right\}\right) \tag{5.13}$$

By iteratively calculating the flows for every time step $t = 0,\ldots,l$ according to Eq. 5.12 and finally applying Eq. 5.13, circular dependencies are resolved without adding any

```
Input
  1  $f_{t=0}(n|\mathbf{c})$                        //Initial inflow caused by the observation
  2  $l$                                            //Max time horizon for prediction
  3  $p(n|\mathbf{c})$                              //All learned transitional probabilities
  4  $\tau(n|\mathbf{c})$                           //All learned transitional time distributions
Algorithm
  5  $\mathbf{F} = \{f^{t=0}(n|\mathbf{c})\}$;      //Initialize set of all known flows
Label 1:                                                                      Iteration
  6  for $t = 1, \ldots, l$;                        //Iterate through time horizon
  7    for every $f(n|\mathbf{c})$ in $\mathbf{F}$; //Iterate through all known flows
  8      for every neighbor $m$ of $n$;             //Iterate through all neighboring nodes
  9        truncate $\mathbf{c}$ until a $p(m|\{n,\mathbf{c}\})$ and $\tau(m|\{n,\mathbf{c}\})$ is available;
 10        $f_t(m|\{n,\mathbf{c}\}) = p(m|\{n,\mathbf{c}\}) \cdot \sum_i (\tau_i(m|\{n,\mathbf{c}\}) \cdot f_{t-i}(n|\mathbf{c}))$;  //Calculate flows for
                                                    //current time step
 11        add $f(m|\{n,\mathbf{c}\})$ to $\mathbf{F}$, if not yet included;  //Update flow set
Label 2:                                                                      Calculate result
 12  for $t = 1, \ldots, l$;                        //Iterate through time horizon
 13    for $n = 0, \ldots, N$;                      //Iterate through all nodes
 14      $o_t(n) = o_{t-1}(n) + \sum_{\mathbf{c}} f_t(n|\mathbf{c}) - \sum_m \sum_{\mathbf{c}} f_t(m|\{n,\mathbf{c}\})$;  //Calculate
                                                    //observational probabilities
Return
 15  $o_t(n)$,   $t = 1, \ldots, l$,   $n = 0, \ldots, N$   //Observational probabilities
                                                    //for all nodes
```

Figure 5.5: *Iterative prediction method*

additional computational complexity to the procedure presented in Section 5.1.1 and the multiple convolutions for every iteration of Eq. 5.7 are basically reduced to one convolution per flow. The main difference between both methods is that the iterative approach calculates the convolution of Eq. 5.5 for each flow only once but for one time step after another; whereas, the approximative approach of Section 5.1.2 needs one full convolution for every flow and every iteration. In comparison to the approach in frequency domain as explained in Section 5.1.3, the iterative approach requires even fewer computations since the calculation of a flow for time step $t \leq l$ only needs t multiplications (see Eq. 5.12) at most, whereas the approach in frequency domain utilizes l multiplications for every iteration of Eq. 5.9. Additionally, the initial and final Fourier transformations are not necessary.

An implementation of the algorithm is shown in Fig. 5.5, and Fig. 5.1 shows the iterative approach in an example with a first-order Markov-tree. It only depicts an acyclic graph in order to show the complete prediction up to its final, stable state in only six steps.

5.2 Model Update

Usually, it cannot be guaranteed that the dynamics of a system remain static after the initial transitional and probabilistic models are learned. In order to account for changes, a model update would be beneficial. The prediction framework comprises three elements which can be updated: the topological model, the transitional probabilities, and the transitional time distributions.

The topological model is the most cumbersome element to update. The Mean-Shift clustering, used for determining the spatial coverage, is not able to include new observations into the clustering result without reprocessing the whole set of previously observed data. A naive approach would be to collect new observations and create a new topological model in a batch if the observations indicate a significant change in the underlying topology. Unfortunately, this procedure also invalidates all transitional probabilities and time distributions since the new topology cannot be mapped onto the old model due to a different number of nodes, shifted positions of the nodes, and different results in the Delaunay triangulation. Considering the topological graph as a Self-Organizing Map [Kohonen, 1990] or a Neural Gas [Martinetz et al., 1991] and updating it to new observations does not change the topology of the graph and provides a better alternative than a batch update. But, by shifting nodes towards new observations, the Delaunay triangulation (see Section 3.4) is violated and the edges may not connect only neighboring nodes any more. Additionally, the transitional time distributions will become invalid since the time to transit from one state to another is dependent of their respective spatial position. Also, the transitional probabilities are not guaranteed to be correct anymore since they usually depend on the spatial position, too. It is apparent that it is best not to update the topological model after it is created. In order to account for possible changes and to cover sparse regions, a set of random nodes can be seeded across the whole state space before the edges are created by the Delaunay triangulation. But, if the topology has changed significantly, a complete recalculation of the topological graph, the transitional probabilities and time distributions in inevitable.

If the topological graph is kept static, the use of the Markov-tree enables an easy updating of the transitional probabilities and time distributions. Since the transitional probabilities are based on relative frequencies, they can be updated to new observations on-line by just incrementing the corresponding counter in the Markov-tree in the same fashion as described in Section 4.2. By using a simple decaying mechanism, the Markov-trees can even account for frequent changes in the motion mechanics and thus adapt to different situations during its application phase. The update scheme of the transitional time distributions, on the other hand, depends on their implementation (see Section 4.3): if KDE was chosen to represent the distribution, it can be updated by adding a new kernel to the distribution, similar to a histogram, every time a new observation is available as explained in Section 4.3.1. But, if a parametric representation of the time distribution is used, an on-line update is usually not possible (depending on the involved method - e.g., GMM, or KDA) and a batch update is often the only possibility to incorporate new observations.

5.3 Conclusion and Summary

The novel prediction algorithm which distinguishes itself from all other existing prediction methods not only by its comprehensive information content but also by its unique approach of using probabilistic flows, was presented in this chapter. This algorithm uses the topological graph, first introduced in Chapter 3, which was enriched by transitional probabilities and time distributions in a learning phase as explained in Chapter 4. The probabilistic flow was introduced in Section 5.1. It describes the transfer of an observational probability (or occupancy) from one node to a neighboring node over time. The formal definition of the prediction method was given in Section 5.1.1, its practical implementation was presented in Section 5.1.2 (which got optimized in Section 5.1.3 by a transformation into the frequency domain), and a computationally less expensive version was given in Section 5.1.4. If the system dynamics have changed, the topological and probabilistic models can be updated as explained in Section 5.2.

The proposed procedure to predict a movement with the above mentioned methods is intended as follows: First, the last known state of the system must be transformed into corresponding flows. In the example of a person walking along an area, the last node n and path $\mathbf{c}$ of the person needs to be determined and an inflow $f(n|\mathbf{c})$ to node n with a value of one for the time step $t = 0$ must be created. The person moved along the nodes $\mathbf{c} = c_1, \ldots, c_i$ to n, and their observational probability at node c_1 and later at node n was one since they were directly observed. Thus, the flow to n along the path $\mathbf{c}$ is one. Second, the first iteration of Eq. 5.12 is performed. It creates new inflows at the neighboring nodes m with the condition $\mathbf{c}$ extended by n. Whereas only one flow was known in the initial state, additional flows from the last node n towards its neighbors m are now in existence. In every further iteration of Eq. 5.12, the existing flows may cause additional ones with an extended condition. Eventually, a truncation of the condition will be necessary due to unavailable entries in the corresponding Markov-tree, as explained in Section 5.1.2. This will restrict the total number of used flows and will prevent their exponential duplication at every iteration. In order to obtain a prediction up to time step $t = l$, a total of l iterations of Eq. 5.12 needs to be performed. Finally, the observational probabilities of every node up to time step $t = l$ can be computed using Equation 5.13.

In order to enable the incorporation of new observations into the prediction algorithm during its application, it is recommended to seed random nodes into the topological graph in the learning phase before edge creation to account for unobserved spatial states. The Markov-tree enables an easy on-line updating of the transitional probabilities and KDE is strongly recommended for representing the transitional time distributions due to its incremental updating capabilities.
Chapter 3 through Chapter 5 described the whole prediction algorithm and the necessary methods involved. In order to evaluate the results provided by the prediction algorithm, an error measure and several experiments will be presented in the following chapter. The proposed prediction algorithm will also be compared to the state of the art methods as introduced in Section 2.2.

Chapter 6

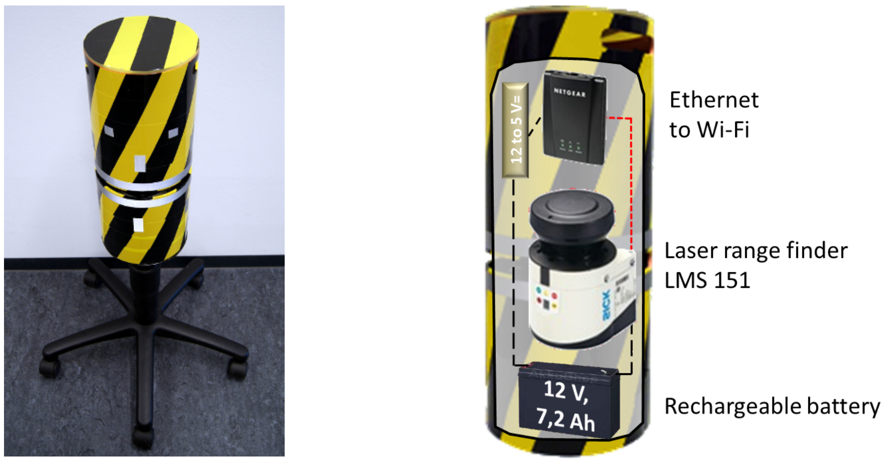
Experimental Evaluation

In the last chapter, the main prediction framework and its methods were described extensively. In this chapter, several experiments and their results will be presented in order to compare the framework to the state of the art and to evaluate the quality of the prediction results.

Two datasets will be used for evaluation. The first was recorded in the Humboldt-Foyer at the Ilmenau University of Technology, and the second was taken from the Beijing taxi trajectory dataset [Zhu et al., 2013, Zhang et al., 2011, Zhang, 2009]. Both datasets are described in Section 6.1 in greater detail.

The most common error measures are presented in Section 6.2 in order to assess different aspects of the prediction framework and to compare it to the state of the art. Those measures are not suitable for fully evaluating the proposed prediction framework since it provides a spatio-temporal probability distribution as a result. Thus, a new error measure, similar to the receiver operating characteristic (ROC), is also presented.

These error measures are used in the experiments in Section 6.3, 6.4, and 6.6. The first experiment in Section 6.3 examines the influence of different representations of the spatial space. It compares the cluster-based topology to a simple grid representation and analyzes the effect of different cluster sizes on prediction accuracy. Section 6.4 describes and evaluates an experiment to contrast the iterative prediction with the prediction using frequency domain (see Section 5.1.3 and 5.1.4). The second experiment will show whether the approximations in frequency domain have a considerable influence on prediction accuracy and examines the speed-up of the iterative approach. The proposed prediction method is then compared to the state of the art [Bennewitz, 2004, Vasquez Govea, 2007, Ikeda et al., 2013] in Section 6.5 and a broad evaluation is given in Section 6.6 in order to enable a comparison to new prediction algorithms. Section 6.7 will summarize the experiments.

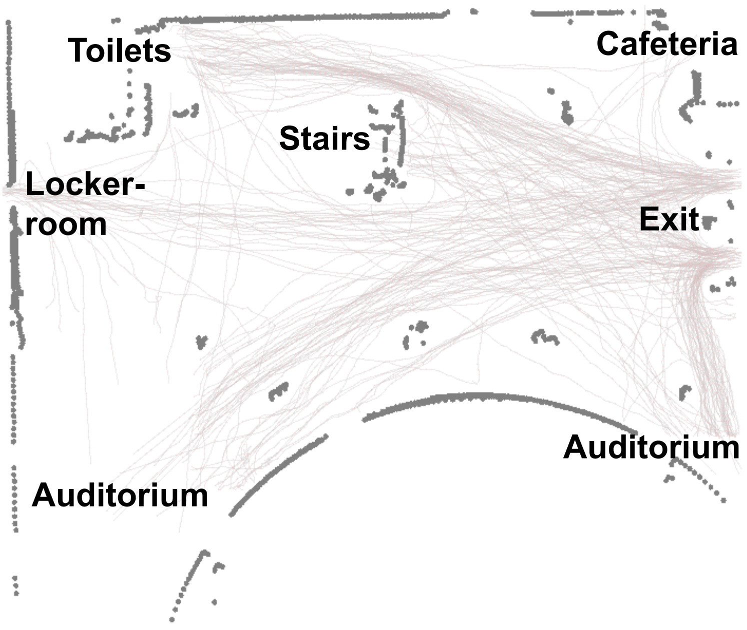


(a) Mobile LIDAR unit (b) Interior of the unit

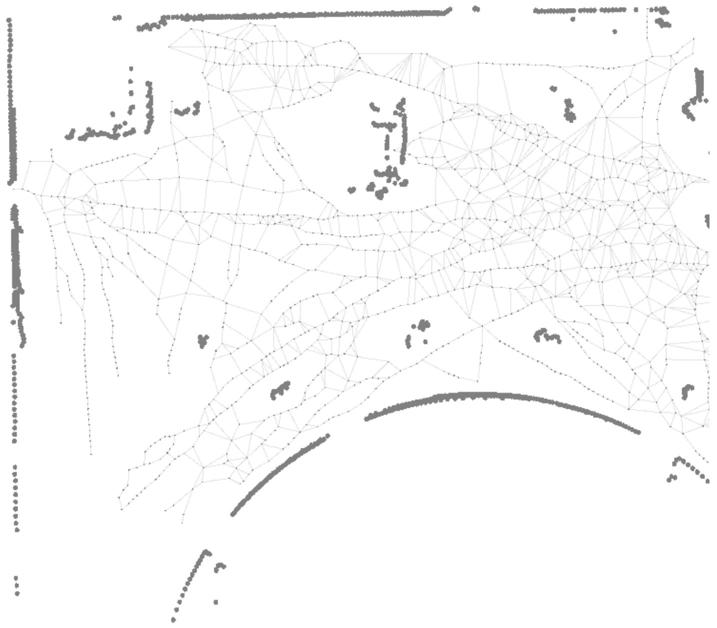
Figure 6.1: *The mobile LIDAR sensor unit used for acquiring the Humboldt dataset. A picture of the sensor on top of a trolley is shown in (a). Reflective tape was applied in order to improve the automated calibration of multiple sensors. The LIDAR sensor is placed behind the slit between the two horizontal reflector tapes. A schematic sketch of the interior is shown in (b). The electronics are powered by a lead battery either directly or by a DC Power converter. The LMS151 LIDAR sensor is connected to an Ethernet to Wi-Fi bridge in order to provide a wireless data connection to a recording server. The upper part (covered by a cap) of the mobile unit provides a charging port, a power switch, a replaceable fuse, and access to the Wi-Fi bridge in order to connect the recording server directly to the scanner via an Ethernet cable, if desired.*

6.1 Databasis

A dataset of human movement trajectories was recorded in the Humboldt-Foyer at the Ilmenau University of Technology in order to develop the prediction framework. The recordings took place in the foyer (measuring approximately $15m \times 26m$ and depicted in Fig. 6.2) of the main auditorium between two exams. It resulted in the Humboldt dataset containing a total of 231 trajectories with a mean length of $16.8m$ and a mean duration of $19.4s$. They were recorded with three mobile LIDAR (Light Detection And Ranging) sensor units each consisting of a SICK LMS151 scanner, a lead battery as a power supply, and a Wi-Fi bridge which connected the scanner to the recording server (see Fig. 6.1). After registering their scan data with an automated calibration procedure, as described in [Schenk et al., 2012a, Schenk et al., 2012b], the students were tracked with an algorithm incorporating multiple particle filters while they walked along the foyer, as explained in [Schenk et al., 2011]. After filtering out short tracks ($\leq 1m$), it resulted in 231 trajectories of which one half were used for creating the topological graph (shown as a light gray mesh in Fig. 6.2) and the probabilistic model, while the other part was used for evaluation in the experiments of



(a) Humboldt dataset



(b) Topological graph

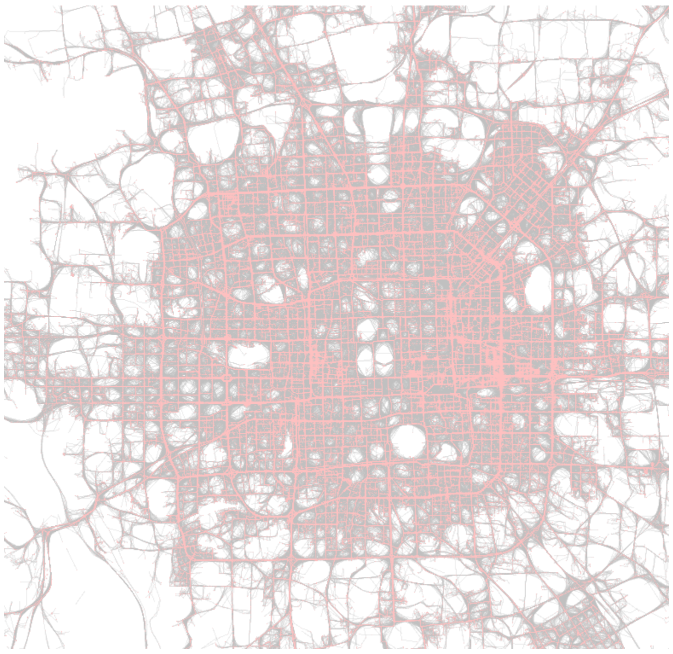
Figure 6.2: *Top view of the Humboldt Foyer. In (a), the layout and trajectories are shown. Walls and static obstacles are depicted in dark gray, and the trajectories are shown in a light gray color. In (b) the trajectories are replaced by a topological graph with its edges shown as a light gray mesh and the nodes as dark dots.*

the following sections. The first $3m$ of every test trajectory were used as the observation and the remaining part was taken as the ground truth in the evaluation.

Since a dataset with only 115 test trajectories can hardly provide a statistically sound evaluation and the creation of a bigger dataset is both expensive and beyond the scope of this thesis, an openly available and extensive trajectory dataset was chosen to make this prediction framework comparable to other approaches: the Beijing taxi trajectory dataset [Zhu et al., 2013, Zhang et al., 2011, Zhang, 2009]. For a comparison to the state of the art, it would have been more convenient to use a dataset containing pedestrian movements; but, unfortunately, no published dataset with the necessary number of trajectories was available at the time. The Beijing taxi trajectory dataset was chosen as a reference since cars are usually operated by humans and thus provide movement trajectories following remotely similar intentions as pedestrians but on a larger scale. It contains the GPS tracks of $8,602$ taxis driving in the Beijing region for a duration of one month (see Fig. 6.3). The raw dataset consists of nearly 125 million GPS positions and covers a traveled distance of approximately 52 million kilometers in total.

Since the tracks are only stored as a sequence of GPS points for each individual car, a preprocessing step was necessary to extract individual trips. First, the track of a car was split if more than 15 minutes have passed between two consecutive points. It is a trade-off between splitting the trips of different passengers and maintaining a continuous track in case of traffic congestion or short stopovers. In order to account for missing intermediate points and disabled GPS devices, tracks are split if a distance greater than $1km$ lies between two consecutive GPS positions. If the remaining trajectories are shorter than $200m$, they are discarded because they do not provide much useful information for a prediction task on the scale of Beijing. The filtering resulted in a dataset with a total of 5.1 million trajectories consisting of 116 million GPS positions and covering a distance of 26 million kilometers in total. This dataset was also split in half - one for training and the other part for evaluation. Unfortunately, the evaluation of the prediction framework on the whole 2.05 million test trajectories would require several decades on modern computer hardware (an Intel(R) Core(TM) i7 quad core processor running at $2.67GHz$ was used for the evaluation). Therefore, the $1,000$ longest trajectories (with 667 thousand GPS points, a length of 134 thousand kilometers and a total duration of more than $7,398$ hours) were taken as a reduced test set. In order to predict a test trajectory, the first $10km$ were used as an observation while the remaining part was used as the ground truth for the evaluation.

Since almost every prediction algorithm found in literature is evaluated on a dataset which is not available to the public, the use of the openly available Beijing taxi trajectory dataset enables other scientists to compare their prediction methods to the prediction framework presented in this thesis. By focusing on taxi cab movements in


(a) Beijing taxi trajectory dataset

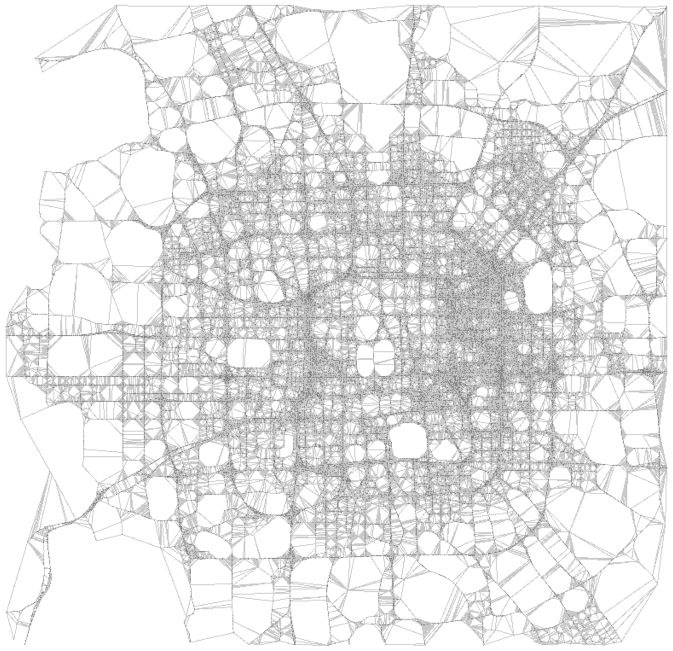

(b) Topological graph

Figure 6.3: *Beijing taxi trajectory dataset. In (a), the GPS points are shown as pale red points and the connections between them as gray lines. Only a small subset (approximately* 218 *thousand trajectories with* 5.2 *million GPS points) is presented, covering an area of* $30km \times 31km$*. A topological graph, created on these trajectories, comprising* $14,272$ *nodes and* $57,067$ *edges, is illustrated in (b). Only the edges can be seen since the nodes are too small to be identified.*

an enormous city, a prediction algorithm can also be compared to the proposed prediction framework utilizing a variety of different datasets which share the same type of motions such as the NYC's Taxi Trip Data [Whong, 2014], the T-Drive dataset, which was recorded in Beijing [Yuan et al., 2010, Yuan et al., 2011, Zheng, 2011], the GeoLife GPS Trajectories dataset [Zheng et al., 2009, Zheng et al., 2008, Zheng et al., 2010, Zheng, 2012], the epfl/mobility dataset of taxi cab rides in San Francisco [Piorkowski et al., 2009], or a collection of user-submitted GPS tracks on © OpenStreetMap [OpenStreetMap and contributors, 2013].

6.2 Error Measure

In the state of the art, several error measures are applied for evaluating a trajectory prediction algorithm: the Euclidean distance error, which describes the mean distance of the prediction to the ground truth, the more intuitive lock-in accuracy, the next state accuracy for discretized state spaces such as those found in mobile networks, and the prediction error over the observed proportion of the trajectories are the most prominent. Since none of these error measures are able to fully grasp the capabilities of the presented prediction method, a new error measure is presented in Section 6.2.5.

6.2.1 Euclidean Distance Error

The most basic measure describes the mean Euclidean distance between the predicted and the real location for a given time offset t_Δ [Jeung et al., 2010]. The error can be calculated as the Euclidean distance of $\mathbf{o}_\Delta$ and $\mathbf{x}_{S+\Delta}$ utilizing an observed trajectory up to time t_S, the point $\mathbf{x}_{S+\Delta}$ of the trajectory for the future time $t_{S+\Delta} = t_S + t_\Delta$, and the prediction $\mathbf{o}_\Delta$ for $t_{S+\Delta}$. The mean Euclidean distance error is computed by averaging over all test trajectories.
By comparing different prediction algorithms using the Euclidean distance error, three problems can arise if they are not evaluated on the same dataset. The first and most obvious problem also applies to every other error measure: the prediction performance is dependent on the system dynamics which should be predicted. For example, a prediction algorithm evaluated on the movements of planets may provide excellent results, whereas another prediction method evaluated on the Brownian movement of molecules is more likely to give significantly poorer results. It is only possible to contrast two prediction methods evaluated on different datasets with each other if the predicted system dynamics are similar. For instance, it is likely that the movements of people walking in a shopping mall are following dynamics similar to the movements of people at an airport thus enabling a comparison. But, in the case of more diverse system dynamics, such as people in a shopping mall and cars driving around in a city,

the Euclidean distance error measure can provide at least a rough comparison of two different prediction methods. The example of cars in a city versus pedestrians in a shopping mall points to the second problem of comparing two methods on different datasets with the Euclidean distance error: the spatial and temporal scales must be in the same magnitude. It is, for example, difficult to assess if a method which is predicting car movements in a city with an error of 1km for a time horizon of $t_\Delta = 60\text{min}$ is better or worse than a prediction algorithm that is able to predict someone's movement in a shopping mall with a mean error of 15m for a time horizon of $t_\Delta = 30\text{s}$. The third problem with using the Euclidean distance error on different datasets is that it does not provide any insight into the influence of the length of observed trajectories. A prediction method tested on a dataset with long trajectories can utilize more information and will, most likely, perform better than a prediction algorithm evaluated on a dataset with short trajectories.

Due to these deficiencies, the Euclidean distance error measure only provides useful information if the prediction algorithms are evaluated on the same dataset with the same time offset. Since the error is easy to compute, it is best utilized for fine-tuning a prediction algorithm and assessing different methods in its development.

6.2.2 Lock-In Accuracy

The Euclidean distance error also has a practical disadvantage. In the example of predicting someone's movement in a mall, a prediction algorithm with an error of $50m$ is not better than a prediction method with an error of $200m$ - both predictions are basically useless. The likelihood of obtaining a correct prediction would be much more interesting for evaluating the capabilities of a prediction algorithm. The common interpretation of a "correct" prediction is that it is within a certain distance (or lock-in range) to the ground truth [Ikeda et al., 2013]. Thus, the lock-in accuracy for a given distance d and time delta t_Δ describes the proportion of the test trajectories whose prediction is within the distance d to the ground truth.

At first glance, the lock-in accuracy does not solve the problems of the Euclidean distance error and even introduces a new dependency by the lock-in range d. However, it is easier to find suitable ranges for different dynamic systems (e.g., it might be reasonable to have a range of $2m$ for a pedestrian and a time horizon of $30s$ or $200m$ for a car and a time horizon of 10 minutes) than to directly compare their Euclidean distance errors. Thus, the lock-in accuracy gives a more intuitive assessment and it should be preferred over the Euclidean distance error.

6.2.3 Next State Accuracy

The Euclidean distance error and the lock-in accuracy is primarily suitable for continuous state spaces. In the case of a prediction in a discretized state space, such as mobile networks, it is common to evaluate the prediction method in terms of the prediction accuracy for the next state [Prasad and Agrawal, 2010, Wanalertlak et al., 2011]. It is defined as the proportion of the test trajectories whose next state was predicted correctly, similar to the lock-in accuracy.
By merely evaluating the prediction method for the next (or sometimes the second next or generally speaking the n^{th} next) discrete state, spatial scales in different problem statements have almost no influence on the prediction error as long as the chosen discretization samples the state space adequately. Yet, similar to the Euclidean distance measure, the next state accuracy does not reflect the amount of information used for prediction. Furthermore, by only focusing on the next spatial state, the time horizon is neglected and the prediction accuracy for different time offsets is not assessed.

6.2.4 Error over Trajectory

One drawback of the Euclidean distance error, the lock-in accuracy, and the next state accuracy is the inability to account for the number of observations used to predict a trajectory: the longer a trajectory is observed, the more information is available for the algorithm to substantiate a prediction thus increasing the accuracy. Furthermore, the error measures are dependent on the time offset of the prediction. Predicting a trajectory one second into the future will more likely yield better results than predicting the same trajectory one hour into the future. The obvious solution to this problem is to provide the error (or accuracy) for a range of time offsets like in [Gidófalvi et al., 2011, Ikeda et al., 2013, Jeung et al., 2010], improving the comparability of different prediction algorithms.
A similar, yet different, approach is to always use the final position of a test trajectory as the prediction goal and present increasingly longer fractions of the trajectory as an observation to the prediction algorithm as applied in [Bennewitz, 2004, Vasquez and Fraichard, 2004]. An error is calculated for every fraction of the trajectory resulting in a plot of the prediction error over the fraction of the trajectory. By using the final position as the prediction target, it inherently accounts for different time offsets since the duration of the unobserved part of the trajectory decreases with the increasing fraction of the trajectory presented to the prediction algorithm. Unfortunately, this approach can only be applied if the trajectories of the test dataset are almost the same length or if the error is normalized to the length of the trajectory. If the unnormalized error is used, a dataset with short trajectories would result in an error plot different from a dataset with long trajectories and would impede a comparison.

6.2.5 Prediction ROC and AUC

Most of the error measures found in literature for evaluating a prediction algorithm assume that the algorithms only provide one position as the most probable result. Thus, they do not account for multiple options. For example, if a prediction algorithm were to calculate two likely positions for a given trajectory - one with a probability of 49.9% $1m$ off of the ground truth and a second one with 50.1% but $100m$ away from the real position - the Euclidean distance error would result in an error of $100m$ even if the second option, with almost the same probability, is much more accurate.

The presented prediction algorithm not only provides a few options as a result but also a probability distribution over every possible state. For a prediction with a big time offset, it is likely that the probability distribution disperses across a large area with several almost equally possible states. Using a traditional error measure on such a prediction would result in misleading interpretations. In order to account for predictions providing a probability distribution, a new error measure needs to be found.

In the field of computer vision, it is common to evaluate re-identification algorithms on their receiver operating characteristic (ROC) (e.g., [Eisenbach et al., 2012]). A re-identification algorithm usually takes an unknown sample and compares it to a set of N known images. A score representing the similarity to the sample is calculated for every image. For every sample of a test dataset, the scores are calculated and the known images are sorted accordingly in descending order. This results in a sorted list for every sample, and the ROC is obtained by calculating the relative frequency (or true positive rate - TPR) for each rank $n = 1, \ldots, N$ that the correct image is within the first n items of the lists. Thus, the ROC can be used to assess the probability of having a correct match (TPR) as a function of the false acceptance rate (FAR).

The proposed prediction algorithm can be seen as a re-identification system for positions. Its prediction result is basically a list of nodes with the probability to encounter an observed trajectory at their respective position for any given time. In the analogy of image re-identification, the probabilities correspond to the scores, and the set of nodes resembles the set of known images. By sorting the list of nodes according to their probability, an ROC can be calculated in the same way it is obtained in the re-identification example:

For each sample $s = 1, \ldots, S$ of a test dataset, a prediction $o^{s,t}(n)$ for every node $n = 1, \ldots, N$ and all time steps $t = 0, \ldots, t_{\max}$ up to a maximum time $t_{\max}$ is calculated, and the node $n_{\text{gt}}^{s,t}$ which lies next to the ground truth is determined. Afterward, the nodes n are sorted according to their observational probability $o^{s,t}(n)$ in descending order resulting in a new, sorted list $\hat{\mathbf{N}}^{s,t} = \left\{\hat{n}_1^{s,t}, \ldots, \hat{n}_N^{s,t}\right\}$. The ROC is constructed by calculating the relative frequency $\text{TPR}(m,t)$ of the occurrence of the correct node $n_{\text{gt}}^{s,t}$ (or true positive rate) in the first m nodes of the sorted lists for every $m = 1, \ldots, N$

and every sample $s = 1, \dots, S$ as formalized in Eq. 6.1. The false acceptance rate is calculated by $FAR(m) = m/N$.
As becomes apparent in the equations, the ROC is dependent on the current time delta t. It transfers the error measure from a common two dimensional form (e.g., Euclidean distance error over time delta) to a three dimensional plot (true positive rate over time delta and false acceptance rate). At first glance, the complex error surface may not be understood intuitively, but it enables a comparison to other error measures. For a cut along the false acceptance rate of $1/S$, which corresponds to the leftmost FAR and the $\text{TPR}\,(1,t)$ curve, it becomes comparable to the lock-in accuracy with a range of approximately half of the mean node distance.

$$\text{TPR}\,(m,t) = \frac{\sum_s^S \text{isIn}\left(n_{\text{gt}}^{s,t}, \hat{\mathbf{N}}^{s,t}, m\right)}{S} \tag{6.1}$$

with

$$\hat{\mathbf{N}}^{s,t} = \left\{\hat{n}_1^{s,t}, \dots, \hat{n}_N^{s,t}\right\}$$

and

$$\text{isIn}\left(n_{\text{gt}}^{s,t}, \hat{\mathbf{N}}^{s,t}, m\right) = \begin{cases} 1 & \text{if } n_{\text{gt}}^{s,t} \in \left\{\hat{n}_1^{s,t}, \dots, \hat{n}_m^{s,t}\right\} \\ 0 & \text{else} \end{cases}$$

A common measure to condense an ROC to a single value is the area under the curve (AUC), which integrates the TPR along the FAR. A value of one would indicate a perfect prediction algorithm whereas a value of 0.5 corresponds to an algorithm which merely takes a random guess. Since the prediction ROC (PROC) is dependent on time delta t, the prediction AUC (PAUC) is also a function of t, gradually decreasing from one to 0.5 over time. The PROC may yield more detailed information about the quality of the prediction results and allows for a comparison to more common error measures, but the PAUC provides a more intuitive assessment of the prediction method.

6.2.6 Conclusion

As shown in this chapter, the error measure for a prediction algorithm is strongly dependent on the used methods and its applications. For evaluating prediction algorithms operating in the continuous state space, the Euclidean distance error may seem like a good and computationally easy error measure since it provides a direct interpretation of their capabilities. However, it can only be used for a profound comparison of different algorithms if the prediction algorithms are evaluated on the same dataset.

A more eligible measure for comparing different datasets is the lock-in accuracy. A good choice of its lock-in range is crucial for a proper comparison but it is easier to choose than to compare two Euclidean distance errors. Furthermore, it enables a better assessment of a prediction algorithm applied on structured environments, like road nets or buildings, since obstacles or other spatial restrictions are not accounted for with the Euclidean distance (a mean Euclidean error of $10m$ may seem fair for predicting a person in a building - but only if it is on the same floor and not three levels directly below the person).

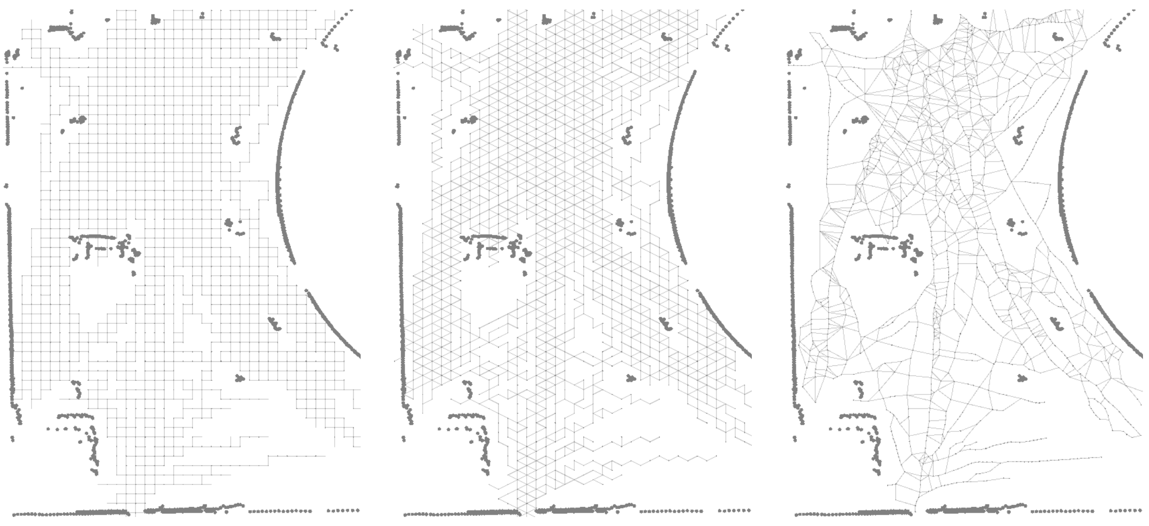
Prediction algorithms operating on a discretized spatial space can be easily compared by the (n^{th}-) next state accuracy, even if they are evaluated on different datasets. The measure is independent from the spatial scales, but it also does not take both the amount of used information and the prediction time offset into account.

In order to factor in the amount of used information and the prediction offset at the same time, it is advisable to always use the last known points of the test trajectories as the ground truths of the prediction and not to record the error (i.e., the Euclidean distance error or the lock-in accuracy) for different time offsets. The fraction of the trajectories which gets presented to the prediction algorithm is varied, and the errors are calculated for each fraction. The resulting plot makes it easier to compare two prediction methods since it intrinsically normalizes the time scale.

All of these error measures and their modifications are only able to account for one prediction result. Since the presented prediction framework provides a probability distribution as the result instead of a distinct state, those measures are only applicable to compare it to different algorithms. The PROC and PAUC are better suited to evaluate the presented framework because they are able to take not only the most probable prediction into account but all other states and their probabilities, too. Therefore, the PROC and PAUC are used for evaluating the experiments in Section 6.3 and 6.4, whereas the respectively used error measure is applied to compare the proposed prediction framework to the state of the art in Section 6.5.

6.3 Topological Representation

In order to evaluate the benefit of a Mean-Shift-based topology, as introduced in Section 3.3, it was compared to grid-based representations with a rectangular and a sixfold-neighborhood pattern (see Section 3.2) on the Humboldt dataset. A time step of $\Delta t = 0.1s$ and a time horizon of $t_\Delta = 20s$ was chosen for prediction. The Markov-trees have a depth of five and used the Chebyshev-criterion for a margin of $\epsilon = 30\%$ and a confidence greater than 75% (see Eq. 4.2 in Section 4.2). The different graphs are shown in Fig. 6.4 and their individual PROC plots and PAUC diagrams

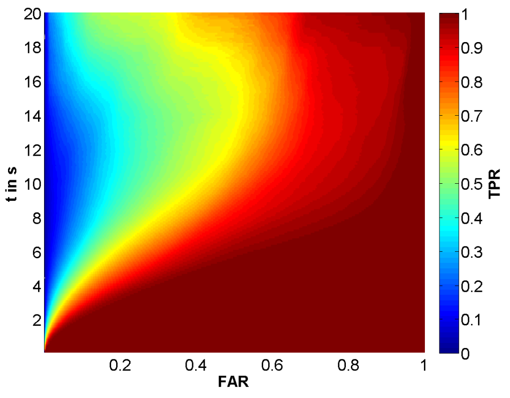


(a) Rectangular (b) Sixfold neighborhood (c) Mean-Shift clustered

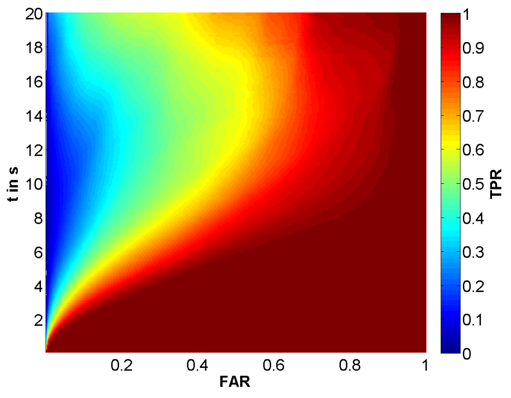
Figure 6.4: *Graphs with three different methods for representing the topology. Walls and static obstacles are depicted as dark gray dots, and the graphs are plotted with small, black nodes and gray edges. A base length of $b = 0.5m$ was used for the rectangular and the sixfold neighborhood graph, and a kernel width of $\gamma_a = 0.15m$ was utilized in the graph created with the Mean-Shift algorithm.*

are depicted in Fig. 6.5.

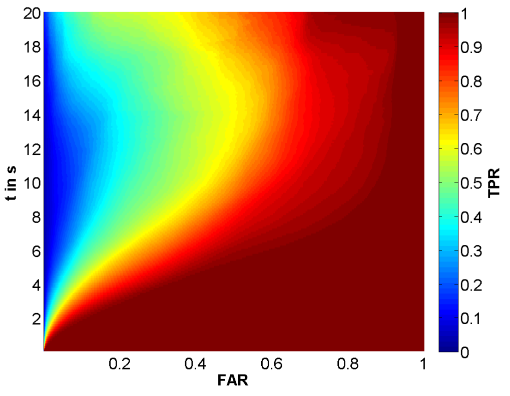
At first glance, there is no apparent difference between the PROC plots. If, for example, an 80% probability of obtaining the correct prediction result is desired, all topological representations provide almost the same false acceptance rate over time: following the $TPR = 0.8$ contour, it starts at an $FAR = 0$ for the time $t = 0s$ (since the result is always perfect for a time horizon of zero seconds), declining to an $FAR \approx 0.65$ for $t = 13s$ (i.e., an 80% prediction accuracy can be achieved by checking the most probable 65% of all nodes of the graph), and rising to an $FAR \approx 0.55$ till $t = 20s$ (for $t = 20s$, $\sim$ 10% less nodes than for $t = 13s$ must be checked in order to achieve the desired accuracy of 80%). It is worth noting that the prediction performance rises for $t > 13s$ which is due to the selected scenario: almost all trajectories begin and end in some distinct areas in the Humboldt dataset. Several students left the coverage of the LIDAR sensors through the main exit doors, the restroom door, or the entrance door to the main auditorium. Since the probability distribution tends to first disperse across the state space and then accumulates at these main exit regions, it provides a more diffuse prediction for the intermediate states of the trajectories than for the final point. This behavior can also be seen in the PAUC plots in Fig. 6.5(d). Furthermore, it confirms in a more intuitive modality that all three graphs provide almost equally positive prediction results.



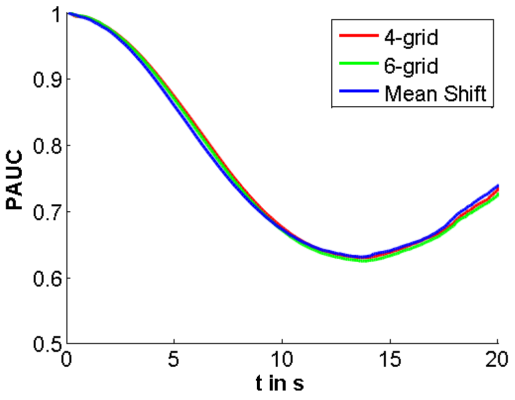
(a) PROC of rectangular graph



(b) PROC of sixfold neighborhood graph



(c) PROC of Mean-Shift clustered graph

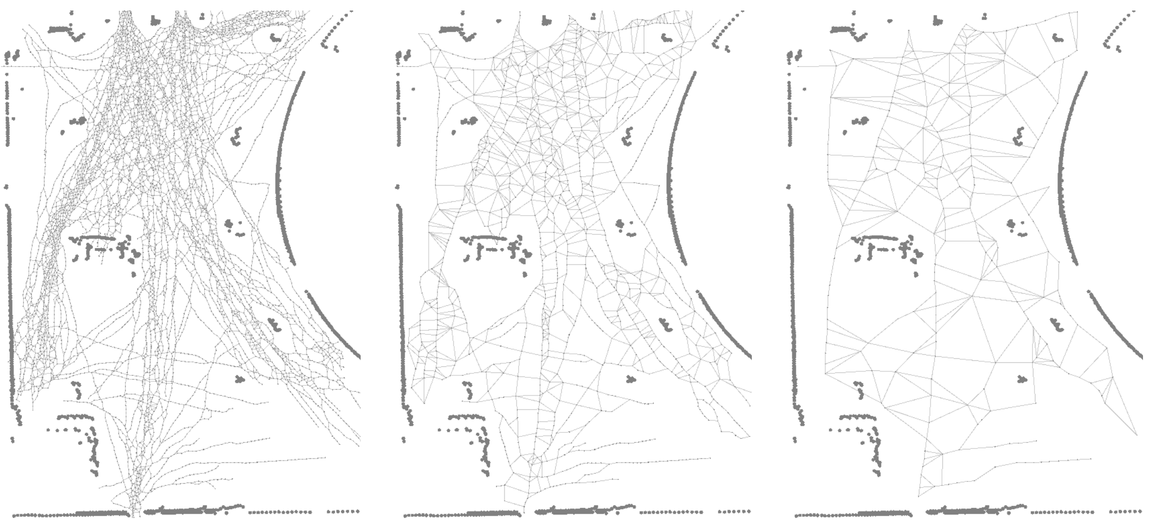


(d) PAUC of all graphs

Figure 6.5: *Prediction results for grid-based and Mean-Shift clustered graphs. As can be seen in the PROC and PAUC diagrams, all three graphs perform almost equally well.*

In addition to the methodological advantages as stated in Section 3.2, the reason to prefer the cluster-based representation over the grid-based topologies lies in its reduced memory requirements: Both regular grids used in the experiment were spanned by $2,320$ nodes, whereas the Mean-Shift clustering provided us with a net comprising only $1,220$ nodes - almost half as many. With a graph of fewer nodes, the prediction algorithm creates fewer flows (as explained in Section 5.1) for calculating a prediction, thus reducing the computational requirements of the algorithm.

In order to assess the influence of the chosen kernel width for clustering on the prediction accuracy, three nets were created with a kernel width of $\gamma_a = 0.075m$, $\gamma_b = 0.15m$, and $\gamma_c = 0.3m$ according to the algorithm in Section 3.3.4 and 3.4. The topological graphs resulting from the clustering are shown in Fig. 6.6 and their individual PROC plots and PAUC diagrams are depicted in Fig. 6.7. The PROC in Fig. 6.7 are similar to the plots in Fig. 6.5 but with more deviant shapes toward the final time horizon.

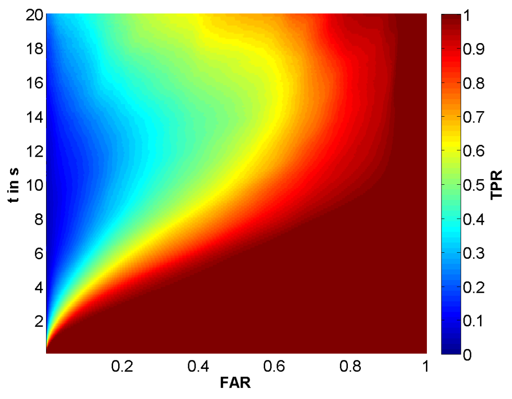


(a) Graph for $\gamma_a = 0.075m$ (b) Graph for $\gamma_b = 0.15m$ (c) Graph for $\gamma_c = 0.3m$

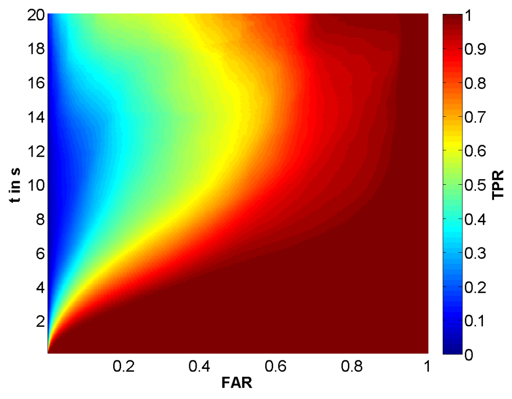
Figure 6.6: *Three topological graphs obtained by the Mean-Shift algorithm with different kernel widths. Walls and static obstacles are depicted as dark gray dots, and the graphs are plotted with small, black nodes and gray edges. It is worth noting that the graph in (a) almost covers every single trajectory of the dataset (please compare to Fig. 6.2) resulting in densely sampled areas close to the exit, whereas the graph in (c) leaves large patches uncovered and has some edges longer than* $3m$.

The graph with a kernel of $\gamma_b = 0.15m$ gives the best results for all time steps, the graph with $\gamma_c = 0.3m$ provides the least accurate predictions from the beginning on, and the kernel of $\gamma_a = 0.075m$ performs in between the other two. The reason for the comparably bad performance of the kernel width $\gamma_c = 0.3m$ can be seen in Fig. 6.6(c): large areas with some, but not many, trajectories are not always covered by nodes or are only represented by long edges. It leads to ambiguities if a test trajectory is observed in these areas and it might get assigned to the wrong sequence of nodes (as explained in Section 4.1 and Fig. 4.1) thus resulting in a wrong prediction. The small kernel width of $\gamma_a = 0.075m$ provides a better result, but it covers the other extreme: for the available number of training trajectories, the state space is sampled too densely. A lot of transitions are only covered by individual observations resulting in almost no generalization of the learned data.

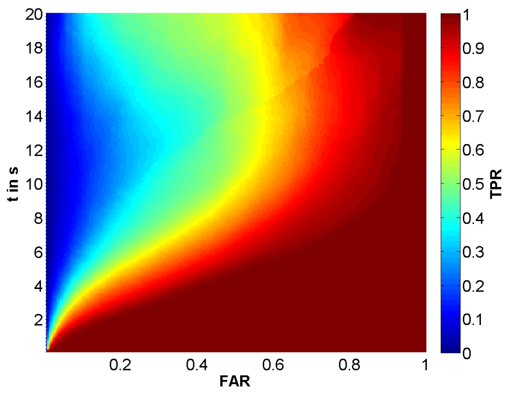
The experiment shows that the kernel width is an important parameter and needs to be tuned to the current application at hand. However, Fig. 6.5 also shows that the kernel width is insensitive to small deviations from its optimum: the results are not as good but are still on a comparable level. A rule of thumb for a suitable kernel width is to determine the desired minimum distance between nodes in dense areas of the state space and use it as the width for creating the graph with the Mean-Shift approach.



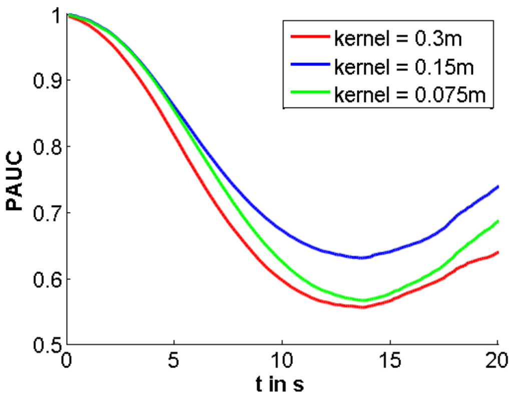
(a) PROC of graph with $\gamma_a = 0.075m$



(b) PROC of graph with $\gamma_b = 0.15m$



(c) PROC of graph with $\gamma_c = 0.3m$



(d) PAUC of all graphs

Figure 6.7: *Prediction results for Mean-Shift clustered graphs with three different kernel widths. While they perform almost similar in the first seconds, their accuracy diverges for further time steps. The kernel of $\gamma_b = 0.15m$ performs best on every time step.*

6.4 Precision Loss in Frequency Domain

Section 5.1 described several implementations for the main prediction algorithm. The solution using the frequency domain, as explained in Section 5.1.3, provides a feasible and fast approach but the iterative solution, introduced in Section 5.1.4, is expected to be even faster. The latter iteratively calculates the timesteps in succession, whereas the former approximates all flows for all timesteps with a decreasing residual error for each additional iteration. In applications where a large time horizon is desired but imprecisions are acceptable to a certain extent, it may be expedient to use the approach using the frequency domain instead of the iterative solution for saving computational resources.

In order to ascertain the computational differences between the two approaches, their prediction accuracy and computational demands are compared on the Humboldt dataset. A timestep of $\Delta t = 0.1s$ with a total time horizon of $t_\Delta = 102.4s$ was used for prediction in order to utilize optimized Fast Fourier Transformations on a vector length of $l = 2^{10} = 1024$. The iterative approach inherently needs a total of $l = 1024$ iterations for predicting the full time horizon and the algorithm utilizing the frequency domain was also configured to calculate the full $l = 1024$ iterations in order to obtain the best result for comparison.

The results of both algorithms are shown in Fig. 6.8. The PROC and PAUC plots show a time horizon of $t = 20s$ instead of the full horizon of $t_\Delta = 102.4s$ since only a few of the test trajectories were longer thus preventing a statistically sound evaluation for $t > 20s$. The PROC plots in Fig. 6.8(a) and 6.8(b) do not show any significant difference but a slightly decreasing performance of the solution using the frequency domain compared to the iterative approach is apparent in the PAUC graphs in Fig. 6.8(c). This is believed to result from residual errors of the initial and final Fourier transformations but also to originate from floating point errors especially on the higher frequencies during the main iterations. Still, the overall difference in the results of both approaches is marginal.

In order to evaluate the computational complexity of both approaches, the number of calculations involved in each iteration was counted on one randomly chosen prediction task. Since both algorithms use very similar processing schemes (as can be seen by comparing Fig. 5.4 and Fig. 5.5 in Section 5.1) with the specific calculation of the flows as their main difference, only the multiply and add operations (line 13 in Fig. 5.4 and line 10 in Fig. 5.5) were counted in this experiment. As can be seen in the results in Figure 6.8(d), the algorithm using frequency domain needs approximately one hundred times more calculations than the iterative approach. The former needs $1,024$ for the first iteration, which relates to one element-wise multiplication of a flow vector with a time distribution vector in frequency domain. The latter needs five calculations since the involved transitional time distribution is merely of the length $i = 5$ (which refers to the index i in line 10 in Fig. 5.5). Since all $1,024$ elements of the transformed transitional time distribution vectors have to be multiplied element-wise in the frequency domain for calculating the next iteration of a flow, the iterative approach is in advantage of processing much shorter distribution vectors, which are rarely longer than 20 elements, thus giving the observed speed-up. The first iteration of this specific prediction created only one new flow, whereas the following iterations incorporated multiple flows thus increasing the computational needs until the number of flows saturate after approximately 100 iterations.

By comparing the mean time instead of the number of computations needed for predicting the test trajectories by the frequency approach and the iterative algorithm, a more practical assessment of their computational complexity can be achieved. A

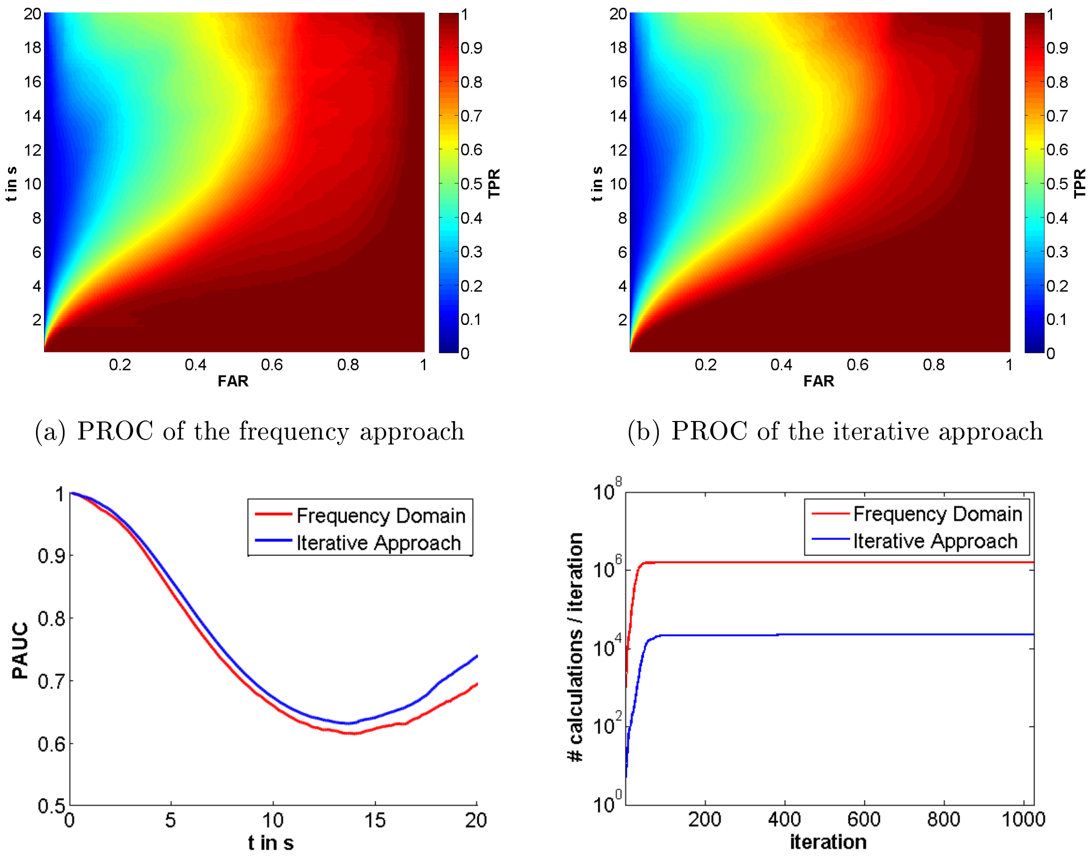


Figure 6.8: *Prediction results of the iterative algorithm and the approach using the frequency domain. The PROC are similar in shape but the PAUC plots show a slight difference in their performance: the solution using the frequency domain becomes gradually inferior to the iterative solution with an increasing time horizon. Also, the number of required computations is approximately two magnitudes higher for the frequency domain than for the iterative approach.*

prediction with the iterative approach took $3.8s$ on average, whereas $37s$ are needed when utilizing the frequency domain (evaluated on all test trajectories on an Intel(R) Core(TM) i7 quad core processor running at $2.67GHz$). At first glance, this contradicts the measurements of Fig. 6.8(d), which would imply a factor of 100 in contrast to the measured speed-up of ~ 10. However, the element-wise multiplications of the FFT vectors heavily exploit SIMD instructions of the CPU whereas the iterative approach does not get that much of a benefit. Furthermore, the overhead of iterating through the flows, nodes, and neighbors is also not incorporated in the measurements of Fig. 6.8(d).

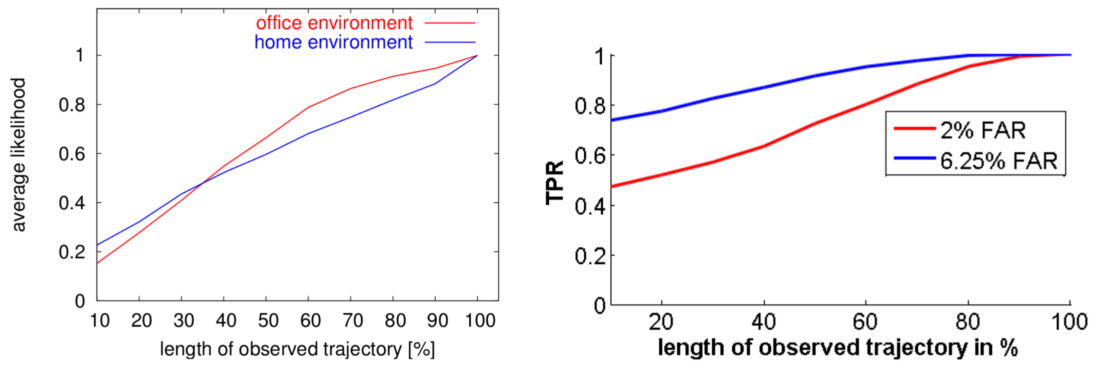
Nevertheless, a speed-up factor of approximately 10 between the frequency approach and the iterative algorithm limits the utility of the frequency domain. In order to benefit from a faster but less accurate prediction, the algorithm should process less than a tenth of the maximum number of iterations, but this would yield a very rough estimate of the probability distribution at best. Therefore, the iterative approach is recommended as explained in Section 5.1.4 in order to obtain the best results in a reasonable amount of time.

6.5 Comparison to the State of the Art

Chapter 2 gave an overview of established prediction methods and pointed out three promising algorithms which may be used for a versatile prediction framework. In order to compare the proposed prediction framework to the state of the art, this section individually contrasts the methods of [Bennewitz, 2004], [Vasquez Govea, 2007], and [Ikeda et al., 2013] with the prediction framework presented in Chapters 3 to 5. The error measures used in the following subsections were obtained by using the $1,000$ longest trajectories of the Beijing test dataset (see Section 6.1) on the topological graph shown in Fig. 6.3(b). The graph was obtained by clustering the training data with a kernel width of $\gamma = 25m$ in order to achieve a level of detail on the scale of individual roads. It resulted in a graph comprised of $14,272$ nodes and $57,067$ edges. The predictions were conducted with a timestep of $\Delta t = 1s$ and a total time horizon of $t_\Delta = 1,200s$ to provide a tradeoff between computational requirements, temporal resolution, and an adequate time horizon for a prediction of car movements. The Markov-trees have a depth of five and used the Chebyshev-criterion for a margin of $\epsilon = 10\%$ with a confidence greater than 75% (see Eq. 4.2 in Section 4.2).

6.5.1 Comparison to [Bennewitz, 2004]

Despite using a self created dataset, the evaluation of the prediction algorithm presented in [Bennewitz, 2004] employs an error measure which enables a simple comparison to the proposed prediction algorithm. For two different datasets (one obtained in an office environment and another in a home environment), the likelihood of mapping a test trajectory to the correct motion pattern was measured for increasingly longer observations of the test trajectories (as shown in Fig. 6.9(a)). By interpreting the prediction task as recognizing the correct motion pattern (i.e., the future trajectory up to a certain goal) out of the set of the n known motion pattern, the error measure can be seen as the true positive rate (TPR) of predicting the correct final goal of a test trajectory for different lengths of observations for a false acceptance rate (FAR) of $1/n$.



(a) Results of [Bennewitz, 2004] (b) TPR of the proposed framework

Figure 6.9: *Comparison of the proposed prediction method with the algorithm of [*Bennewitz*, 2004]. In order to enable a comparison, the true positive rate (TPR) for a false acceptance rate (FAR) of* 2% *for the office environment and an FAR of* 6.25% *for the home environment was recorded for increasingly longer fractions of the Beijing test trajectories as explained in Section 6.5.1. A comparison of both graphs show that the proposed prediction framework performs considerably better than the pattern based algorithm of [*Bennewitz*, 2004].*

In order to measure the true positive rate on the Beijing dataset utilizing the proposed prediction framework, $1,200$ second long sections of the test trajectories and the nodes in the topological graph corresponding to the endpoint of the sections were taken as the ground truth. Afterwards, increasingly longer fractions of these sections were presented to the prediction framework and the TPR of the prediction results was calculated for the same FAR used in [Bennewitz, 2004] (please see Section 6.2.5 for the details on calculating the FAR and TPR). The EM algorithm in [Bennewitz, 2004] determined a total of 49 motion patterns for the office environment and 16 patterns for the home environment. Thus, the likelihood of matching the correct motion pattern relates to an FAR of $1/49 \approx 2\%$ for the office environment and an FAR of $1/16 = 6.25\%$ for the home environment. The corresponding results of the proposed prediction framework on the Beijing test trajectories are shown in Fig. 6.9(b). A comparison with Fig. 6.9(a) shows that it performs significantly better than the pattern based algorithm of [Bennewitz, 2004], especially for short observations. However, it should be mentioned that the EM algorithm correlates the full observation to all known patterns and the increasing likelihood for longer trajectories is related to the additional knowledge of the motion. In order to limit the memory requirements of the Markov-trees (see Section 4.2), the proposed prediction framework cuts the input to the last six nodes to which the observed trajectory corresponds. Since the last six nodes relate to fewer than ten percent of the used sections of the test trajectories, the higher TPR for increasingly longer observations results from the fact that the last point of the observation gets closer to the endpoint, thus reducing the time horizon and the complexity for the prediction instead of adding useful knowledge.

It cannot be stressed enough that the results in Fig. 6.9 are obtained on different datasets with different motion dynamics. Unfortunately, a reimplementation of the prediction method of [Bennewitz, 2004] to evaluate it on the Beijing dataset for a direct comparison is not useful due to the enormous computational complexity of determining the number of model components and the motion pattern on all 2.05 million training trajectories. Although the experiment in Fig. 6.9(b) was tuned to roughly correlate to the experiment in [Bennewitz, 2004], the results of the comparison should be assessed with caution.

6.5.2 Comparison to [Vasquez Govea, 2007]

The prediction algorithm of [Vasquez Govea, 2007], called GHMM, bears resemblance to the proposed prediction framework. It utilizes a similar topological graph for sampling the state space and uses a probabilistic model to predict the following state to a given observation with a first order Markov chain while providing an algorithm to update both models on-line. As promising as the thesis in [Vasquez Govea, 2007] appears to be, it does not provide experimental results which may be used for comparing it to other state of the art methods. On several undisclosed datasets, it gives a mean distance error for an unknown time horizon as the only error measure related to the prediction accuracy. A reimplementation and evaluation on the Beijing dataset was abandoned due to the estimated time needed for training and prediction. Since the idea of the topological graph in [Vasquez Govea, 2007] is similar to the proposed one, the final graph can be assumed to be of a similar size as in Fig. 6.3(b) with $14,272$ nodes and $57,067$ edges.

The computational requirements of the model training and the prediction was evaluated in the related work [Vasquez et al., 2009] on an Intel(R) Core(TM) 2 Duo, and a linear relation of the computing time to the number of edges was discovered. According to [Vasquez et al., 2009], learning one point of the training data requires $4\mu s$ per edge. For the full Beijing training set with fifty-eight million data points and the estimated number of fifty-seven thousand edges, the time required to train the model would add up to 77 days ($\frac{4\mu s}{\text{edges}\cdot\text{points}} \cdot 57,000\text{edges} \cdot 58,000,000\text{points} \cdot 0.5$). Even on a faster computer (e.g., the Intel(R) Core(TM) i7 quad core processor running at $2.67GHz$ involved in the experiments of this thesis would need 29 days), it would still need weeks to train the prediction model of GHMM. Four weeks of training might be acceptable, but the evaluation of the reduced Beijing test dataset would require even more time. Given the results of [Vasquez et al., 2009], the prediction of one observation would require $21\mu s$ per edge. Unfortunately, the time horizon for the prediction

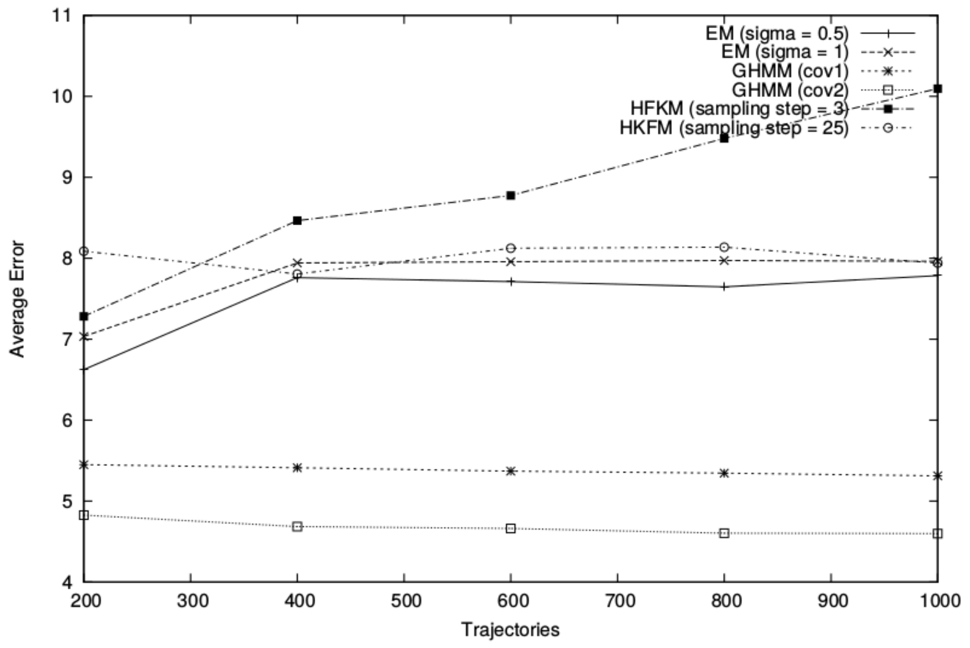


Figure 6.10: *Comparison of the GHMM prediction method of [*Vasquez Govea*, 2007] with the EM algorithm of [*Bennewitz*, 2004] and the HFKM method of [*Hu *et al., 2006] for different sizes of the simulated training dataset, as published in [*Vasquez *et al., 2009]. As mentioned in 6.5.2, the GHMM algorithm provides a mean distance error which is approximately* 40% *better than the EM algorithm.*

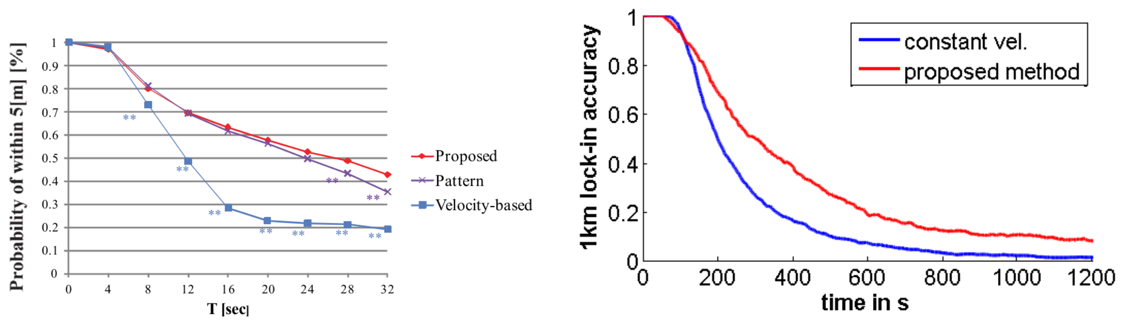
was not mentioned in either [Vasquez Govea, 2007] nor [Vasquez et al., 2009] but it is believed to be 15 steps. Using this information, it is estimated to require 370 days ($\frac{21\mu s}{\text{edges}\cdot\text{points}} \cdot 57,000\text{edges} \cdot 667,000\text{points} \cdot \frac{1,200\text{steps}}{15\text{steps}} \cdot 0.5$) for evaluating the prediction method on the reduced Beijing test dataset for a time horizon of 20 minutes (or 138 days on the Intel(R) Core(TM) i7). Due to these preliminary estimations, a reimplementation and evaluation on the Beijing dataset is believed to be unfeasible and an indirect comparison was chosen instead. Fortunately, the GHMM algorithm was also published with more insightful experiments using an artificial dataset in [Vasquez et al., 2009]. As shown in Fig. 6.10, it was compared to the methods of [Bennewitz, 2004] (labeled EM) and [Hu et al., 2006] (plotted as HFKM) by evaluating their average distance error for different sizes of the training dataset. By taking the best results of Fig. 6.10, the mean prediction error of GHMM is approximately 60% of the error of the EM algorithm. Since Section 6.5.1 also shows significantly better results of the proposed prediction framework compared to the EM method of [Bennewitz, 2004], it can be assumed that the proposed prediction framework performs at least equal to the algorithm of [Vasquez Govea, 2007]. Both publications of the GHMM algorithm also mentioned a decrease in prediction accuracy in the event of broken training trajectories. Due to this problem and the advantage that the proposed prediction framework can not only utilize first order Markov chains but also chains of an arbitrary length, it is believed that it would perform even better than GHMM in a direct comparison on real-world data.

6.5.3 Comparison to [Ikeda et al., 2013]

A prediction algorithm similar to the proposed prediction framework was presented in [Ikeda et al., 2013]. It shares both the idea of the topological graph and a probabilistic transition model but it is strongly focused on the motion mechanics of pedestrians and the correlated concept of sub-goals. Sub-goals are successive points in space that people tend to use for navigating through an environment towards a final goal by heading to the closest visible sub-goal until they see the next one. Since the concept needs spatial information for calculating the lines of sight and tries to model pedestrian behavior, the prediction method of [Ikeda et al., 2013] is highly limited in their application and not usable on the Beijing dataset. Nevertheless, it is once again possible to compare the proposed prediction framework to the sub-goal algorithm indirectly. As shown in Fig. 6.11(a), its performance was compared using the lock-in accuracy with a lock-in range of $d_s = 5m$ (see Section 6.2.2) on an undisclosed dataset with a common constant velocity extrapolation model (CVM).
In order to find a suitable lock-in range d_b for the evaluation on the Beijing dataset, both the maximum time horizon and the speed of the cars were taken into account. In [Ikeda et al., 2013], a time horizon of $t_{\Delta,s} = 32s$ was chosen and the average human walking speed was $v_s = 1.4m/s$ according to [Browning et al., 2006, Mohler et al., 2007]. On the Beijing dataset, a time horizon of $t_{\Delta,b} = 1,200s$ was selected and the average car speed on the test dataset was $v_b = 7.4m/s$. Using Eq. 6.2, an equivalent lock-in range of $d_b = 991m \approx 1km$ was used for evaluating the proposed prediction framework on the Beijing test dataset.

$$d_b = \frac{v_b}{v_s} \cdot \frac{t_{\Delta,b}}{t_{\Delta,s}} \cdot d_s \tag{6.2}$$

A way of understanding the equivalent lock-in range according to Eq. 6.2 is to use a linear motion as a simple example. If the assumed velocity of a motion is off by a certain factor, the Euclidean distance error increases linearly with the time horizon. In order to obtain the same lock-in error on two different time horizons t_a and t_b, the lock-in range d_a needs to be multiplied by the proportion of the time horizons $d_b = \frac{t_b}{t_a} \cdot d_a$. The same holds for scaling the velocity: the Euclidean distance error increases linearly with the velocity. For instance, if a person moves with a velocity of $v_a = 1m/s$ and the movement is predicted to be twice as fast, a lock-in range of



(a) Results of [Ikeda et al., 2013] (b) Lock-in accuracy of the proposed framework

Figure 6.11: *Comparison of the proposed prediction method with the graph based algorithm of [Ikeda et al., 2013]. Despite the intricacy involved in the comparison, as explained in Section 6.5.3, both plots show that the proposed method in (b) performs similarly to the method of [Ikeda et al., 2013] in (a). Compared to a simple constant velocity model, it provides significantly better results towards the end of the time scale. Benevolently assuming that the the ordinate of the plot in (a) is not scaled in percent as stated by the label of the y-axis in the figure of [Ikeda et al., 2013], the proposed algorithm in (b) does not provide an accuracy as good as in (a) for longer time horizons. Yet, due to the factors elucidated in Section 6.5.3, the performance of the proposed prediction method can be considered almost equal to the results of [Ikeda et al., 2013].*

$d_a = 10m$ is left after $t_a = 10s$. If a car is moving with $v_b = 10m/s$ and the movement is predicted to be twice as fast, a lock-in range of $d_a = 10m$ is left after $t_b = 1s$. In order to get the same lock-in error, the lock-in range d_a needs to be multiplied by the velocities proportion $d_b = \frac{v_b}{v_a} \cdot d_a$ as well. Combining both relations results in Eq. 6.2.

The comparison of the proposed prediction framework to the constant velocity model on the Beijing test dataset with a lock-in range of $1km$ is shown in Fig. 6.11(b). Taking CVM (blue plot) as a reference, it can be seen that the proposed prediction framework performs in a similar way to the sub-goal algorithm. For example, with CVM crossing the 50% mark (for $t = 12s$ in Fig. 6.11(a) and $t = 199s$ in Fig. 6.11(b)), both prediction algorithms still provide an accuracy of 70%. Towards the end of the time horizon with $t = 32s$, the sub-goal method is able to provide a 43% accuracy while CVM achieves 19%. The same holds for the proposed prediction framework for $t = 367s$: it was able to predict 42% of the test trajectories within the lock-in range, whereas CVM accomplished 19%. Towards the end of the time horizon in Fig. 6.11(b), the constant velocity model is failing with less than 2% accuracy while the proposed prediction framework is still able to predict almost 10% the test trajectories correctly.

Unfortunately, the experimental setup in [Ikeda et al., 2013] covers an area which does not include many junctions and thus favors the constant velocity model. It would be interesting to see the performance of the sub-goal algorithm on areas with

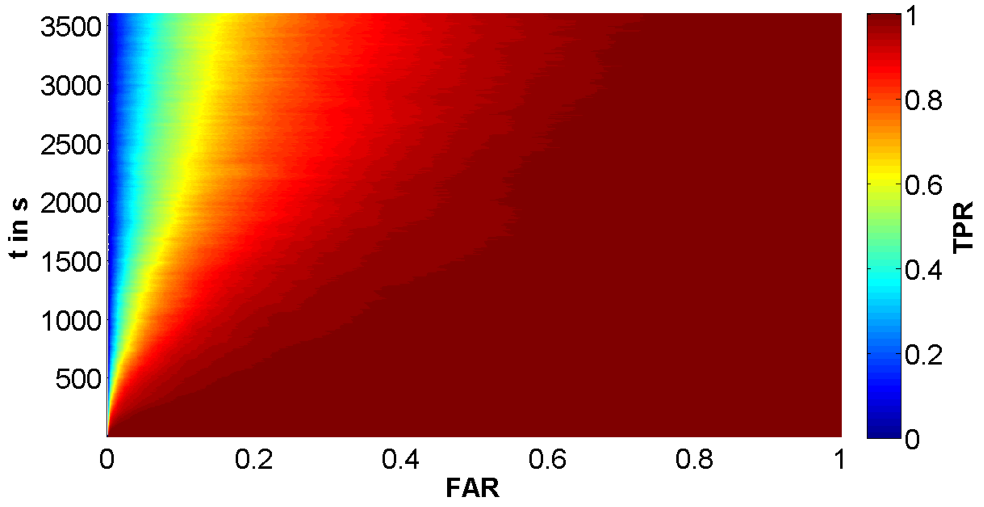


Figure 6.12: *Results of the proposed prediction framework on the Beijing test dataset in form of the PROC metric. As can be seen, the proposed prediction framework can provide a true positive rate of at least* 50% *for a false acceptance rate of* $\sim$ 10% *for every time step up to one hour.*

more branches and crossings. Nevertheless, it can at least be stated that the proposed prediction framework performs in a similar manner to the sub-goal algorithm while providing a much more versatile approach without involving model specific assumptions.

6.6 Long Term Prediction

The last section showed the difficulties involved in comparing a novel prediction framework to an already existing algorithm whose originator did not concentrate on the evaluation of the prediction performance. An extensive quantitative evaluation using multiple error metrics on the graph of Section 6.5 is presented in this section in order to facilitate the comparison of a new method to this prediction framework. The next state accuracy and the error over trajectory are not considered because they do not provide much useful insight due to both the specific test dataset at hand and the low influence of longer observations on the prediction result as explained in Section 6.5.1. For all experiments, the time horizon was extended to one hour to emphasize the capabilities of the proposed prediction framework and provide a better basis for a comparison.

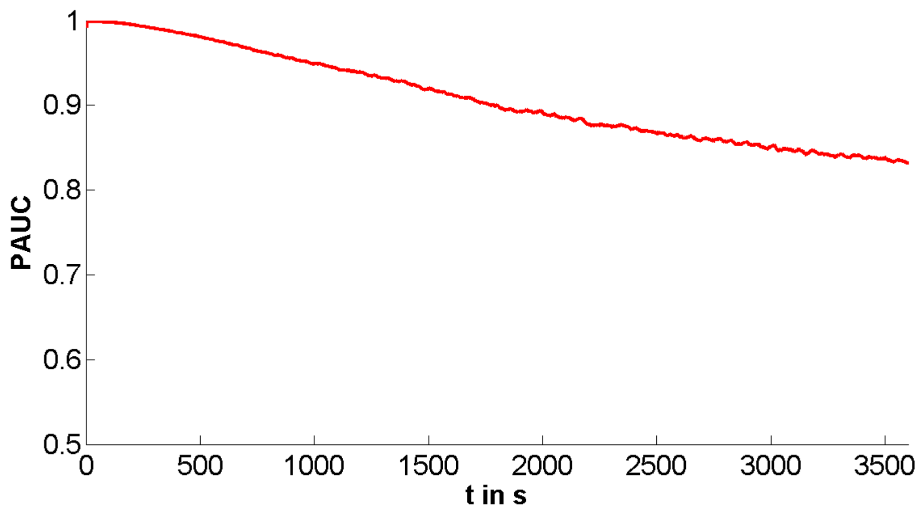


Figure 6.13: *Results of the proposed prediction framework on the Beijing test dataset. As shown in the PAUC diagram, the framework performs remarkably well, even for a time horizon of one hour with an area under curve (AUC) of* > 0.83.

6.6.1 Prediction ROC and AUC

The prediction performance on the Beijing test dataset using PROC and PAUC is shown in Fig. 6.12 and 6.13 respectively. The proposed prediction framework clearly provides an accurate prediction result even for large time horizons. After one hour, the ground truth was covered by 20% of the most probable nodes for more than 60% of the test trajectories (see Fig. 6.12). The area under the curve for $t = 3,600s$ of 0.83 (as seen in Fig. 6.13) emphasizes the fact that the prediction is still far from being a simple guess. It can be assumed that the prediction for an even larger time horizon will still yield a good result. Unfortunately, the Beijing taxi trajectory dataset might not provide a good databasis for an evaluation with a larger time horizon since one hour is already a rather uncommon duration for a taxi ride and might not apply to the usual motion model.

6.6.2 Euclidean Distance Error

The more commonly used Euclidean distance error of the prediction using the node with the highest probability is shown in Fig. 6.14 in blue. It monotonically increases up to a mean error of $7.1km$ for a time horizon of one hour. For an algorithm only providing one distinct point as a prediction, the result can be considered basically useless. But, since the proposed prediction algorithm calculates a probability distribution instead of a single point, a more adequate distance error, the expected Euclidean distance error [Vasquez Govea, 2007] e_{exp}, is also plotted in Fig. 6.14 in red in order

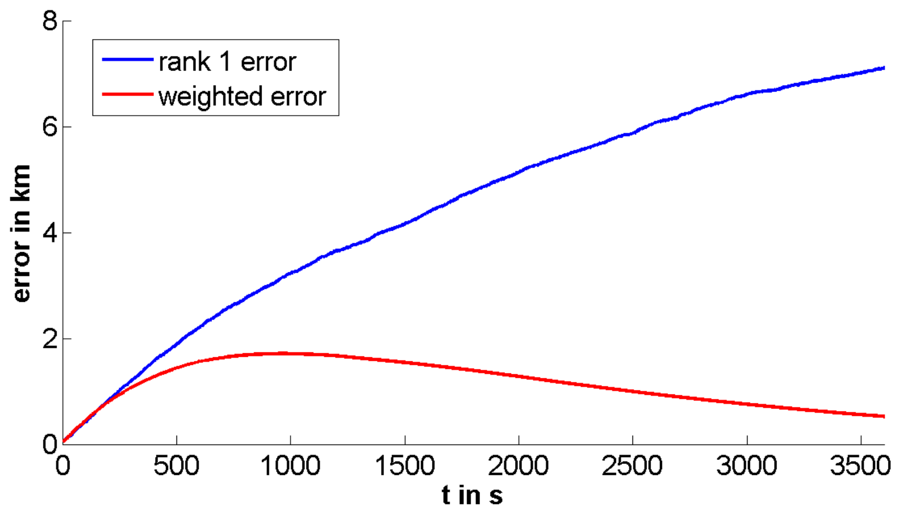


Figure 6.14: *The mean Euclidean distance error of the proposed prediction algorithm is shown in blue, and the expected Euclidean distance error is shown in red for a time horizon of up to one hour. The Euclidean distance error only incorporates the most probable node, whereas the expected Euclidean distance error is the weighted sum of the distances to the ground truth of all nodes (see Eq. 6.3) thus accounting for the whole probability distribution instead of the rank one result. As expected, the Euclidean distance error increases over time whereas the expected error shows a strange behavior by decreasing after t* $\sim 950s$. *The implications of that result are described in further detail in the text below.*

to account for the multimodal prediction results. Every node n contributes to this error as its distance d_n to the ground truth weighted by its probability $o(n)$ according to Eq. 6.3.

$$e_{\exp} = \sum_n d_n \cdot o(n) \tag{6.3}$$

Thus, the distance error of nodes with high probabilities have a stronger influence on the overall error than nodes with a low probability. As shown in Fig. 6.14, the expected error follows the first rank error up to a time horizon of $t \approx 250s$ and then diverges from it until it reaches its maximum of $1.7km$ at $t \approx 950s$ and declines to an error of $0.5km$ for the maximum time horizon of one hour. The plot hints that the prediction result is mostly unimodal up to $t \approx 250s$ since both distance errors are almost the same. Subsequently, new modes emerge and cause the expected error to deviate and slow its ascent since alternate paths and branches get a chance to compensate for less accurate predictions. The most interesting behavior can be seen after $t = 950s$ with the expected error decreasing over time. It contradicts the evaluation using PROC and PAUC in Fig. 6.12 and 6.13 which clearly indicates that the prediction becomes

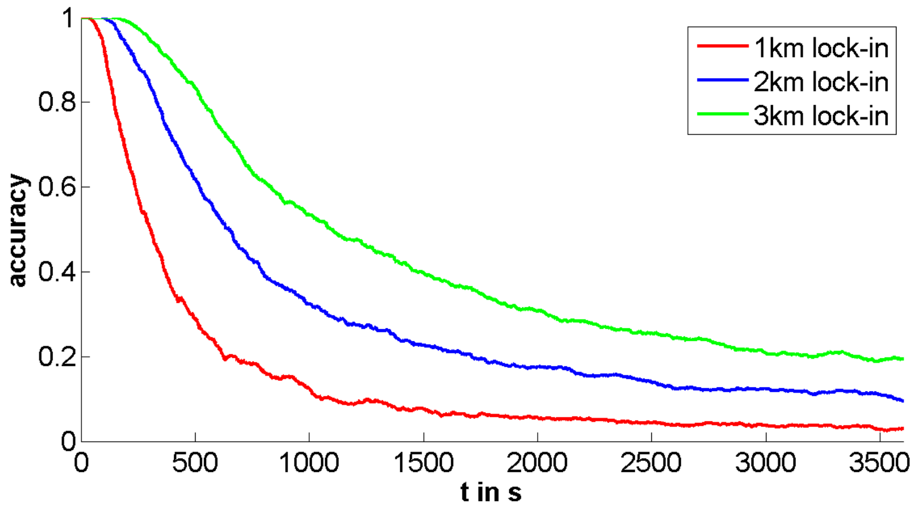


Figure 6.15: *Lock-in error of the proposed prediction framework with different lock-in ranges and a time horizon of up to one hour using the most probable node as the prediction (i.e., rank one lock-in accuracy).*

less accurate with an increasing time horizon. It is a good example that simple error measures such as a weighted distance error do not always provide a good assessment of a prediction algorithm since they do not account for effects like an indistinct probability distribution with a lot of nodes having almost equally low probabilities, which was the cause for that particular decrease in the expected Euclidean distance error in Fig. 6.14.

6.6.3 Lock-In Accuracy

Using lock-in accuracy for evaluation with a lock-in distance of one, two and three kilometers results in the plot of Fig. 6.15. Since the proposed prediction framework provides a probability distribution as the prediction result, only the most probable position was used as the prediction in this evaluation; thus, it resembles the rank one lock-in accuracy. On the given graph with $14,272$ nodes and an area of 930 square kilometers, a random guess would result in accuracies of approximately 0.34‰, 1.4‰, and 3.1‰ for the lock-in ranges of one, two and three kilometers respectively. As shown in Fig. 6.15, the proposed prediction algorithm performs much better than that, even for a time horizon of one hour with accuracies of 2.9%, 9.5%, and 19.4% for the corresponding lock-in ranges. However, it should be noted that the taxi trajectories occasionally stay in one area of Beijing and the probability distribution of the prediction is often almost evenly spread out at further time horizons which helps in achieving high accuracies for lock-in ranges bigger than two kilometers. In order to avoid these effects, much smaller lock-in ranges were chosen in the following evaluations.

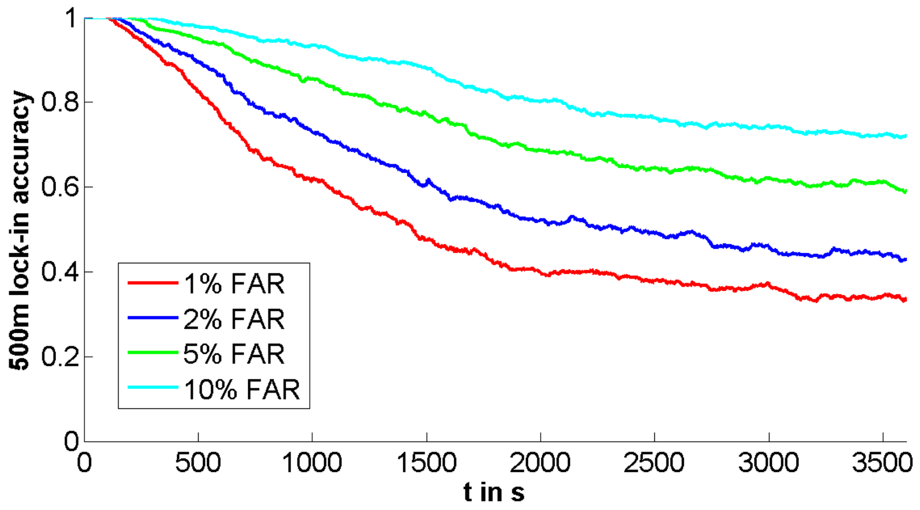


Figure 6.16: *Lock-in accuracy for different false acceptance rates (FAR). Instead of giving the probability that the most probable prediction is within the lock-in range, the first n percent of the nodes ordered by their probabilities is taken and it is assessed if any of them are within the lock-in range. Another interpretation is to first take the most probable node and check if it is within the lock-in range. If it is not, the second most probable node is taken and checked for its distance to the ground truth to be lower than the lock-in range. This procedure is continued until either n percent of the nodes have been evaluated or at least one node is within the range. If a node is found, the prediction is treated as correct and as false otherwise. This evaluation was done for the most probable* 1%, 2%, 5%, *and* 10% *of the nodes and a lock-in range of* $500m$ *for a time horizon of up to one hour. As can be seen,* $\sim 33\%$ *of all test trajectories had at least one of the* 1% *most probable nodes within* $500m$ *of the ground truth after one hour. If a false acceptance rate of* 10% *is taken, even* $\sim 72\%$ *of the test trajectories were predicted correctly using this error measure.*

6.6.4 Extended Lock-In Accuracy

Since the proposed prediction framework does not provide only one prediction but rather a probability distribution over the whole state space, it is expedient to modify the lock-in accuracy to not only take the most probable prediction into account but to incorporate the full distribution. This leads to an ROC like interpretation of the lock-in accuracy which takes a certain percentage (relating to the false acceptance rate) of the most probable nodes and assesses the probability that at least one of them is within the lock-in range. The results can be found in Fig. 6.16 for FARs of 1%, 2%, 5%, and 10% and a lock-in range of $500m$. The plots provide a more insightful evaluation than the first rank lock-in accuracy in Fig. 6.15 since it takes the whole probability distribution into account instead of just the most probable result. It shows that the real position of a taxi in the test dataset can be found after one hour of its last observation with a probability of more than 33% by searching through the 143

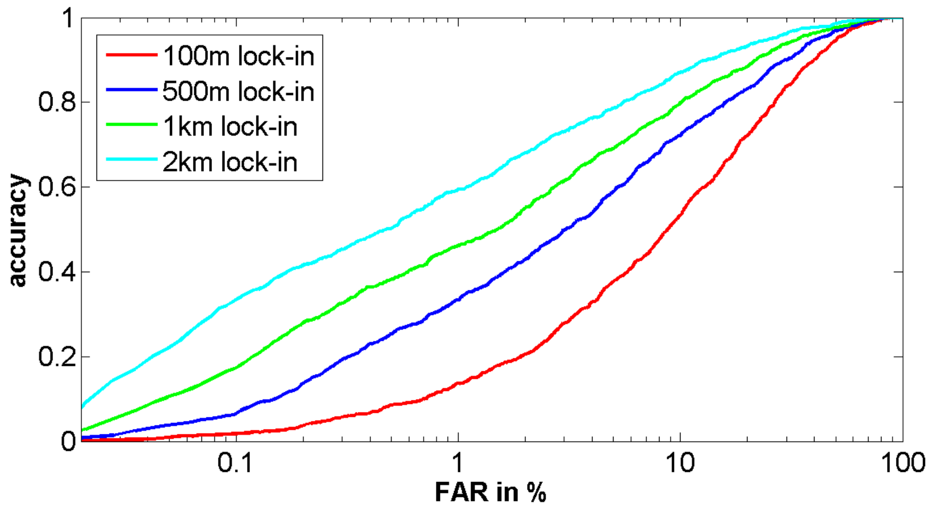


Figure 6.17: *ROC plot of the proposed prediction framework using the lock-in accuracies for a time horizon of one hour and different lock-in ranges. The ROC plot depicts the true positive rate for different false acceptance rates, shown on a logarithmic scale in order to enable a better evaluation in the more interesting regions of the plot.*

(i.e., 1% of the 14,272 nodes of the topological graph used in the experiments) most probable nodes of the prediction. By searching through the 1,427 most probable nodes (which relates to an FAR of 10%), the chances can be improved to an impressive 72%.

6.6.5 Lock-In ROC

In order to determine a suitable false acceptance rate for a desired lock-in accuracy, ROC like evaluations are shown in Fig. 6.17 and Fig. 6.18. For different lock-in ranges between $100m$ and $2km$, they show the lock-in accuracies for false acceptance rates between 0.2‰ and 100% and a time horizon of one hour in Fig. 6.17 and 20 minutes in Fig. 6.18. An FAR of 1‰ is equivalent to taking the 14 most probable nodes of the prediction result out of the total 14,272 nodes of the topological graph. For example, the 14 most probable nodes of a one hour prediction are within two kilometers of the ground truth with a probability of 33% according to the ROC in Fig. 6.17. Since one hour might be an uncommon duration for a taxi ride, Fig. 6.18 provides the same evaluation with a more suitable time horizon of 20 minutes, resulting in a probability of more than 58% for the same FAR and lock-in range. By searching through the 143 most probable nodes (i.e., a false acceptance rate of $\sim$ 1%), the probability rises to almost 60% for one hour and to 83% for a horizon of 20 minutes. If the result should be within $500m$ of the ground truth, the lock-in accuracy of the 14 most probable nodes reduces to 7.3% and 18.6% for a prediction length of one hour and 20 minutes

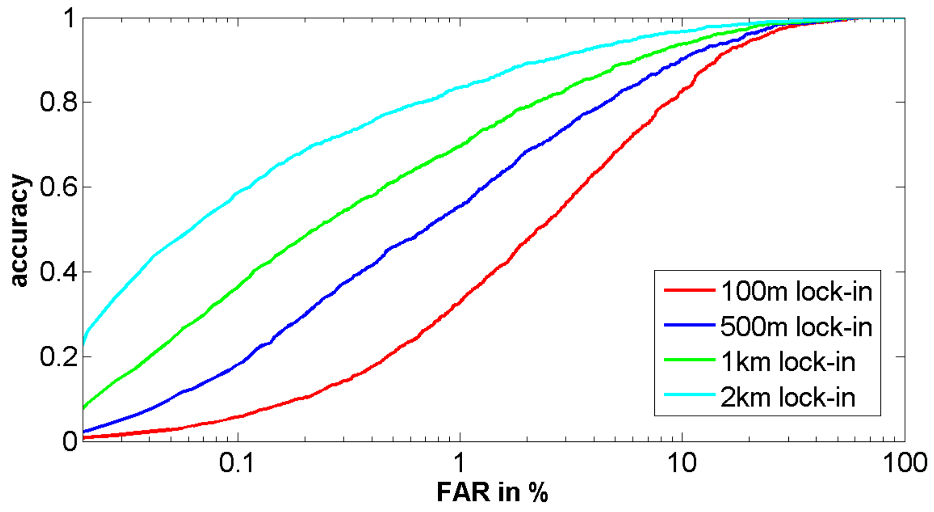


Figure 6.18: *ROC plot of the proposed prediction framework using the lock-in accuracies for a time horizon of* 20 *minutes and different lock-in ranges. Since the time horizon of one hour as in Fig. 6.17 is a rather uncommon duration for a taxi ride and thus might not apply to the usual motion model, the same evaluation was also conducted with a more suitable time horizon of* 20 *minutes.*

respectively, which is still a remarkably good result given the size of the state space and the length of the time horizon.

6.6.6 Intermediate Conclusion

The broad range of metrics used for evaluating the proposed prediction framework gives a concordant predication: it performs extraordinarily well on the Beijing test dataset for predictions of 20 minutes and even one hour into the future. Furthermore, the evaluation provides several anchors for researchers to compare their prediction algorithm to the proposed prediction framework.

In order to enhance the quantitative evaluation visually, an exemplary prediction of a taxi cab in Beijing is shown in Fig. 6.19. The car came from north heading south and turned east at the center crossing. The prediction started after the car drove approximately one kilometer, and the result is shown as an accumulated probability distribution from $t = 0s$ to $t = 600s$ colored from turquoise (low probability) via yellow (medium probability) to red (highest probability). As can be seen, the proposed prediction framework was able to correctly predict the path of the car and conjecture a straight movement across the intersection. Back roads and paths running in different directions are also anticipated with a lower probability.

In Fig. 6.20, another prediction result is shown. Additionally, the probability distributions for distinct time horizons (Fig. 6.20(c) to Fig. 6.20(f)) are provided, too. While

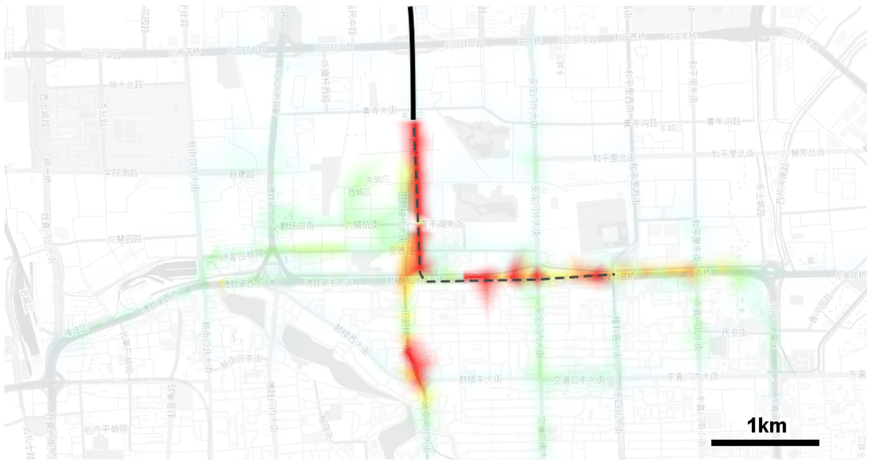


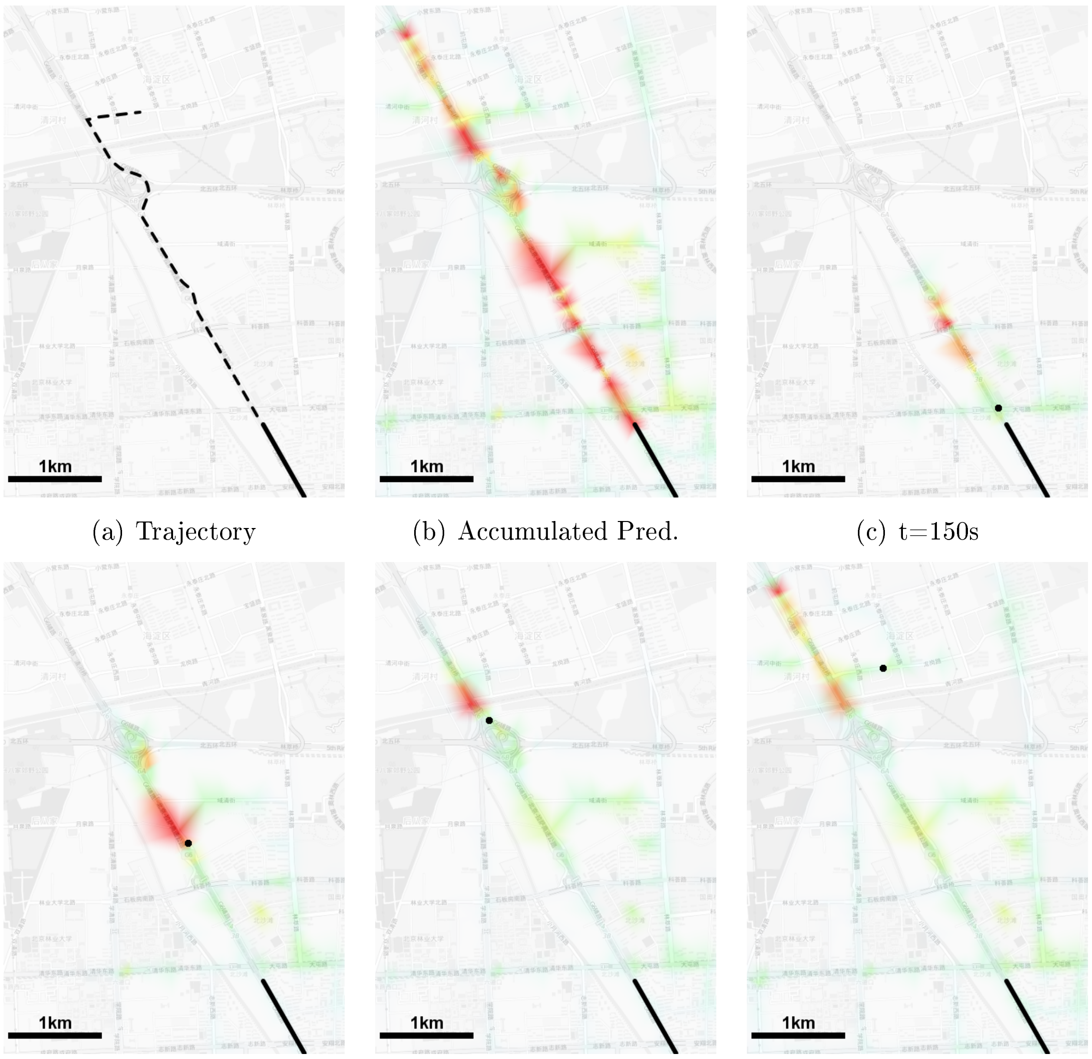
Figure 6.19: *Exemplary prediction of a single taxi drive. The observation is shown as a solid black line and the path for the next ten minutes is plotted as a dashed line. The probability distribution is shown in colors ranging from turquoise (low probability) to red (high probability). Since it is not possible to show the distribution for every time step in a single picture, the accumulated distribution is shown from $t = 0$ to $t = 600s$ to depict the most probable paths the car might take in that time period. As can be seen, the prediction correctly assumes a left turn (east) at the crossing in the middle while still maintaining a slightly lower probability for heading further south. The final position of the car at $t = 600s$ conforms with the predicted probability distribution, too. Paths along the side roads are also considered and taken into account with a low probability.*[1]

the proposed prediction framework performs remarkably well for a time horizon of up to 550 seconds, it also illustrates a situation with a less accurate prediction for the further time steps in Fig. 6.20(f). The prediction result assumes the car will stay on the road with a high probability; however, the car has taken a right turn. Despite the irregularity, a lesser but still significant probability is assigned to the nodes close to the real position of the car. Thus, the prediction does not contain the ground truth as the first rank anymore but it is still within the twenty most probable nodes, complying with a false acceptance rate of less than 2‰.

6.7 Conclusion

This chapter addressed several different evaluation-related topics. First, in Section 6.1, a small dataset was presented to contrast different methods of the prediction framework. The openly available Beijing taxi trajectory dataset was subsequently introduced for a comparison of the proposed prediction framework to the state of the art and the

[1]The maps in Fig. 6.19 and 6.20 were taken from openstreetmap.org "© OpenStreetMap contributors"



(a) Trajectory (b) Accumulated Pred. (c) t=150s

(d) t=300s (e) t=450s (f) t=600s

Figure 6.20: *Exemplary prediction of a single taxi drive. The observation is shown as a solid black line and the path for the next ten minutes is plotted as a dashed line in (a). The accumulated probability distribution for the time horizon of $t = 0s$ to $t = 600s$ is shown in colors ranging from turquoise (low probability) to red (high probability) in (b). The individual probability distributions for every 150 seconds are shown in (c-f) with the ground truth marked as a black dot. As can be seen, the car appears to be in a traffic jam in (c), causing the prediction to hurry ahead. In (d) and (e), the car is close to the major peak of the probability distribution, whereas minor peaks account for slower and faster traffic (orange, yellow and green specks). Different possible paths of the car (e.g., turning left, or right at the crossing close to the start) are also accounted for with low probabilities. In (f), the car takes a right turn but the proposed prediction framework assumes a high probability that the car will stay on the road. However, smaller probabilities (green and turquoise) are also correctly assigned to the vicinity of the car.*[1]

necessary steps for a post processing of the raw data was explained. Then, the most common error measures were presented briefly in Section 6.2, followed by the introduction of new error measures, PROC and PAUC, which were suitable to the multimodal results of the prediction.

The first experiment in Section 6.3 contrasted grid-based with the Mean-Shift based topologies. The former did not perform worse than the latter but the Mean-Shift based topology is preferred because it requires fewer nodes, edges, and less computational power while performing equally well.

The follow-up experiment evaluated the influence of the Mean-Shift kernel size on the prediction error. It showed that the kernel size is an important parameter, which needs to be well adjusted to the prediction problem at hand. As a rule of thumb, it should be set to the desired minimum distance of neighboring nodes. But, small deviations to the optimal size still yield good results which eases the task of setting the Mean-Shift kernel size.

Afterwards, the computational requirements of the proposed prediction framework were examined by comparing the iterative solution with the prediction in frequency domain in Section 6.4. Unfortunately, the computation in frequency domain can only speed up the prediction in the very unlikely case when a coarse result is sufficient for the problem at hand. The iterative solution should be favored since it not only provides the most accurate results, it is also fast enough for most of the prediction tasks on consumer hardware (e.g., it needs $3.8s$ for predicting a trajectory of the Humboldt dataset up to $102.4s$ into the future on an Intel(R) Core(TM) i7).

In order to evaluate the overall performance of the proposed prediction framework, it was compared to three state of the art prediction algorithms in Section 6.5 which are versatile in its application in Section 2.2. Unfortunately, it was not possible to contrast all methods on the Beijing test dataset due to either unmet assumptions or a tremendous computational complexity, and only an indirect comparison was possible. Thus, the results of the experiments should be assessed with prudence.

The proposed prediction method performed better than the EM algorithm of [Bennewitz, 2004] by comparing both on the true positive rate for obtaining the correct final position of the test trajectories for different length of observations. Since the estimation of the number of model components and the calculation of motion pattern do not scale well for the EM algorithm, the proposed prediction framework is also preferred while learning the prediction model on an enormous training dataset.

Despite the promising methods of the GHMM prediction algorithm, as presented in [Vasquez Govea, 2007], it still needs further research on improving the computational requirements on larger datasets. While providing better results than the EM algorithm of [Bennewitz, 2004], the GHMM algorithm is believed to perform equally to the proposed prediction framework at best while being more prone to the

imperfection of real-world datasets such as broken trajectories and much less capable to meet computational restrictions in real-time applications.
The sub-goal prediction algorithm of [Ikeda et al., 2013] performed similarly to the proposed prediction framework by comparing them to a constant velocity extrapolation, using the lock-in accuracy. Both prediction algorithms do utilize similar probabilistic methods but the former needs additional spatial information and its topological graph is tuned to a pedestrian motion model thus hindering other applications whereas the proposed prediction framework is in advantage by employing a more versatile topological model.
In order to enable a more convenient comparison to new prediction algorithms, a broader evaluation on several error measures is given in Section 6.6. The proposed prediction framework was evaluated on the Beijing test dataset using PROC, PAUC, Euclidean distance error, lock-in accuracy, and several variants of the lock-in accuracy which were modified to accommodate to the multimodal probability distributions the prediction gives as a result. All experiments showed that the proposed prediction framework performs remarkably well, even for a time horizon of 20 minutes and up to one hour. The visualization of selected test trajectories in Section 6.6.6 showed that the framework is also able to provide useful predictions, especially in cases which are considered wrong if only the most probable node would be taken into account.
In summary, the proposed prediction framework performs better than the addressed state of the art methods while providing a much more versatile and powerful approach which can be adapted to every prediction task on a continuous state space. It is advisable to utilize the Mean-Shift based topological graph (see Section 3.3.4) in conjunction with the iterative approach (as explained in Section 5.1.4) in order to obtain the best prediction results.

Chapter 7

Conclusion and Future Work

An innovative prediction framework consisting of three elements was presented in this thesis. The novel flow based prediction algorithm processes an observed trajectory into a comprehensive probability distribution for the whole state space and an extensive time horizon by utilizing a probabilistic transition model and an efficient topological graph. Not only does the extent of the result surpass state of the art prediction methods, which usually provide just a few distinct points or trajectories as a result, but also the quality of the result challenges current prediction algorithms.
The next section will summarize the results and contributions of this thesis, whereas Section 7.2 introduces new ideas which may help to further improve the prediction performance and the computational requirements of the proposed prediction framework.

7.1 Summary

This thesis dealt with several aspects of a long term motion prediction framework. Chapter 1 introduced the topic and named some exemplary applications of a motion prediction. Chapter 2 selected relevant state of the art methods and motivated the architecture of the proposed prediction framework. The main components of the framework were presented in Chapter 3 to 5. Finally, the resulting algorithm was evaluated in Chapter 6. The next sections will address these aspects respectively in more detail followed by a juxtaposition contrasting the advantages and disadvantages of the proposed prediction framework.

7.1.1 Applications

Some possible applications of the proposed prediction framework were introduced in Chapter 1. The practical aspects of implementing them are now discussed in the example of a mobile wireless network, autonomous cars, service robots, and a marketing predictor.

Mobile Networks

A main goal of trajectory prediction in mobile wireless networks is to anticipate the next base station a user is going to connect to [Wanalertlak et al., 2011]. Since the topology of the network is usually given by the cell towers, it can directly get transferred into the topological model without the need of a clustering algorithm. For every cell tower, a node is placed at the respective position, and the Delaunay triangulation can be utilized to connect the nodes with edges. The service provider of a mobile network perceives the movements of a user as the signal strength towards certain cell towers. Since a trajectory of an observation must be mapped onto the nodes of the topological graph (see Section 4.1), it is not necessary to triangulate the user in the Euclidean space. Instead, the sequence of the last cell towers the user was connected to (i.e., the tower with the highest signal strength) can be directly used as an observation. After gathering enough training data, the transitional probabilities and the transitional time distributions between the nodes are calculated. It is usually only important to predict the next cell tower of a user instead of a complete probability distribution. Thus, the flow based calculations are not necessary and a simple one step prediction (i.e., one step of the approximative solution in Section 5.1.2) needs to be performed using the transitional probabilities given the current observation.
In summary, the prediction of the next cell tower of a mobile user in a wireless network does not pose a problem to the proposed prediction network. Due to the specific restrictions of the application, several simplifications (e.g., the omission of the clustering step, or the focus on only the next node) of the involved methods are possible and help in improving and optimizing the workflow.

Autonomous Cars

It is a crucial ability for autonomous cars to predict trajectories of other vehicles and pedestrians in order to avoid collisions and obey the traffic rules [Hermes et al., 2009]. The proposed prediction method can be used to solve this task, but it poses some difficulties. First of all, the state space of an autonomous car is usually given in the form of an egocentric perspective. Using a global coordinate system for the topological model is not feasible since it can, in the worst case, comprise the whole world which in turn would cause enormous memory requirements and an unrealistic number of observations is needed to learn the topological and probabilistic model. By using a local coordinate system, observations of the surroundings are transformed according to the egomotion of the car which induces a high variety in the state space. If, for the sake of simplicity, the autonomous car can move with two speeds in one direction, an object showing up at a certain distance to the front left and moving to the right can follow two trajectories (one for each speed of the car) in the local coordinate system. Since the position, speed and movement direction of the autonomous car can change continuously, an observation can have a multitude of trajectories in the topological graph. Creating a topological model on such a local coordinate system is possible, but the high diversity of observations will most likely cause very imprecise predictions because a lot of unrelated situations are superimposed onto one point in the state space and the same situation may be observed in different regions of the local coordinate system.

One method of increasing the prediction accuracy is to not only use the Euclidean coordinates as the state space but also to include the speed and steering angle of the car as additional dimensions. By using the previous example, an object showing up at the same relative point and following the same global trajectory will always cause the same trajectory for the same speed and steering angle in the topological graph. Thus, it can be predicted with a much higher accuracy. The only disadvantage of such an approach is the need of many more observations to train the topological graph and the probabilistic model due to the higher dimension of the state space (also known as "the curse of dimensionality" [Bellman and Corporation, 1957]). Using knowledge about the physical correlations between the Euclidean trajectories of objects and the current speed and steering angle of a car, it is possible to include an observation into the topological and probabilistic models not only for the car's current speed and angle, but for every possible combination of both. Such a procedure would greatly improve prediction accuracy and reduce the number of observations needed to train the models. Furthermore, it would also give a good example of how the proposed prediction method can be improved by model specific knowledge.

It is expected that the proposed prediction framework can be successfully applied to an autonomous car for predicting vehicles and pedestrians in traffic. But, since the trajectories are highly confined by the environment and follow very distinct models (due to both physical and regulatory laws), a prediction algorithm tailored to the specific task may provide equal or even better results without the need to learn a topological and probabilistic model in advance.

Service Robots

The task of predicting movements on a mobile service robot may seem very similar to the task of predicting trajectories on an autonomous car; however, it yields some major differences.

First of all, the state space of a robot in a home environment, for example, is less confined. The robot is not restricted to a road network, is allowed to move around freely without traffic restrictions, and usually has a higher degree of freedom in its motions [Siegwart et al., 2011]. Therefore, the prediction framework needs to cover several situations and, thus, more observations are required to create an adequate topological and probabilistic model. If the environment the robot operates in is restricted in size, the use of a global Euclidean coordinate system is advised. The robot will miss many observations since it may only be able to observe its immediate surroundings. But, the application of a two dimensional global coordinate system will most likely need less observations to create an appropriate topological and probabilistic model compared to a much more complicated, high dimensional, and local system, similar to the recommendations for the autonomous car.

Second, the environment a mobile service robot operates in is usually not very extensive and is almost always located inside of a building. Thus, the topological graph can cover the entire state space with few memory requirements, which in turn encourages the use of a global Euclidean coordinate system.

Third, the robot itself is a point of interest in most environments, causing pedestrians to observe, approach, avoid, or interact with it [Sabanovic et al., 2006, Müller et al., 2008]. The robot introduces a high degree of variety since it influences the trajectories of the observations depending on its own position. Contradictory to the previous recommendation to use a global coordinate system, this behavior can best be countered by using a local coordinate system since the robot (being the attractor or deflector) will always be located in the origin of the topological model.

Using the proposed prediction framework with a global Euclidean coordinate system is expected to perform well on a mobile service robot. But, due to the robot's interactions with its surroundings and the limited perception range, prediction algorithms using social force models [Luber et al., 2010] or similar approaches are believed to provide better prediction results, if a long training phase to collect observations needs to be avoided.

Marketing Predictor

Contrary to the previous two applications, the main task of a prediction algorithm in marketing is to enable a gentle "collision" with a customer. Since the use of machine learning, cognitive robotics, and AI is relatively new in the fields of experimental marketing, ample research still needs to be done to identify the most beneficial use cases for prediction algorithms.

In the naïve example of a spatial prediction of customers in a mall in order to intercept them and offer services, the implementation of the proposed prediction framework is straightforward: Using a global coordinate system, the topological graph is advised

to be created by a clustering algorithm, preferably the Mean-Shift algorithm, as explained in Section 3.3. If the environment is highly structured (e.g., by shelves), a manual creation of the topological graph may also be feasible. After learning the probabilistic model (as addressed in Chapter 4), utilizing Markov-trees to store the transitional probabilities and Kernel Density Estimations to represent the transitional time distributions, the prediction framework using the iterative solution as explained in Section 5.1.4 can be applied to predict a customer's trajectory.
By applying the prediction algorithm on more abstract metrics (like purchase intentions [Dees et al., 2008], impulsiveness, and health consciousness [Bearden and Netemeyer, 1999]) describing a customer on a non-spatial scale, the future shopping behavior and a suitable time to offer a product may be predicted. With the current level of research in these fields, it is not possible to provide tangible advice on how to apply the prediction framework because the concomitants are not yet known. However, it can be assumed that the generic approach for a spatial state space should provide a decent starting point.

Conclusion

The proposed prediction framework offers the potential of a beneficial generic solution to a variety of prediction problems. As long as the state space and the involved motion models do not change significantly, the framework provides useful and comprehensive prediction results. But, if the state space does not remain static (e.g., due to the use of a local coordinate system) or if the motion models do change (like in the example of the mobile service robot), the prediction framework may not be able to give a very precise prediction result. By incorporating the respective variations into the state space, more accurate results can be achieved but in exchange for an increased need of observations.

7.1.2 Motivation

After introducing the concept and necessity of a long term motion prediction for several application scenarios in Chapter 1, the current state of the art was presented and analyzed in Chapter 2. One peculiarity was that no method in the related literature was able to provide a probability distribution over the whole spatio-temporal state space. Predicting the motion of an object with state of the art algorithms usually results in a number of most likely future trajectories or a few distinct points. Such simplistic results prevent the application of the prediction algorithm for tasks which need to determine the probability to observe an object in a certain region, or even at a single specific point in the spatio-temporal space. Furthermore, almost all state of the art algorithms were tailored to a specific problem statement and are therefore

only applicable to a very limited set of tasks. However, three algorithms [Bennewitz, 2004, Vasquez Govea, 2007, Ikeda et al., 2013] were identified as being versatile and thus selected as a reference to this thesis.
The proposed prediction algorithm was also intended to be used on a specific subject, but the drawbacks of the state of the art methods inspired the creation of a most versatile prediction framework, assessing the entire state space by providing a comprehensive probability distribution as a result.

7.1.3 Framework

The evaluation of the state of the art in long term motion prediction unveiled a common architecture: First, a topological representation was created in a learning phase in order to enable an easy processing of trajectories or observations. Afterwards, a probabilistic model was learned which is encoding similarities of observed movements or state transitions. Finally, both parts were processed by an algorithm in order to predict an observation. This architecture was also adopted for the proposed prediction framework and Chapter 3 to 5 focused on the respective components.

Topological Representation

Chapter 3 dealt with the topological representation of the state space and introduced the topological graph as a well suited concept. Whereas grid-based graphs, as presented in Section 3.2, are simple and straightforward, a data driven topology provides both a better sampling of the state space and has less computational requirements. The data driven graph can be created by first determining the nodes based on clustering observations and then connecting them by edges. After assessing different clustering methods for the creation of the nodes in Section 3.3, the Mean-Shift algorithm is recommended because it requires the user to set only one easily ascertainable parameter, and it is able to sample the state space according to the density of the training data. One disadvantage of the Mean-Shift algorithm is its lack of an on-line updating mechanism. If the graph has to be learned iteratively, the Growing Neural Gas is a better suited clustering algorithm because it can be easily updated and inherently connects the nodes with edges. Unfortunately, the Growing Neural Gas needs six parameters to be tuned to the characteristics of the state space, which makes it more difficult to obtain a good clustering result.
After determining the nodes, preferably with the Mean-Shift algorithm, they can be connected with edges using the Delaunay triangulation, as described in Section 3.4. It ensures that any continuous trajectory in the state space can be represented as a sequence of connected nodes without any interruptions due to a missing edge be-

tween two successive nodes. This property is of utmost importance for the proposed prediction algorithm to function properly.
If memory requirements are no issue, the use of a regular grid as the topology is also a valid option. As shown in Section 6.3, it does not perform much differently than a graph learned by a clustering algorithm. Using a grid avoids the time consuming part of learning a topology based on observations in exchange for a higher memory requirement and a slightly increased computational complexity in the application phase.

Probabilistic Model

Once the topological graph is created, it has to be enriched with a probabilistic model, as presented in Chapter 4. The probabilistic model is comprised of two elements: the transitional probabilities for each node and the transitional time distributions for every edge. As explained in Section 4.1, the transitional probabilities of a node provide the likelihood of an observation to transit each connected neighbor. In a learning phase, they are estimated by the relative frequencies of the training trajectories performing the respective transition.
For every edge, a transitional time distribution specifies how much time is needed for a transition between the connected nodes, as described in Section 4.3. It is learned by taking the related observations of a training dataset and sampling the time they need for that transition. Those samples are then used to estimate a time distribution, either in a parametric form, like GMM, or preferably as a non-parametric KDE.
In order to account for the previous path of an observation, the transitional probabilities and time distributions are not restricted to a first-order Markov model (i.e., the probabilities and distributions are only dependent on the current state). A novel concept, introduced as the Markov-tree in Section 4.2, enables an easy method to efficiently learn, manage, and process transitional probabilities and time distributions with Markov chains of arbitrary lengths. The only restriction on the length of the previous path taken into account during the computations is imposed by the amount of available training data. For this purpose, an assessment for the quality of a probability estimation using Chebyshev's inequality is presented and recommended to be used for limiting the descend into the Markov-tree to an adequate depth.

Prediction Framework

As presented in Chapter 5, the main prediction method utilized the topological graph and the probabilistic model in order to calculate a comprehensive probability distribution across the whole state space and for every time step. The main idea behind the prediction method was adopted from common path planning algorithms: an initial belief, i.e., the current observation, is spread across the entire graph like a wavefront with respect to its topology.

The concept of a probabilistic flow was introduced in Section 5.1 in order to implement such a procedure. Starting from the node of the last known observation with the observational probability of one, it diverges at nodes and is delayed at edges while it spreads throughout the entire graph according to the transitional probabilities and transitional time distributions respectively. In Section 5.1.1, the formal definition of the prediction algorithm was given using a continuous timescale. Then, an approximative implementation was presented in Section 5.1.2 in order to transfer it to a computationally less challenging discrete time scale and to provide a more practical algorithm. However, the approximative implementation involves multiple convolutions, making it very time consuming. By transferring the computation into the frequency domain, a faster algorithm can be implemented, as explained in Section 5.1.3. The computational complexity can be reduced even further in most of the use cases if the approach is changed from approximating and refining the probabilistic flows iteratively to calculating them for every time step successively, as shown in Section 5.1.4.

If the underlying motion model is not static and tends to change over time, an on-line updating mechanism is required to keep the topological and the probabilistic models up to date. Section 5.2 discussed the model update and concluded that the topological model should stay static as long as possible since nearly any change will also severely affect every transitional probability and time distribution. Randomly seeded nodes can be added prior to creating the edges and learning the probabilistic models. States and regions of the state space that have yet to be observed and accounted for will be taken into consideration and extensive recalculations will be avoided. Updating the probabilistic models is less critical due to the benefits of the Markov-tree. It enables an easy and effective refining of the transitional probabilities and time distributions during the application phase of the proposed prediction framework.

7.1.4 Evaluation

The proposed prediction framework was evaluated in Chapter 6. Section 6.1 introduced the used datasets and the most common error measures were explained in Section 6.2. Since none of these errors were able to fully address the rich information the proposed prediction framework provides in form of a comprehensive probability distribution, new error measures, the PROC and PAUC, were introduced in Section 6.2.5.

The evaluation of different topological graphs in Section 6.3 unveiled that the Mean-Shift approach provided results comparable to more simple grid-based topologies while enabling a much faster and computationally less expensive prediction. Furthermore, it was shown that the kernel size had a significant influence on the prediction accuracy and must be chosen carefully. While being tolerant of small deviations, it is recommended to tune the kernel size to the minimum distance two neighboring nodes should have.

Section 6.4 compared the prediction algorithm using the frequency domain, as introduced in Section 5.1.3, to the iterative approach, as presented in Section 5.1.4. Both approaches provided similar results but the iterative algorithm was almost one order of magnitude faster. The algorithm using the frequency domain approximates the whole probability distribution for the full time horizon step by step, whereas the iterative algorithm calculates the prediction result successively one time step at a time. Hence, the frequency domain is only advantageous if a rough estimate of the prediction is needed for a point in time in the distant future. In nearly every other use case, the iterative approach is to be preferred.

The comparison of the proposed prediction framework to state of the art algorithms [Bennewitz, 2004, Vasquez Govea, 2007, Ikeda et al., 2013] in Section 6.5 proved to be difficult since a direct comparison was not possible. They were evaluated on undisclosed datasets and a reevaluation on adequately extensive datasets would not have been feasible. However, indirect comparisons have shown that the proposed prediction framework was at least on par with or was even superior to the state of the art algorithms while providing a more comprehensive prediction result and a more versatile approach.

In order to facilitate the comparison of a new method to this prediction framework, a broad evaluation on several error measures was given in Section 6.6. For this task, an openly available dataset, containing recordings of taxi movements in Beijing [Zhu et al., 2013, Zhang et al., 2011, Zhang, 2009], was chosen for the evaluation. This dataset can also be substituted by a variety of other datasets [Zheng et al., 2008, Piorkowski et al., 2009, Zheng et al., 2009, Zheng et al., 2010, Yuan et al., 2010, Yuan et al., 2011, Zheng, 2011, Zheng, 2012, OpenStreetMap and contributors, 2013] without difficulty because it represents a common type of motion on an enormous, spatially almost unrestricted state space. As shown in Section 6.6, the proposed prediction framework was able to predict a taxi movement with a considerable accuracy twenty minutes into the future on the scale of a huge city. Even a prediction of one hour into the future still yields decent results if a concession to spatial accuracy is accepted.

7.1.5 Considerations

In order to decide if the proposed prediction framework is a worthwhile solution to a specific prediction problem, it is not only important to be aware of the implementational details as summarized in the previous sections, but it is also necessary to know of its advantages and disadvantages.

Advantages

- Versatile:
  The most important quality of the proposed prediction framework is its highly versatile architecture. Because both the topological and the probabilistic model are created by using observations, no model specific knowledge needs to be implemented. This property enables a broad range of applications since the motion model of most prediction problems is often either unknown or it is just a coarse approximation. But, by creating a probabilistic model based on observations, all factors influencing a specific motion are intrinsically accounted for.

- Adaptable:
  Despite the fact that model specific knowledge does not have to be implemented, additional information about the motion mechanics or the topology can be used to enhance the prediction framework and to reduce the need of observations. One option would be to extend a single observation across the state space using knowledge about the relation of the different dimensions (similar to the transformation of a single trajectory onto different velocities and steering angles in the example of the autonomous car in Section 7.1.1). Another way of including problem specific knowledge would be to directly limit, modify, or set the transitional time distributions, transitional probabilities, or the topological graph according to the restrictions and dynamics of the given system.

- Few parameters:
  The adaptability of the versatile approach is not the only reason why the proposed prediction framework enables an easy implementation for a given problem. The fact that only one sensitive parameter, the kernel width as mentioned in Section 3.3.4, needs to be tuned to the system at hand (besides the more generic and less critical implementational parameters such as the time step width, or the final horizon for the prediction) enables sufficient prediction results early on without the need to perform several reruns to find an optimal set of values. But, even that parameter can be avoided if the clustering algorithm is omitted due to a known topology (similar to the example of a mobile network in Section 7.1.1) or if a regular grid is chosen.

- On-line updating:
  As mentioned in Section 5.2, the prediction framework is, within some restrictions, able to incorporate new observations into the topological and probabilistic models even after the initialization phase. This ability can be used to either start using the prediction algorithm early on and improve it during its application, or to account for changes in the motion mechanics of the system (e.g., an

obstruction in the topology like a road closure).

Disadvantages

- Global topology:
  The main disadvantage of the proposed prediction framework is the need of a global (i.e., mostly static and not frequently changing) topology. It is also possible to use a local coordinate system but it should be avoided since it introduces another set of problems, as discussed in Section 7.1.1. A global topology complicates the application of the proposed prediction framework on moving systems, such as robots or cars, since a transformation of the systems observations into a global coordinate system according to its egomotion becomes necessary.

- Requires extensive data:
  Processing a global topology on a computer restricts the application of the prediction algorithm to problems in a bounded state space. Within this state space, all the major situations must be covered by observations in order to obtain good prediction results. For example, a car's movements can be predicted on a scale of a city with manageable effort (as demonstrated in Section 6.6), but to predict its trajectory on the scale of the entire world requires an enormous amount of observational data and processing power, which makes it practically unfeasible. Nevertheless, a method to reduce the needed observational data and processing power for such an application will be presented later in Section 7.2.2.

- Slow clustering:
  If a cluster-based topology is chosen instead of a regular grid, the creation of the topological model may take a lot of time if a huge state space needs to be covered. For a small area, the Mean-Shift algorithm is able to process the observations relatively quickly (e.g., on the Humboldt dataset, the clustering takes only some seconds on consumer hardware). However, a larger state space needs more observations in order to be adequately represented, which in turn results in a significantly longer runtime of the Mean-Shift clustering algorithm (e.g., it takes weeks on consumer hardware to create the topological graph for the Beijing dataset). The computational needs can be reduced by using a clustering algorithm with a low complexity (e.g., the K-Means algorithm has a complexity of $O(n)$ compared to the Mean-Shift algorithm with $O(n^2)$) but it introduces another set of disadvantages, as discussed in Section 3.3.

7.2 Modifications

Despite the remarkably accurate prediction results and the real-time capabilities of the proposed prediction framework, there is still some room for improvement and modifications. Its performance can be improved quantitatively by optimizing the computationally intensive modules and by lowering their requirements. Furthermore, the prediction capabilities can be enhanced qualitatively by applying a divide and conquer approach on the propagation of the probabilistic flows. Those two aspects of future research will be presented in more detail in the next two sections.

7.2.1 Performance improvements

Reducing the computational requirements helps in several aspects. This speeds up the prediction process, and it also enables the use of a finer temporal resolution, a larger time horizon, or the processing of a greater state space. The results in Section 6.4 show that the framework is able to predict small to mid-sized state spaces at least one order of magnitude faster than real-time. The average prediction on the Humboldt dataset with a graph consisting of $1,220$ nodes and $2,041$ edges took 3.8 s. A time horizon of 102.4 s and a temporal resolution of 0.1 s was chosen and the prediction was performed on an Intel(R) Core(TM) i7 quad core processor running at 2.67 GHz while using less than 50 MByte of RAM (including the overhead of a GUI and visualization). This performance does not suggest the need of an improvement but unfortunately, the Beijing scenario showed that the computational requirements did not scale linear with the number of edges or the length of the time horizon. On the graph with $14,272$ nodes and $57,067$ edges, a prediction 1,200 s into the future with a temporal resolution of 1 s took 750 s and allocated about 1.1 GByte of RAM. The prediction is also faster than real-time but the necessity of an optimized implementation becomes more obvious. Some experiments in Section 6.6 even used a time horizon of 3,600 s and a prediction took 3,160 s on average while allocating almost 10 GByte of RAM. Such an application is nearly unfeasible even if it is slightly faster than real-time. In order to process such a huge state space and time horizon, it would be advantageous to lower both the computational requirements and the memory usage.
The most obvious option is to parallelize the flow update since the flows are independent from each other for every individual time step. If it is implemented on a GPU, each flow in Line 10 of the pseudocode in Fig. 5.5 can be computed by one processing unit, which would result in a tremendous speed-up (depending on the number of processing units of the GPU) compared to the calculation on a single CPU core. Unfortunately, the transfer of the algorithm onto a GPU is difficult since a lot of attention needs to be paid to memory management. The amount of RAM on a graphics card is

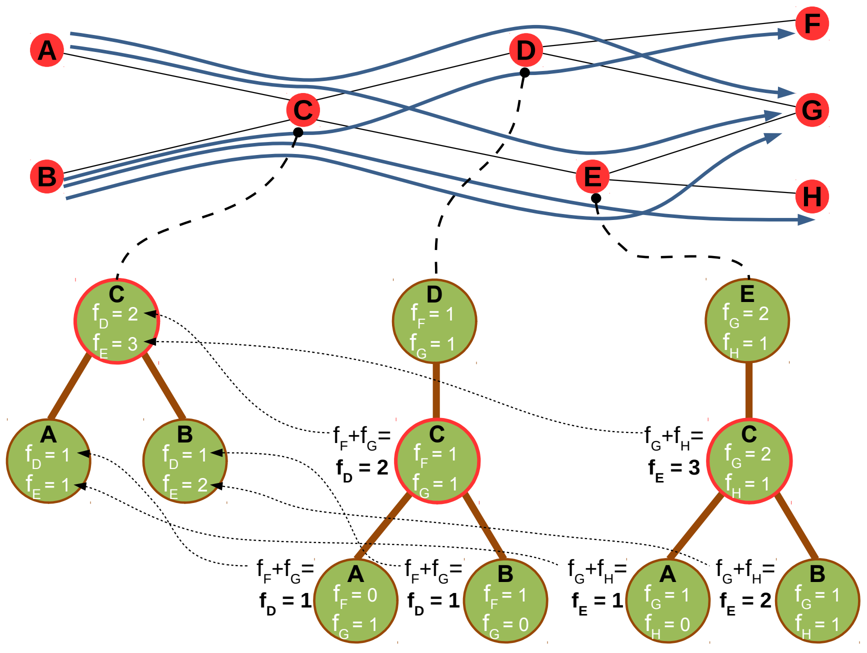


Figure 7.1: *Redundancy in Markov-trees of neighboring nodes. The nodes of the topological graph in the upper part of the figure are shown as red circles with black edges between them, and observed trajectories are depicted as blue lines with an arrow representing their direction. The Markov-trees of nodes C, D, and E are shown in the lower half after enriching them with the five observations. The vertices of the Markov-trees are depicted as green circles and are connected by brown lines. They are labeled with their index as a black capital letter and their individual counter f_x for the number of observed transitions to the neighboring nodes in white. The vertex for node C is highlighted red to show the similarities in the trees. The Markov-trees of node D and E are similar to the tree of node C, extended by their own vertex on top. As indicated by the dotted black arrows in the lower half of the figure, the counter of the Markov-tree of node C can be deduced from the counter of the neighboring Markov-trees.*

usually smaller than the RAM of the host computer and it can be understood that the complete information of the flows, time distributions, and transitional probabilities does not fit into it. Thus, it is necessary to determine and reorganize the information needed for the calculations of the current time step on the CPU before finally computing the iteration on the GPU. This necessity and the involved copy operations between the host and device RAM adds overhead to the algorithm and might impede the benefits of the parallelization of Line 10 of the pseudocode in Fig. 5.5. However, utilizing multiple CPU cores for the task is much easier because all processing units can operate on the host RAM and do not require any additional memory management or copying.

One method of lowering the memory requirements is to reduce the redundancy in the current implementation of the probabilistic model, especially the Markov-trees. They share a lot of information, as depicted in Fig. 7.1. Neighboring Markov-trees do not only have a similar structure but also the information stored in one tree is deducible from its adjacent Markov-trees. In the example of node D in Fig. 7.1, the Markov-tree shares the same structure of the tree of node C, extended by a vertex for node D on top of it. Furthermore, the counter f_D for the vertices of the tree of node C can be calculated by adding together all counter f_F and f_G of their respective vertices in the Markov-tree of node D. This is due to the fact that all observations contributing to the counter f_D are going from node C to node D and then leaving node D towards its neighbors. Similar to Kirchhoff's current law in electrical engineering, the number of observations entering node D is equal to the number of observations leaving it. Thus, the sum of all counter of a vertex in the Markov-tree of node D (outgoing from node D) equals the counter f_D of the respective vertex in the Markov-tree of node C (incoming to node D). However, this only holds true for observations passing the nodes. If an observation vanishes at a node or a new one starts, this rule does not apply because the sums of ingoing and outgoing observations are no longer equal. Nevertheless, the vast amount of redundancy is a good indicator that the solution of using one individual Markov-tree for each node is not ideal. It may be possible to save a lot of memory and reduce the redundancy by using one big structure, similar to a global Markov-tree, to store the transitional counter and time distributions.

7.2.2 Hierarchical Graph

While parallelization or the application of a global Markov-tree would reduce the computational requirements, the concept of hierarchical graphs might be useful for improving the qualitative prediction capabilities. The utilization of just one big graph to predict taxi movements in a huge city accentuated a practical inadequacy in the experiments of Section 6.6: the probabilistic model did not contain a comprehensive description of the motion mechanics despite the large training dataset. Due to the combinatorial complexity, the depth of the Markov-trees had to be restricted to five levels and the use of Chebyshev's inequality often inhibited the descent into the trees even further. Thus, the cars previous path was only considered for the last five nodes at most. With a common distance of 100 m between two neighboring nodes, only the last ~400 m of the observed trajectory have an influence on the prediction result.

This contradicts the common intention of using a taxi cab: it is not used to drive straight over a crossing, to take a turn, or to go around the corner. A taxi cab is mostly taken to go from a starting point to a goal, usually several kilometers away. Yet in order to incorporate the information of the starting point and thus the main motive

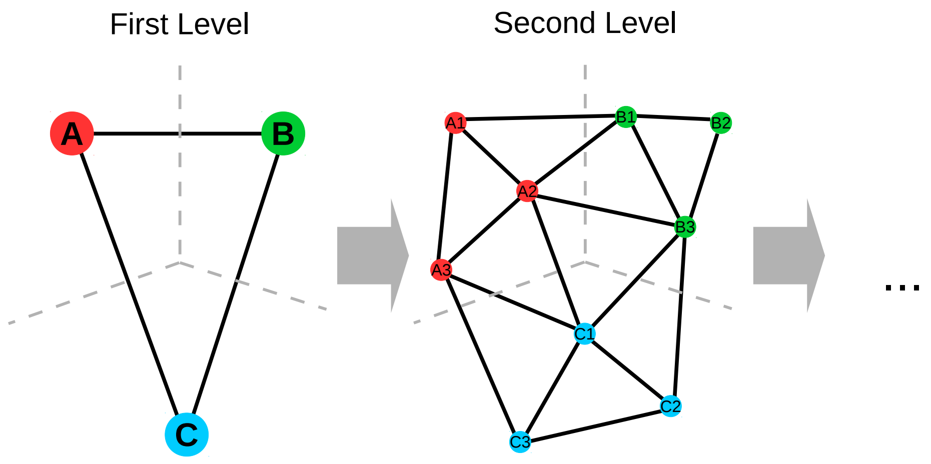


Figure 7.2: *The first two layers of an exemplary hierarchical graph. The nodes of the first layer are depicted as red, blue, and green dots on the left, connected by black edges. The corresponding child nodes in the second level are shown in their respective color. The topological border between the three nodes of the first layer are shown as gray dotted lines. The number of layers is not restricted but just the first two are shown in order to keep the figure clear and simple. The prediction starts in the first layer by calculating all flows. The resulting flows are then transferred onto the next layer and refined. This procedure is repeated until the lowest layer has been processed. Transferring a flow one layer down is considered to be the most challenging aspect of the hierarchical graph. Due to the topological change of a transfer, a flow in the first level coming from A to B and continuing to C has multiple representations in the second level: it can either come from A1 or A2, continue with an arbitrary sequence of B1, B2, and B3, and finally head towards C1, or C2. How the upper level flow is distributed to the multitude of lower level possibilities needs to be dealt with in future research.*

of a taxi drive, the complete observed trajectory instead of just the last ~400 m must be included for calculating a better prediction result. A topological graph covering the whole state space with more widespread and less nodes will take a longer past of the trajectory into account but it also reduces the spatial resolution drastically. Additionally, it is still not guaranteed that the full observation can be represented by a sequence of five or less nodes without condoning a further reduction in spatial resolution.

This dilemma might be solvable by using a hierarchical graph structure, as depicted in Fig. 7.2. Its uppermost level contains a very coarse graph with only a few nodes covering the whole state space. The next subordinate level holds a graph which samples the state space with more nodes (e.g., ten times the number). With each subsequent level, the state space is represented by more nodes until the desired spatial resolution is attained by the lowest graph.

In order to calculate a prediction, the proposed prediction framework is first applied on the graph of the uppermost level to obtain a probability distribution on a very low resolution. In the example of taxi cab movements across a city, this would provide information about the general destination of the car and how it will get there globally (e.g., what suburbs it will most likely travel through). Each node in the current graph corresponds to a set of child nodes in the next level as determined by their Voronoi regions. The edges in the lower graph which are shared by nodes with different parent nodes are defined as border edges since they represent the lower level link of the respective edge in the upper level. The prediction result of the upper level can be refined by transferring all flows of the upper level to the child graph according to the respective border edges they travel along. Those transferred flows can then be used to predict the probability distribution on the lower graph. This procedure is repeated until the lowest level is processed and the high resolution result is calculated.

Two problems arise with such a procedure. First, the observed trajectory cannot just be mapped onto the uppermost graph in order to predict a movement because it might only cover one node and valuable information (e.g., the direction of movement) gets lost. It should be mapped onto the lowest graph of the hierarchy and propagated upwards while keeping the information about the initial inflow it creates on every level, which might add its own set of problems.

Second, the transfer of a probabilistic flow one level down is significant. A flow on the upper level, coming from node A, currently at node B and going to its neighboring node C essentially describes the transition from the border to the Voronoi region of node A, across the domain of the current node B, to the boundary of the region of node C. Thus, the flow should be presented as an inflow on the lower level at the B-C border edges towards the sub-graph of C. Unfortunately, the topology of the lower level is different from the upper level and the condition of the flow (which is $[C|B, A, \ldots]$ on the upper level) has multiple representations on the lower level since various sequences of nodes to get from the A-B border to the B-C border may exist in the sub graph of B (in the example of Fig. 7.2, $[C2|B3, B2, B1, A1]$, $[C1|B3, B1, A1]$, or $[C1|B3, B1, A2]$ are just a few of many options). Furthermore, it is very likely that several B-C border edges are in existence (two in the example of Fig. 7.2) which makes it necessary to distribute the flow among them.

Solving these two issues is considered to be the greatest obstacle in developing an elaborate concept of the hierarchical graph. However, it can be assumed that the benefit of such an enhancement is not only a better prediction result. It is also likely to provide an improvement of the computational requirements because the flows do not propagate through the graph in its full extend. Due to the divide and conquer approach, they are processed in small patches of the graph, essentially reducing the total number of flows and limiting the combinatorial complexity of their condition.

Bibliography

[Asahara et al., 2011] Asahara, Akinori, K. Maruyama, A. Sato and K. Seto (2011). *Pedestrian-movement prediction based on mixed Markov-chain model.* In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 25–33. ACM.

[Ashbrook and Starner, 2003] Ashbrook, Daniel and T. Starner (2003). *Using GPS to learn significant locations and predict movement across multiple users.* Personal and Ubiquitous Computing, 7(5):275–286.

[Aurenhammer, 1991] Aurenhammer, Franz (1991). *Voronoi diagrams - a survey of a fundamental geometric data structure.* ACM Computing Surveys (CSUR), 23(3):345–405.

[Bachmann et al., 2013] Bachmann, Anja, C. Borgelt and G. Gidófalvi (2013). *Incremental Frequent Route Based Trajectory Prediction.* In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, p. 49. ACM.

[Baum et al., 1970] Baum, Leonard E, T. Petrie, G. Soules and N. Weiss (1970). *A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains.* The annals of mathematical statistics, pp. 164–171.

[Bearden and Netemeyer, 1999] Bearden, William O and R. G. Netemeyer (1999). *Handbook of marketing scales: Multi-item measures for marketing and consumer behavior research.* Sage.

[Bellman and Corporation, 1957] Bellman, R. and R. Corporation (1957). *Dynamic Programming.* Rand Corporation research study. Princeton University Press.

[Bennewitz, 2004] Bennewitz, Maren (2004). *Mobile robot navigation in dynamic environments.* PhD thesis

[Bennewitz et al., 2005] Bennewitz, Maren, W. Burgard, G. Cielniak and S. Thrun (2005). *Learning motion patterns of people for compliant robot motion*. Internationl Journal of Robotics Research, 24:31–48.

[Browning et al., 2006] Browning, Raymond C, E. A. Baker, J. A. Herron and R. Kram (2006). *Effects of obesity and sex on the energetic cost and preferred speed of walking*. Journal of Applied Physiology, 100(2):390–398.

[Burgard et al., 1999] Burgard, Wolfram, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner and S. Thrun (1999). *Experiences with an interactive museum tour-guide robot*. Artificial intelligence, 114(1):3–55.

[Chebyshev, 1867] Chebyshev, PL (1867). *On mean values*. Journal de mathématiques pures et appliquées, 2(12):177–184.

[Cignoni et al., 1998] Cignoni, Paolo, C. Montani and R. Scopigno (1998). *DeWall: A fast divide and conquer Delaunay triangulation algorithm in E_d*. Computer-Aided Design, 30(5):333–341.

[Comaniciu et al., 2001] Comaniciu, Dorin, V. Ramesh and P. Meer (2001). *The variable bandwidth mean shift and data-driven scale selection*. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1, pp. 438–445. IEEE.

[Dees et al., 2008] Dees, Windy, G. Bennett and J. Villegas (2008). *Measuring the effectiveness of sponsorship of an elite intercollegiate football program*. Sport Marketing Quarterly, 17(2):79.

[Delaunay, 1934] Delaunay, Boris (1934). *Sur la sphere vide*. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 7(793-800):1–2.

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

[Drengner et al., 2011] Drengner, Jan, S. Jahn and C. Zanger (2011). *Measuring event–brand congruence*. Event Management, 15(1):25–36.

[Dwyer, 1987] Dwyer, Rex A (1987). *A faster divide-and-conquer algorithm for constructing Delaunay triangulations*. Algorithmica, 2(1-4):137–151.

[Eisenbach et al., 2012] Eisenbach, Markus, A. Kolarow, K. Schenk, K. Debes and H. Gross (2012). *View invariant appearance-based person reidentification using fast online feature selection and score level fusion*. In *Advanced Video* *and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, pp. 184–190. IEEE.

[Eisenbach et al., 2013] Eisenbach, Markus, P. Scheiner, A. Kolarow, K. Schenk, H.-M. Gross and I. Weinreich (2013). *Learning Illumination Maps for Color Constancy in Person Reidentification.* Workshop Farbbildverarbeitung, 19.

[Ester et al., 1996] Ester, Martin, H.-P. Kriegel, J. Sander and X. Xu (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise.*. In *Kdd*, vol. 96, pp. 226–231.

[Fritzke et al., 1995] Fritzke, Bernd et al. (1995). *A growing neural gas network learns topologies.* Advances in neural information processing systems, 7:625–632.

[Fukunaga and Hostetler, 1975] Fukunaga, K. and L. Hostetler (1975). *The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition.* IEEE Transactions on Information Theory, 21(1):32–40.

[Fulgenzi et al., 2009] Fulgenzi, Chiara, A. Spalanzani and C. Laugier (2009). *Probabilistic motion planning among moving obstacles following typical motion patterns.* In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 4027–4033. IEEE.

[Giannotti et al., 2007] Giannotti, Fosca, M. Nanni, F. Pinelli and D. Pedreschi (2007). *Trajectory pattern mining.* In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 330–339. ACM.

[Gidófalvi et al., 2011] Gidófalvi, Győző, C. Borgelt, M. Kaul and T. B. Pedersen (2011). *Frequent route based continuous moving object location-and density prediction on road networks.* In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 381–384. ACM.

[Gross et al., 2014] Gross, H-M, K. Debes, E. Einhorn, S. Mueller, A. Scheidig, C. Weinrich, A. Bley and C. Martin (2014). *Mobile Robotic Rehabilitation Assistant for walking and orientation training of Stroke Patients: A report on work in progress.* In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pp. 1880–1887. IEEE.

[Hafting et al., 2005] Hafting, Torkel, M. Fyhn, S. Molden, M.-B. Moser and E. I. Moser (2005). *Microstructure of a spatial map in the entorhinal cortex*. Nature, 436(7052):801–806.

[Han et al., 2004] Han, Bohyung, D. Comaniciu and L. Davis (2004). *Sequential kernel density approximation through mode propagation: applications to background modeling*. In *proc. ACCV*, vol. 4, pp. 818–823.

[Hart et al., 1968] Hart, Peter E, N. J. Nilsson and B. Raphael (1968). *A formal basis for the heuristic determination of minimum cost paths*. Systems Science and Cybernetics, IEEE Transactions on, 4(2):100–107.

[Hellbach, 2010] Hellbach, Sven (2010). *Entwicklung von Methoden zur Unterscheidung und Interpretation von Bewegungsmustern in dynamischen Szenen*. PhD thesis

[Hermes et al., 2009] Hermes, Christoph, C. Wohler, K. Schenk and F. Kummert (2009). *Long-term vehicle motion prediction*. In *Intelligent Vehicles Symposium, 2009 IEEE*, pp. 652–657. IEEE.

[Hu et al., 2006] Hu, Weiming, X. Xiao, Z. Fu, D. Xie, T. Tan and S. Maybank (2006). *A system for learning statistical motion patterns*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 28(9):1450–1464.

[Ikeda et al., 2013] Ikeda, Tetsushi, Y. Chigodo, D. Rea, F. Zanlungo, M. Shiomi and T. Kanda (2013). *Modeling and prediction of pedestrian behavior based on the sub-goal concept*. Robotics: Science and Systems VIII, p. 137.

[Jeung et al., 2008] Jeung, Hoyoung, Q. Liu, H. T. Shen and X. Zhou (2008). *A hybrid prediction model for moving objects*. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pp. 70–79. IEEE.

[Jeung et al., 2010] Jeung, Hoyoung, M. L. Yiu, X. Zhou and C. S. Jensen (2010). *Path prediction and predictive range querying in road network databases*. The VLDB Journal, 19(4):585–602.

[Jockusch and Ritter, 1999] Jockusch, Jan and H. Ritter (1999). *An instantaneous topological mapping model for correlated stimuli*. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, vol. 1, pp. 529–534. IEEE.

[Kellner et al., 2012] Kellner, Dominik, J. Klappstein and K. Dietmayer (2012). *Grid-based DBSCAN for clustering extended objects in radar data*. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pp. 365–370. IEEE.

[Kemeny and Snell, 1960] Kemeny, John G and J. L. Snell (1960). *Finite markov chains*, vol. 356. van Nostrand Princeton, NJ.

[Kohonen, 1990] Kohonen, Teuvo (1990). *The self-organizing map*. Proceedings of the IEEE, 78(9):1464–1480.

[Kolarow et al., 2012] Kolarow, Alexander, M. Brauckmann, M. Eisenbach, K. Schenk, E. Einhorn, K. Debes and H.-M. Gross (2012). *Vision-based hyper-real-time object tracker for robotic applications*. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2108–2115. IEEE.

[Kolarow et al., 2013] Kolarow, Alexander, K. Schenk, M. Eisenbach, M. Dose, M. Brauckmann, K. Debes and H.-M. Gross (2013). *APFel: The intelligent video analysis and surveillance system for assisting human operators*. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pp. 195–201. IEEE.

[Koller-Meier and Van Gool, 2002] Koller-Meier, Esther B and L. Van Gool (2002). *Modeling and recognition of human actions using a stochastic approach*. In *Video-Based Surveillance Systems*, pp. 179–191. Springer.

[Krumm, 2008] Krumm, John (2008). *A markov model for driver turn prediction*. Technical Report, SAE Technical Paper.

[Kuderer et al., 2012] Kuderer, Markus, H. Kretzschmar, C. Sprunk and W. Burgard (2012). *Feature-Based Prediction of Trajectories for Socially Compliant Navigation.*. In *Robotics: Science and Systems*.

[LaValle, 2006] LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

[Lengyel et al., 1990] Lengyel, Jed, M. Reichert, B. R. Donald and D. P. Greenberg (1990). *Real-time robot motion planning using rasterizing computer graphics hardware*, vol. 24. ACM.

[Lloyd, 1982] Lloyd, Stuart (1982). *Least squares quantization in PCM*. Information Theory, IEEE Transactions on, 28(2):129–137.

[Luber et al., 2010] Luber, Matthias, J. A. Stork, G. D. Tipaldi and K. O. Arras (2010). *People tracking with human motion predictions from social forces*. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 464–469. IEEE.

[Makhoul, 1975] Makhoul, John (1975). *Linear prediction: A tutorial review*. Proceedings of the IEEE, 63(4):561–580.

[Makris and Ellis, 2002] Makris, Dimitrios and T. Ellis (2002). *Spatial and Probabilistic Modelling of Pedestrian Behaviour.*. In *BMVC*, pp. 1–10. Citeseer.

[Malthouse and Blattberg, 2005] Malthouse, Edward C and R. C. Blattberg (2005). *Can we predict customer lifetime value?*. Journal of interactive marketing, 19(1):2–16.

[Martinetz et al., 1991] Martinetz, Thomas, K. Schulten et al. (1991). *A "Neural-Gas" Network Learns Topologies*. Artificial neural networks, pp. 397–402.

[McLachlan and Krishnan, 2007] McLachlan, Geoffrey and T. Krishnan (2007). *The EM algorithm and extensions*, vol. 382. John Wiley & Sons.

[Metropolis and Ulam, 1949] Metropolis, Nicholas and S. Ulam (1949). *The monte carlo method*. Journal of the American statistical association, 44(247):335–341.

[Mohler et al., 2007] Mohler, Betty J, W. B. Thompson, S. H. Creem-Regehr, H. L. Pick Jr and W. H. Warren Jr (2007). *Visual flow influences gait transition speed and preferred walking speed*. Experimental brain research, 181(2):221–228.

[Monreale et al., 2009] Monreale, Anna, F. Pinelli, R. Trasarti and F. Giannotti (2009). *Wherenext: a location predictor on trajectory pattern mining*. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 637–646. ACM.

[Müller et al., 2008] Müller, Steffen, S. Hellbach, E. Schaffernicht, A. Ober, A. Scheidig and H.-M. Gross (2008). *Whom to talk to? Estimating user interest from movement trajectories.*. In *RO-MAN*, pp. 532–538.

[Oliver et al., 2000] Oliver, Nuria M, B. Rosario and A. P. Pentland (2000). *A Bayesian computer vision system for modeling human interactions*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22(8):831–843.

[OpenStreetMap and contributors, 2013] OpenStreetMap and contributors (2013). *Planet.gpx*. http://planet.openstreetmap.org/gps/. Accessed: 2016-08-22.

[Pelleg et al., 2000] Pelleg, Dan, A. W. Moore et al. (2000). *X-means: Extending K-means with Efficient Estimation of the Number of Clusters.*. In *ICML*, pp. 727–734.

[Philippsen, 2004] Philippsen, Roland (2004). *Motion planning and obstacle avoidance for mobile robots in highly cluttered dynamic environments*. PhD thesis, École Polytechnique Fédérale de Lausanne.

[Piccardi, 2004] Piccardi, Massimo (2004). *Background subtraction techniques: a review*. In *Systems, man and cybernetics, 2004 IEEE international conference on*, vol. 4, pp. 3099–3104. IEEE.

[Piorkowski et al., 2009] Piorkowski, Michal, N. Sarafijanovic-Djukic and M. Grossglauser (2009). *CRAWDAD dataset epfl/mobility (v. 2009-02-24)*. http://crawdad.org/epfl/mobility/20090224. Accessed: 2016-08-22.

[Prasad and Agrawal, 2010] Prasad, Pratap S and P. Agrawal (2010). *Movement prediction in wireless networks using mobility traces*. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pp. 1–5. IEEE.

[Rabiner, 1989] Rabiner, Lawrence (1989). *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of the IEEE, 77(2):257–286.

[Sabanovic et al., 2006] Sabanovic, Selma, M. P. Michalowski and R. Simmons (2006). *Robots in the wild: Observing human-robot social interaction outside the lab*. In *Advanced Motion Control, 2006. 9th IEEE International Workshop on*, pp. 596–601. IEEE.

[Schenk et al., 2011] Schenk, Konrad, M. Eisenbach, A. Kolarow and H.-M. Gross (2011). *Comparison of laser-based person tracking at feet and upper-body height*. In *KI 2011: Advances in Artificial Intelligence*, pp. 277–288. Springer Berlin Heidelberg.

[Schenk et al., 2012a] Schenk, Konrad, A. Kolarow, M. Eisenbach, K. Debes and H. Gross (2012a). *Automatic calibration of multiple stationary laser range finders using trajectories*. In *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, pp. 306–312. IEEE.

[Schenk et al., 2012b] Schenk, Konrad, A. Kolarow, M. Eisenbach, K. Debes and H.-M. Gross (2012b). *Automatic calibration of a stationary network of laser range finders by matching movement trajectories*. In *Intelligent Robots* *and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 431–437. IEEE.

[Schroeter et al., 2013] Schroeter, Ch, S. Mueller, M. Volkhardt, E. Einhorn, C. Huijnen, H. van den Heuvel, A. van Berlo, A. Bley and H.-M. Gross (2013). *Realization and user evaluation of a companion robot for people with mild cognitive impairments.* In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1153–1159. IEEE.

[Siegwart et al., 2011] Siegwart, Roland, I. R. Nourbakhsh and D. Scaramuzza (2011). *Introduction to autonomous mobile robots.* MIT press.

[Silverman, 1986] Silverman, Bernard W (1986). *Density estimation for statistics and data analysis*, vol. 26. CRC press.

[Stentz, 1994] Stentz, Anthony (1994). *Optimal and efficient path planning for partially-known environments.* In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3310–3317. IEEE.

[Stricker et al., 2012] Stricker, Ronny, S. Müller, E. Einhorn, C. Schröter, M. Volkhardt, K. Debes and H.-M. Gross (2012). *Konrad and Suse, Two Robots Guiding Visitors in a University Building.* In *Autonomous Mobile Systems 2012*, pp. 49–58. Springer.

[Su and Drysdale, 1995] Su, Peter and R. L. S. Drysdale (1995). *A comparison of sequential Delaunay triangulation algorithms.* In *Proceedings of the eleventh annual symposium on Computational geometry*, pp. 61–70. ACM.

[Vasquez and Fraichard, 2004] Vasquez, Dizan and T. Fraichard (2004). *Motion prediction for moving objects: a statistical approach.* In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4, pp. 3931–3936. IEEE.

[Vasquez et al., 2009] Vasquez, Dizan, T. Fraichard and C. Laugier (2009). *Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion.* The International Journal of Robotics Research, 28:1486–1506.

[Vasquez Govea, 2007] Vasquez Govea, Alejandro Dizan (2007). *Incremental learning for motion prediction of pedestrians and vehicles.* PhD thesis, Grenoble, INPG.

[Verhein and Chawla, 2006] Verhein, Florian and S. Chawla (2006). *Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases*. In *Database Systems for Advanced Applications*, pp. 187–201. Springer.

[Von Mises, 1928] Von Mises, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit*. No. Bd. 3 in *Schriften zur wissenschaftlichen Weltauffassung*. J. Springer.

[Wanalertlak et al., 2011] Wanalertlak, Weetit, B. Lee, C. Yu, M. Kim, S.-M. Park and W.-T. Kim (2011). *Behavior-based mobility prediction for seamless handoffs in mobile wireless networks*. Wireless networks, 17(3):645–658.

[Weinrich et al., 2013] Weinrich, Christoph, M. Volkhardt, E. Einhorn and H.-M. Gross (2013). *Prediction of human collision avoidance behavior by lifelong learning for socially compliant robot navigation*. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 376–381. IEEE.

[Weser et al., 2006] Weser, Martin, D. Westhoff, M. Huser and J. Zhang (2006). *Multimodal people tracking and trajectory prediction based on learned generalized motion patterns*. In *Multisensor Fusion and Integration for Intelligent Systems, 2006 IEEE International Conference on*, pp. 541–546. IEEE.

[Whong, 2014] Whong, Chris (2014). *NYC's Taxi Trip Data*. http://chriswhong.com/open-data/foil_nyc_taxi/index.php. Accessed: 2016-08-22.

[Wiest et al., 2012] Wiest, Jürgen, M. Hoffken, U. Kresel and K. Dietmayer (2012). *Probabilistic trajectory prediction with Gaussian mixture models*. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pp. 141–146. IEEE.

[Xu et al., 2005] Xu, Rui, D. Wunsch et al. (2005). *Survey of clustering algorithms*. Neural Networks, IEEE Transactions on, 16(3):645–678.

[Ye et al., 2009] Ye, Yang, Y. Zheng, Y. Chen, J. Feng and X. Xie (2009). *Mining individual life pattern based on location history*. In *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on*, pp. 1–10. IEEE.

[Ying et al., 2011] Ying, Josh Jia-Ching, W.-C. Lee, T.-C. Weng and V. S. Tseng (2011). *Semantic trajectory mining for location prediction*. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 34–43. ACM.

[Yuan et al., 2011] Yuan, Jing, Y. Zheng, X. Xie and G. Sun (2011). *Driving with Knowledge from the Physical World*. In *SIGKDD 2011*. Association for Computing Machinery, Inc.

[Yuan et al., 2010] Yuan, Jing, Y. Zheng, C. Zhang, W. Xie, X. Xie and Y. Huang (2010). *T-Drive: Driving Directions Based on Taxi Trajectories*. In *ACM SIGSPATIAL GIS 2010*. Association for Computing Machinery, Inc.

[Zhang, 2009] Zhang, Lin (2009). *Beijing taxi trajectory dataset*. http://sensor.ee.tsinghua.edu.cn/datasets.html. Accessed: 2015-06-08.

[Zhang et al., 2011] Zhang, Wenzhu, L. Zhang, Y. Ding, T. Miyaki, D. Gordon and M. Beigl (2011). *Mobile sensing in metropolitan area: Case study in beijing*. In *Mobile Sensing Challenges Opportunities and Future Directions, Ubicomp2011 workshop*.

[Zheng, 2011] Zheng, Yu (2011). *T-Drive sample dataset*. https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/. Accessed: 2016-08-22.

[Zheng, 2012] Zheng, Yu (2012). *GeoLife GPS Trajectories*. https://www.microsoft.com/en-us/download/details.aspx?id=52367. Accessed: 2016-08-22.

[Zheng et al., 2008] Zheng, Yu, Q. Li, Y. Chen, X. Xie and W.-Y. Ma (2008). *Understanding mobility based on GPS data*. In *Proceedings of the 10th international conference on Ubiquitous computing*, pp. 312–321. ACM.

[Zheng et al., 2010] Zheng, Yu, X. Xie and W.-Y. Ma (2010). *GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory.*. IEEE Data Eng. Bull., 33(2):32–39.

[Zheng et al., 2009] Zheng, Yu, L. Zhang, X. Xie and W.-Y. Ma (2009). *Mining interesting locations and travel sequences from GPS trajectories*. In *Proceedings of the 18th international conference on World wide web*, pp. 791–800. ACM.

[Zhu et al., 2013] Zhu, Bing, Q. Huang, L. Guibas and L. Zhang (2013). *Urban Population Migration Pattern Mining Based on Taxi Trajectories*. In *Mobile Sensing, 3rd International Workshop on*.

[Ziebart et al., 2009] Ziebart, Brian D, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey and S. Srinivasa (2009). *Planning-based prediction for pedestrians*. In *Intelligent Robots and*

Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pp. 3931–3936. IEEE.

[Zivkovic, 2004] Zivkovic, Zoran (2004). *Improved adaptive Gaussian mixture model for background subtraction.* In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2, pp. 28–31. IEEE.

Anlage 1

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung folgenden Materials haben mir die nachstehend aufgeführten Personen in der jeweils beschriebenen Weise entgeltlich/unentgeltlich[1)] geholfen:

1. ....
2. ....
3. ....

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch bewertet wird und gemäß § 7 Abs. 10 der Promotionsordnung den Abbruch des Promotionsverfahrens zur Folge hat.

(Ort, Datum) *(Unterschrift)*

1) Unzutreffendes bitte streichen.

Annex 1

Declaration

I certify that I prepared the submitted thesis independently without undue assistance of a third party and without the use of others than the indicated aids. Data and concepts directly or indirectly taken over from other sources have been marked stating the sources.

When selecting and evaluating the following materials, the persons listed below helped me in the way decribed respectively for a charge/free of charge[1)]:

1. ....
2. ....
3. ....

Further persons were not involved in the content-material-related preparation of the thesis submitted. In particular, I have not used the assistance against payment offered by consultancies or placing services (doctoral consultants or other persons). I did not pay any money to persons directly or indirectly for work or services which are related to the content of the thesis submitted.

So far the thesis have not been submitted identically or similarly to an examination office in Germany or abroad.

I have been notified that any incorrectness in the submitted above mentioned declaration is assessed as attempt to deceive and, according to § 7 para. 10 of the PhD regulations, this leads to a discontinuation of the doctoral procedure.

(city, date) *(signature)*

1) Delete as applicable.