Technische Universität München
TUM School of Computation, Information and Technology

Robust and Probabilistic Motion Prediction for Intelligent Infrastructure Systems

Christoph Schöller, M.Sc.

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Pramod Bhatotia

Prüfer*innen der Dissertation:

1. Prof. Dr.-Ing. habil. Alois Knoll
2. Asst. Prof. Dr. Igor Gilitschenski

Die Dissertation wurde am 19.05.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.09.2022 angenommen.

Abstract

Autonomous driving promises various benefits to its users, such as improved comfort, more ecological transportation, and – most importantly – higher safety than manual driving. One critical function for the safety of autonomous vehicles is to predict the motion of other traffic participants. Only like this can the vehicle avoid collisions and drive with reasonable speed and efficiency. Using neural networks for this task has become increasingly popular in recent years. However, despite their excellent performance on benchmarks, even the latest neural prediction models suffer from limited generalization and either oversimplified or intractable probabilistic modeling. Furthermore, the best prediction model can only function with a precise state estimate of the traffic and environment. But vehicle perception is not perfect. It can miss detecting other vehicles and is prone to suffer from occlusions due to low perspectives.

In this work, we first confirm these generalization problems of prediction models by showing that, in specific scenarios, state-of-the-art approaches can be outperformed by a simple prediction baseline. To explain these results, we conduct an extensive analysis of how neural prediction models process their inputs and what affects their robustness and generalization. This analysis reveals pitfalls in training prediction networks and clarifies false assumptions about motion prediction. Because our baseline experiments also indicate that generative prediction models struggle to learn the true distribution of the agents' future motion, we propose a novel prediction network based on normalizing flows. Our model can correctly learn the target distribution, it enables tractable likelihood computation, and is context-aware. To achieve reliable perception, we present the intelligent infrastructure system Providentia. The system creates a digital twin of the observed road, and we show that this information can be communicated to a test vehicle to enhance and complement its onboard perception. Furthermore, we perform an extensive holistic system evaluation using aerial images for ground truth generation. Due to its high reliability, Providentia serves as an ideal platform for motion prediction. Therefore, we integrate our prediction model into the system and evaluate its performance and runtimes with real-world data from our testbed.

This work provides an alternative perspective for making motion prediction and environment perception more reliable. We hope that our contributions will help to enable safe autonomous driving in the future. Ultimately, that is the key to its acceptance in society and adoption.

Zusammenfassung

Autonomes Fahren verspricht eine Vielzahl von Vorteilen für seine Nutzer, unter anderem mehr Fahrkomfort, verbesserte Umweltfreundlichkeit und insbesondere eine erhöhte Sicherheit. Um diese Sicherheit zu gewährleisten, ist die präzise Bewegungsvorhersage anderer Verkehrsteilnehmern notwendig. Nur so können Unfälle vermieden und mit angemessener Geschwindigkeit und Effizienz gefahren werden. In den letzten Jahren haben neuronale Netze zunehmend an Popularität zur Bewegungsprädiktion gewonnen. Jedoch weisen selbst neuste neuronale Netze trotz guter Prädiktionsgenauigkeit Probleme bezüglich ihrer Generalisierung auf und nutzen häufig nur einfache oder nicht direkt evaluierbare Wahrscheinlichkeitsverteilungen. Des Weiteren kann selbst ein ideales Prädiktionsmodell nur dann gute Ergebnisse liefern, wenn es akkurate Messergebnisse als Eingangsdaten erhält. Diese Positionsmessungen stammen jedoch aus der Fahrzeugperzeption, welche unter Fehldetektionen und aufgrund der niedrigen Fahrzeugperspektive unter Verdeckungen leidet.

Um die beschriebenen Generalisierungsprobleme von neuronalen Prädiktionsmodellen nachzuweisen, führen wir zuerst ein Experiment durch. In diesem zeigen wir, dass häufig selbst einfachste Methoden wie die Annahme von zukünftig konstanter Geschwindigkeit und Richtung in bestimmten Szenarien genauer sind als moderne neuronale Netze. Um diese Beobachtung zu erklären, analysieren wir, wie neuronale Netze ihre Eingangsdaten verarbeiten und was ihre Robustheit beeinträchtigt. Unsere Analyse zeigt Fallstricke beim Training von neuronalen Prädiktionsmodellen auf und identifiziert falsche Annahmen über Bewegungsprädiktion. Die Ergebnisse unseres Experiments mit einfachen Modellen deutet ebenfalls darauf hin, dass generative neuronale Prädiktionsmodelle nicht die tatsächliche Bewegungsverteilung der Agenten lernen. Um dieses Problem zu lösen, entwickeln wir in dieser Arbeit ein Modell basierend auf Normalizing Flows, welches es ermöglicht, die korrekte Verteilung zu lernen und die Berechnung von Likelihoods in geschlossener Form zulässt. Zudem integrieren wir Interaktions- und Umgebungsinformationen in dieses Modell. Um die verlässliche Wahrnehmung zu erreichen, die unser Prädiktionsmodell benötigt, stellen wir das intelligente Infrastruktursystem Providentia vor. Diese Arbeit beschreibt detailliert die Architektur und den Aufbau dieses Systems und kann in Zukunft als Muster für die Entwicklung ähnlicher Systeme genutzt werden. Unser System ermöglicht es in Echtzeit ein digitales Abbild der Straße zu generieren und wir zeigen, dass dieses Abbild in die Wahrnehmung autonomer Fahrzeuge integriert werden kann, um diese

zu vervollständigen und Lücken zu füllen. Außerdem führen wir eine ganzheitliche Evaluation der Systemleistung durch, für welche wir lokalisierte Luftbilder nutzen. Wir zeigen, dass unser System präzise und zuverlässig ist und deshalb als gute Datenbasis zur Bewegungsprädiktion dient. Anschließend beschreiben wir die Integration unseres Vorhersagemodells in das System und evaluieren seine Prädiktionsgenauigkeit und Ausführungslaufzeit auf realen Systemdaten unseres Testfeldes.

Diese Arbeit zeigt aus einer alternativen Perspektive, wie verlässliche Bewegungsprädiktion mit neuronalen Netzen gestaltet werden kann und wie diese mit der Wahrnehmung eines Infrastruktursystems interagiert. Wir hoffen, dass unsere Forschungsergebnisse einen Beitrag zur Sicherheit zukünftiger autonomer Fahrzeuge leisten wird. Diese Sicherheit ist der Schlüssel zur gesellschaftlichen Akzeptanz und zum Vertrauen in autonomes Fahren.

Acknowledgments

This work would not have been possible without help and support. First, I would like to thank my advisor, Prof. Alois Knoll, for his support throughout my PhD and for giving me the freedom to choose the topics I am interested in the most.

I thank the Providentia team for their hard and successful work on our project. A special thanks to Annkathrin Krämmer for the interesting conversations, discussions, great teamwork, and helping me with my mathematical understanding on many occasions. Furthermore, I thank all members of my research group for the collaborations, scientific exchange, and tips on navigating my PhD, especially at the beginning.

Moreover, I would like to thank Vincent Aravantinos for his supervision in my first PhD year. Thanks to Abhinav Valada and Maxim Tatarchenko for their feedback on my work and general advice on publishing.

I want to thank my friends who helped me balance work and fun in my life, namely Elena Haag, Iris Felsch, Matei Pavaluca, Anca Stefanoiu, Tiffany Fernandez, Anatoli Domashnev, Branka Mirchevska, Semir Festic, Toshika Srivastava, and Nicolas Bihler.

Thanks to my parents, who enabled my whole academic journey. You let me do what I am passionate about in life and supported me all the way through. In the most stressful and daunting times of my life and work, the person standing by my side was my wife, Eteri Sokhoyan. You helped me when I doubted myself, kept me optimistic and focused, and celebrated every success with me, both in private and professional life. Thank you for everything.

Lastly, I dedicate this work to my son Aren.

Notation

This chapter concisely describes the basic mathematical notation and symbols used throughout this thesis. Special notation not contained in this list will be explained when used.

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix or tensor
$\boldsymbol{A}^{-1}$	Inverse of matrix $\boldsymbol{A}$
$\boldsymbol{A}^{\top}$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{I}$	Identity matrix
a_i	Element i of vector $\boldsymbol{a}$
$a_{i:n}$	Elements of vector $\boldsymbol{a}$ from index i to n
$a_{>i}$	Elements of vector $\boldsymbol{a}$ with index greater than i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$a \cdot b$	Scalar multiplication
$\boldsymbol{a} \cdot \boldsymbol{b}$	Dot product
$\boldsymbol{a} \odot \boldsymbol{b}$	Element-wise (Hadamard) product
$\boldsymbol{a} \oplus \boldsymbol{b}$	Vector or tensor concatenation
$\|\boldsymbol{x}\|$	l^2-norm of vector $\boldsymbol{x}$
$\|x\|$	Absolute value of scalar x
$\dim(\boldsymbol{x})$	Number of dimensions of $\boldsymbol{x}$
$\det \boldsymbol{A}$	Determinant of matrix $\boldsymbol{A}$
$\{0, ..n\}$	Set of integers from 0 to n
$(0, ..n)$	Ordered tuple of integers from 0 to n
$[a, b]$	Real interval including a and b
$]a, b]$	Real interval excluding a, but including b

$p(\boldsymbol{x})$	Probability density over variable $\boldsymbol{x}$
$\boldsymbol{x} \sim p(\boldsymbol{x})$	$\boldsymbol{x}$ sampled from random variable following $p(\boldsymbol{x})$
$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})]$	Expectation of $f(\boldsymbol{x})$, following distribution $p(\boldsymbol{x})$
$D_{\mathrm{KL}}(p(\boldsymbol{x}) \parallel q(\boldsymbol{x}))$	Kullback-Leibler divergence between distribution p and q
$f_\theta(\boldsymbol{x})$	Function f parameterized by values θ
$\log x$	Logarithm of value x
$f_1 \circ f_2$	Composition of function f_1 and function f_2
$\frac{\mathrm{d}x}{\mathrm{d}y}$	Derivative of x with respect to y
$\frac{\partial dx}{\partial dy}$	Partial derivative of x with respect to y
∇f	Gradient of function f
$J_f(\boldsymbol{x})$	Jacobian matrix of function $f(\boldsymbol{x})$, evaluated at $\boldsymbol{x}$

Contents

Chapter 1

Introduction

This chapter first explains which problems in autonomous driving this thesis aims to solve and why their solutions are essential. We then outline our key contributions to these problems and list the publications in which the research of this work resulted. Lastly, we state to which extent we collaborated with other researchers in this work.

1.1 Motivation

In the last decade, significant progress in the development of autonomous vehicles and Advanced Driver Assistance Systems (ADAS) has been made. While academia largely drove the initial research and advances in this area [1]–[3], the realization of autonomous driving has now become an important goal for many car manufacturers and technology companies too. Autonomous driving promises a multitude of benefits to its users. Among these benefits are more comfortable, ecological, and economical transportation. However, the primary reason for the development of fully autonomous driving is probably safety. While the numbers of traffic-related deaths in Germany and most other developed countries are slowly decreasing [4], [5], they are still significant.

Though increased traffic safety is a goal of autonomous driving, high safety is also a precondition for its deployment and widespread adoption. An autonomous vehicle is a complex system with many safety-critical functions and components [6]. One of these is its ability to predict the behavior of other traffic participants, including mixed traffic with human drivers, pedestrians, and cyclists. Only with accurate predictions of their future motion an autonomous vehicle can avoid collisions and drive with reasonable speed and efficiency. Furthermore, a good motion prediction helps to avoid sudden and unexpected maneuvers that significantly reduce the passengers' comfort.

An ideal prediction model must be accurate and reliable. It should function across various environments, consider interactions between traffic participants, and use contextual clues like road layouts [7]. Because there are many possible future trajectories that each traffic participant can take, a good prediction model should be based on sound probabilistic theory and model motion as a probability distribution that reflects all possible maneuvers. In recent years, due to their rise in popularity in a various research areas [8]–[11], Artificial Neural Networks (ANNs) have become popular for use as prediction models. However, while many demonstrated great performance,

even the latest neural prediction models suffer from limited generalization and either oversimplified or intractable probabilistic modeling.

Furthermore, even an ideal prediction model can only work with a precise state estimate of the traffic and environment. Only when it is known where exactly the vehicles are and where they were in the past is it possible to make good predictions for their future movement. Autonomous vehicles perceive the state of their surrounding environment with onboard sensors and state-of-the-art algorithms for object detection. However, even state-of-the-art object detection models fail at times [12], [13], for example, in cases when the vehicles or the people to detect are too far away or when the weather and lighting conditions are not ideal. Furthermore, due to their low perspective, the perception of autonomous vehicles is subject to severe occlusions that they cannot resolve on their own.

Intelligent Infrastructure Systems (IISs) can help to alleviate these problems [14]. By placing static sensors above the road, they have a superior perspective on the traffic compared to the vehicles themselves. The sensors in such a distributed system could also cover the road redundantly from various perspectives, and by using sensors of different modalities, the weaknesses of one sensor type can be compensated by the strengths of another. By positioning the sensors this way, such a system is not as prone to occlusions and makes it possible to get more reliable state estimates of the traffic. The system's resulting perception can be used for reliable motion prediction and to fill in gaps in the perception of autonomous vehicles. However, in the literature, these types of sensor systems are not well explored, and little is known about how to build such a system in practice. Furthermore, it is unknown what performance we can expect of such an IIS.

This work addresses the problems mentioned above, and our contribution towards their solution consists of two parts. In the first part, we consider current problems in neural motion prediction and aim to improve these. In the second part, we study how an IIS could contribute to gathering information for reliable motion prediction and how it can enhance the perception of autonomous vehicles. In particular, we aim to answer the following research questions:

- What makes neural motion prediction models robust and generalize well across various scenarios and behaviors?
- How can we build a generative prediction model that learns the true underlying motion distribution and is tractable?
- How can we design and use an IIS to assist autonomous vehicles in their perception and reduce occlusions?
- How can we evaluate the reliability and performance of such a safety-critical system, that is crucial for the assistance of autonomous vehicles?

- How can we integrate a motion prediction model that fulfills the properties we desire into the IIS we described, and what performance can we expect?

We hope to contribute to the safety and reliability of autonomous driving by answering these questions, ultimately improving their acceptance in society, and accelerating their deployment in realistic use-cases.

1.2 Key Contributions

In this work, we make multiple scientific contributions to motion prediction and environment perception. Our goal is to answer the research questions we posed in the previous section. In the following, we briefly describe the contributions of this work towards this goal.

Robustness and Generalization of Neural Prediction Models: Motion prediction is part of an autonomous vehicle's extended perception and is paramount for safe planning and driving. In recent years, neural networks have become very popular for this task and demonstrated excellent performance. However, we know little about their limitations in predicting motion and how to use them safely. In Ch. 3 we demonstrate that, surprisingly, in certain scenarios, even a simple prediction baseline can achieve state-of-the-art performance and outperform neural networks. The success of this baseline raises questions about whether neural networks can utilize all their provided inputs and how these impact their predictions. Therefore, we systematically analyze how these neural prediction models process their input information and what affects their robustness and generalization. In particular, we analyze the impact of environmental priors, the target agent's motion history, and interaction information on the network's predictions. Our results reveal pitfalls in training neural networks and clarify false assumptions about neural motion prediction itself.

Probabilistic and Context-Aware Motion Prediction: Because our experiments with simple baselines indicate that most modern generative prediction models struggle to learn the true underlying distribution of agents' future motion, in Ch. 4 we propose a novel prediction model based on normalizing flows. Normalizing flows are a technique to iteratively mold a simple base distribution into a complex target distribution. Because each transformation in this iterative process is invertible, normalizing flows allow computing the likelihood of a predicted sample in terms of the simple base distribution. By building a motion prediction model with normalizing flows, we can train it directly with Maximum Likelihood Estimation (MLE) and ensure that our model learns the actual target distribution. Our model is also tractable and allows the fast computation of likelihoods for its predictions, which is an important property for use in downstream tasks, like motion planning in autonomous vehicles. We also propose a module based on a fine-grained attention mechanism to make our model interaction- and context-aware. Our evaluation shows that our model FloMo can make accurate predictions for both

pedestrians and vehicles and achieves state-of-the-art results on several popular prediction benchmarks. Furthermore, we propose a scaling augmentation for trajectories during training that improves data diversity and performance, and a noise injection method for training stabilization.

Intelligent Infrastructures System for Perception: In Ch. 5, we present the IIS Providentia that can capture the traffic on a designated stretch of highway and represent it as a digital twin, i.e., a digital representation of the highway's state. Our system achieves this by perceiving the road with a collection of multimodal sensors that are mounted on gantry bridges over the road. Because its sensors are observing the highway redundantly and due to their superior point of view, it can resolve most inter-vehicle occlusions and compensate detection faults. The system then provides this information to vehicles that pass through it and enables them to integrate the received information into their onboard environment perception. By this, the vehicles can close their perception gaps and use our system's digital twin for safe action planning. In our work, we describe how the system was built and explain its hardware and software architecture in detail. An accurate description of a system like ours, along with its challenges, is novel in literature and can serve as a blueprint in the future. We also qualitatively demonstrate the integration of the system's perception in that of a test vehicle and conduct an extensive experiment to evaluate our system's positioning accuracy and detection performance. For this experiment, we use aerial imagery to obtain the reliable ground truth that we need for evaluation. With our experiments, we show that Providentia is suited for enhancing autonomous vehicle perception and that it serves as a reliable data source for motion prediction.

Motion Prediction in Infrastructure Systems: The prediction model FloMo that we propose in this work is intended to complement our IIS Providentia. Therefore, in Ch. 6 we describe the integration of our model in the system and how we utilize Providentia's digital twin to predict the future motion of vehicles that pass through our testbed. Because Providentia is a safety-critical system, we must ensure that the predictions of FloMo on real data from our system are accurate. Furthermore, the inference of our model must be fast, such that it does not become a bottleneck in the system. To ensure these properties, we evaluate different versions of our prediction model and analyze the influence of specific architecture configurations and parameters on its prediction performance and runtime.

1.3 Publications

We previously published parts of the research presented in this thesis in workshop proceedings, scientific magazines, conferences, and journals. Where each publication contributed in particular will be noted in the corresponding chapters. The following is a

chronological list of first-author publications that contributed to this work:

- A. Krämmer*, C. Schöller*, D. Gulati, A. Knoll, "Providentia - A Large Scale Sensing System for the Assistance of Autonomous Vehicles," in *Robotics: Science and Systems (RSS), Workshop on Scene and Situation Understanding for Autonomous Driving*, 2019.
- A. Krämmer*, C. Schöller*, F. Kurz, D. Rosenbaum, A. Knoll, "Vorausschauende Wahrnehmung für sicheres automatisiertes Fahren," in *Internationales Verkehrswesen*, 2020.
- C. Schöller, V. Aravantinos, F. Lay, A. Knoll, "What the constant velocity model can teach us about pedestrian motion prediction," in *IEEE Robotics and Automation Letters (RA-L)*, 2020.
- C. Schöller, A. Knoll, "FloMo: Tractable Motion Prediction with Normalizing Flows," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.
- A Krämmer*, C. Schöller*, D. Gulati, V. Lakshminarasimhan, F. Kurz, D. Rosenbaum, C. Lenz, A. Knoll, "Providentia - A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation," in *Journal of Field Robotics*, 2022.

Contributions in the areas of multimodal sensor calibration and planning under uncertainty are outside this thesis's scope. Therefore, they were not directly used in this work but are listed for completeness:

- C. Schöller*, M. Schnettler*, A. Krämmer, G. Hinz, M. Bakovic, M. Güzet, A. Knoll, "Targetless Rotational Auto-Calibration of Radar and Camera for Intelligent Transportation Systems," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.
- J. Bernhard, P. Hart, A. Sahu, C. Schöller, M. G. Cancimance, "Risk-Based Safety Envelopes for Autonomous Vehicles Under Perception Uncertainty," in *IEEE Intelligent Vehicles Symposium (IV)*, 2022.

1.4 Collaborations

Some of the research in this thesis involved collaborations with other researchers. As the supervisor of this thesis, Prof. Alois Knoll contributed through scientific discussions. Other collaborations beyond this supervision are listed below.

* denotes equal contribution.

- Chapter 5: Building the Providentia system was a large multi-year project that required knowledge about sensors, hardware for heavy computation, networking, software architecture and engineering, and artificial intelligence. Hence, a large team with a variety of skills was necessary for its build-up. The system's hardware was mainly designed and implemented by Philipp Quentin and Maximilian Schnettler, Venkatnarayanan Lakshminarasimhan and Juri Kuhn designed the system's networking, Claus Lenz implemented the object detection with his Cognition Factory GmbH, and its data fusion and tracking is primarily the work of Annkathrin Krämmer. Furthermore, Martin Büchel designed and operated the test vehicle used to demonstrate our system's capability to enhance vehicle perception. The author of this thesis was involved in the system's calibration, the design of its software architecture, data labeling, and the implementation of data recording and processing tools. Moreover, he implemented its initial prediction module, additive services (like collision warnings and lane recommendations), and its position refinement module. Together with Annkathrin Krämmer, the author conceptualized and implemented the system's evaluation. This evaluation includes the preparation of the aerial ground truth, data cleaning and refinement, the data association algorithm, evaluation metrics, and the overall experiment setups. The German Aerospace Center (DLR) recorded the aerial imagery and applied object detection for this evaluation.
- Chapter 6: The integration of our advanced prediction module (see Sec. 6.2) is primarily the work of the student Jurek Olden as part of a guided research project and his master's thesis at the Technical University of Munich. The author of this thesis provided the trained model and contributed to the software design of this integration with technical discussions and advice.

Chapter 2

Background

In this chapter, we briefly explain the mathematical notation and the theoretical concepts necessary to understand this thesis. In particular, we first formally introduce the problem of predicting motion, which – applied in the Intelligent Infrastructure System (IIS) domain – is the focus of this work. Then we describe several types of neural networks that are frequently used for motion prediction. In the next section, we explain Normalizing Flows, the probabilistic machine learning concept that our new prediction model is based on. Lastly, we explain sensor calibration and coordinate transformations, a central mathematical concept in a distributed sensor system like the one we propose in this work.

2.1 Motion Prediction

Predicting the motion of agents or objects is a fundamental problem in computer vision and robotics. It is required for various perception algorithms, like tracking and state estimation, and enables autonomous agents to plan safe and collision-free paths.

While various notions of motion prediction exist, we focus on predicting the motion of traffic agents as trajectories of point estimates in this work. In particular, a trajectory is a finite sequence $\boldsymbol{\phi}_i = (\boldsymbol{p}_i^0, ..., \boldsymbol{p}_i^T)$ of positions $\boldsymbol{p}_i^t = (x_i^t, y_i^t)$ of traffic agent i over discrete time steps $t \in (0, ..., T)$. The time discretization depends on the update frequency with which the perceiving system can detect and track the traffic agents. This work assumes that each position $\boldsymbol{p}^t$ in such a trajectory represents the tracked object's geometric center, projected on a ground plane. To predict the future motion $\boldsymbol{x}_i = (\boldsymbol{p}_i^{t+1}, ..., \boldsymbol{p}_i^{t+n})$ of the target agent, only its past trajectory $\boldsymbol{o}_i = (\boldsymbol{p}_i^0, ..., \boldsymbol{p}_i^t)$ up to a time step t can be observed.

While it is possible to address this prediction problem with rule- or physics-based methods, this work focuses on data-driven models. In order to train a supervised machine learning model $f_\theta(\boldsymbol{o}_i) = \hat{\boldsymbol{x}}_i$ for motion prediction, trajectories in a recorded dataset $\boldsymbol{\phi} \in \mathcal{T}$ can be split into an observed and prediction part. Then the model's parameters are typically optimized as follows:

$$\arg\min_\theta \frac{1}{N} \sum_i^N L(f_\theta(\boldsymbol{o}_i), \boldsymbol{x}_i). \tag{2.1}$$

Here, $L(f_\theta(\boldsymbol{o}_i), \boldsymbol{x}_i)$ is a distance measure that quantifies the dissimilarity between the predicted trajectory $\hat{\boldsymbol{x}}_i$ and $\boldsymbol{x}_i$, and N denotes the size of dataset $\mathcal{T}$. In practice, the squared error is commonly used to measure dissimilarity. Like this, the model will learn to predict the trajectory with the smallest expected error.

2.1.1 Probabilistic Perspective

Depending on the scenario, there are often multiple possible future trajectories that an agent may take. For example, a vehicle can turn right, left, or drive straight ahead on a cross-road. Taking this into account while making predictions can reflect reality more accurately.

We can achieve this by viewing motion prediction as a probabilistic problem using a generative model instead of a regression model. Here the goal is to learn a multivariate conditional probability distribution $p(\boldsymbol{x}_i \mid \boldsymbol{o}_i)$ over future trajectories. How we can train such a model depends on its capabilities, but ideally, a variant of Maximum Likelihood Estimation (MLE) is used.

The learned distribution $p(\boldsymbol{x}_i \mid \boldsymbol{o}_i)$ then covers all possible outcomes of how an agent will move, along with their corresponding likelihoods. Hence, making a concrete prediction is achieved by simply sampling a future trajectory $\hat{\boldsymbol{x}}_i \sim p(\boldsymbol{x}_i \mid \boldsymbol{o}_i)$ from the model.

2.1.2 Context Aware Prediction

Besides only conditioning the trained model on the target agent's motion history, we can also use additional context information. For example, the future motion of the target agent also depends on how surrounding agents will move. By additionally providing the model with information of their motion histories $\{\boldsymbol{o}_j : j \neq i\}$, the model can infer possible colliding paths that the target agent most likely will not take and instead predict those that become more likely. Other relevant information could be the layout of the agent's environment or additional semantic environment information.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning models that are inspired by biological neural networks and date back to the 1940s [15]. However, with the increase of computational power in recent years, neural networks have become dominant in various areas of artificial intelligence and computer vision. As this includes motion prediction, in this chapter, we explain the basic and most relevant concepts that are necessary to understand to follow the rest of this work.

2.2.1 Feed-Forward Neural Networks

Feed-Forward Neural Networks (FNNs), also known as Multilayer Perceptron (MLP), are the most basic type of neural network. As its name suggests, the input it receives is

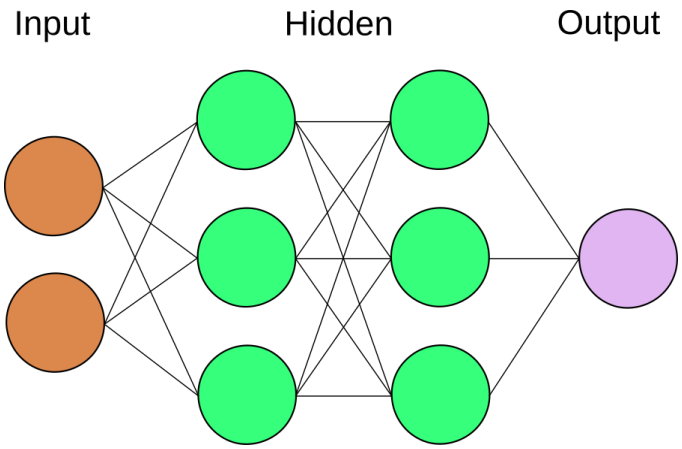


Figure 2.1: The architecture of a simple FNN with two hidden layers. The circles represent neurons, each column is a layer in the network, and the connecting lines represent the parameters that connect different neurons.

only flowing in the forward direction through such a network to the output.

The smallest unit of a FNN is the artificial neuron. Such a neuron receives a numerical input vector $\boldsymbol{x}$, multiplies each input dimension x_i with a weight w_i and adds a bias term b. The resulting scalar value is then passed through a non-linear activation function σ, such as a Rectified Linear Unit (ReLU) [16]. In vector notation, this can be denoted as

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T \boldsymbol{w} + b). \tag{2.2}$$

A FNN consists of multiple layers of artificial neurons. Arranging neurons in layers makes it possible to realize vector-valued functions that are efficiently computable in matrix form. The first layer is the input layer, and the last layer is the output layer. Depending on the use-case, the output layer omits activation functions not to limit the numeric output range. Each layer in between is called a hidden layer of the neural network. By passing the input vector $\boldsymbol{x}$ through each subsequent layer, it is transformed into the desired output. Fig. 2.1 illustrates the architecture of a simple FNN with a two-dimensional input, two hidden layers with three neurons each, and a one-dimensional output.

To train a neural network, we define a loss function that quantifies the discrepancy between the network's actual output and its desired output. By computing the derivative of the network's parameters for the loss, we can optimize its weights for a dataset with stochastic gradient descent algorithms [17]. The backpropagation algorithm can realize this gradient computation efficiently by minimizing the necessary computations [15].

2.2.2 Recurrent Neural Networks

Sequential prediction problems like machine translation have a varying lengths of input and output data [10]. Motion prediction can be interpreted as a sequence-to-sequence

prediction problem too. Because the input and output dimensions in FNNs are fixed, they are not suited for predicting such sequences. Recurrent Neural Network (RNN) address this problem with internal memory, their hidden state. They can be called in a loop and store information about earlier time steps. A basic RNN can be defined as

$$\boldsymbol{h}^{t+1} = f(\boldsymbol{x}^t, \boldsymbol{h}^t; \theta), \tag{2.3}$$

where $\mathbf{h}^{t+1}$ is the new hidden state vector, $\mathbf{x}^t$ is the input at time step t, $\mathbf{h}^t$ is the last hidden state and θ the network's parameters, e.g., its weights. Typically an RNN will use an additional output layer or sub-network to transform $\mathbf{h}^{t+1}$ to $\mathbf{x}^{t+1}$. The function f can be implemented with a FNN, and its inputs $\mathbf{x}^t$ and $\mathbf{h}^t$ are concatenated. The most common way to train a RNN is to compute the network's prediction error over multiple time steps and aggregate it, i.e.,

$$L(\theta) = \frac{1}{T} \sum_{t=0}^{T} d(\hat{\boldsymbol{y}}^t - \boldsymbol{y}^t). \tag{2.4}$$

Here $\hat{\boldsymbol{y}}^t$ is the network's predicted output for time step t, and the function d measures the discrepancy between prediction and ground truth. To train the RNN, we can use gradient descent with Backpropagation through time (BPTT) time [15]. In BPTT, the network is unrolled through time, but its weights are shared at each time step. The resulting computational graph can then be optimized with regular backpropagation.

Long Short-term Memories

Training neural networks with gradient descent requires repeatedly multiplying gradients of different layers in the backpropagation. As a result, the gradients in earlier layers can become either very small or very large. These phenomena are called the vanishing and exploding gradient problems, respectively [18], [19].

In recurrent neural networks that process long sequences, these issues become even more severe than in FNNs. To address this problem, Hochreiter et al. [20] propose the Long Short-term Memory (LSTM), that is a specialized type of RNN. The LSTM is defined as

$$\begin{aligned}
\boldsymbol{f}^t &= \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_f) \\
\boldsymbol{i}^t &= \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{h}^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}} &= \tanh(\boldsymbol{W}_c \cdot [\boldsymbol{h}^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_c) \\
\boldsymbol{c}^t &= \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \odot \tilde{\boldsymbol{c}} \\
\boldsymbol{o}^t &= \sigma(\boldsymbol{W}_o \cdot [h^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_o) \\
\boldsymbol{h}^t &= \boldsymbol{o}^t \tanh(\boldsymbol{c}^t).
\end{aligned} \tag{2.5}$$

As its activation function σ, it uses the sigmoid function that outputs values in range $]0, 1[$. Instead of only a hidden state like in the basic RNN, a LSTM also has an additional

cell state $\boldsymbol{c}$. In particular, in one step from $t-1$ to t, it first computes a forget gate $\boldsymbol{f}^t$, an input gate $\boldsymbol{i}^t$ and the cell update $\tilde{\boldsymbol{c}}$. All of these quantities depend on the last hidden state $\boldsymbol{h}^{t-1}$ and the new input $\boldsymbol{x}^t$, which are concatenated. Then the new cell state is updated by multiplying the old cell state with the forget gate and then adding values from the cell update that are selected with the input gate. The final output is computed by transforming $\boldsymbol{c}^t$ with a non-linear tanh activation and multiplying with the output gate $\boldsymbol{o}^t$.

In a regular RNN, that uses a sigmoid or tanh as its activation function, the influence of an old hidden state $\boldsymbol{h}$ on the output will decrease and eventually fade for long sequences. This effect can be shown by calculating the gradient of $\boldsymbol{h}^t$ regarding the network's output. In an LSTM this problem is circumvented with the cell state $\boldsymbol{c}$. By updating the cell state linearly with $\boldsymbol{f}^t$ and $\boldsymbol{i}^t$, the LSTM has the option to simply not modify $\boldsymbol{c}^{t-1}$ and retain old information over very long sequences.

Gated Recurrent Units

The Gated Recurrent Unit (GRU) [21] can be considered a simplified version of the LSTM. Its performance and capability to handle dependencies in long-term sequences is comparable to that of the LSTM. However, due to its simplicity it requires fewer parameters, and hence it is smaller to store and faster to execute. A GRU is defined as

$$\begin{aligned} \boldsymbol{z}^t &= \sigma(\boldsymbol{W}_z \cdot [\boldsymbol{h}^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_z) \\ \boldsymbol{r}^t &= \sigma(\boldsymbol{W}_r \cdot [\boldsymbol{h}^{t-1} \oplus \boldsymbol{x}^t] + \boldsymbol{b}_r) \\ \tilde{\boldsymbol{h}}^t &= \tanh(\boldsymbol{W} \cdot [(\boldsymbol{r}^t \odot \boldsymbol{h}^{t-1}) \oplus \boldsymbol{x}^t]) \\ \boldsymbol{h}^t &= (\boldsymbol{1} - \boldsymbol{z}^t) \odot \boldsymbol{h}^{t-1} + \boldsymbol{z}^t \odot \tilde{\boldsymbol{h}}^t. \end{aligned} \tag{2.6}$$

Here $\boldsymbol{z}^t$ serves as the input gate and with $\boldsymbol{1} - \boldsymbol{z}^t$ at the same time as the forget get. The reset gate $\boldsymbol{r}^t$ decides which information from $\boldsymbol{h}^{t-1}$ is relevant to compute the state update $\tilde{\boldsymbol{h}}^t$.

For both the LSTMs and GRUs, it is common practice to stack multiple such networks to process one time step. This can be achieved by passing the original input $\boldsymbol{x}_t$ at time step t to the first GRU layer, and then each subsequent GRU layer receives the previous layer's hidden state until the final hidden output is produced. The hidden state recurrence in each layer is maintained, as would be the case for a single layer GRU. As for stacking multiple layers in deep FNNs, this increases the model's capacity and enables it to learn more complex tasks.

2.2.3 Generative Neural Networks

A generative model is a model that learns to generate data that follows a certain distribution $p(\boldsymbol{x})$. This type of model can be implemented as a neural network. Such generative neural networks have been successfully applied to the generation of images [22]–[24], audio [11], or point clouds [25]. Furthermore, in recent years the interest in modeling

motion prediction as a generative problem has grown as well [26]–[28]. In this section, we present the two most popular types of generative neural networks, the Variational Autoencoder (VAE) [29] and the Generative Adversarial Neural Network (GAN) [30].

Variational Autoencoders

The idea for the VAE [29] originates from the goal to find an optimal encoding $\boldsymbol{z}$ for a given sample $\boldsymbol{x}$, i.e., $p(\boldsymbol{z} \mid \boldsymbol{x})$. However, by expanding this term to

$$p(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})}{\int p(\boldsymbol{x}, \boldsymbol{z})\mathrm{d}\boldsymbol{z}}, \tag{2.7}$$

it becomes clear that due to the integral in the denominator this is intractable. Instead of learning $p(\boldsymbol{z} \mid \boldsymbol{x})$ directly, it can be approximated with a variational posterior $q(\boldsymbol{z} \mid \boldsymbol{x})$ by minimizing the Kullback-Leibler divergence (KLD) $D_{\mathrm{KL}}(p(\boldsymbol{z} \mid \boldsymbol{x}) \| q(\boldsymbol{z} \mid \boldsymbol{x})) \geq 0$. By rearranging this expression, we can derive

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log p(\boldsymbol{x} \mid \boldsymbol{z})\right] - D_{\mathrm{KL}}(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})). \tag{2.8}$$

This term states that we can approximate the true target distribution $p(\boldsymbol{x})$ with the RHS lower bound. The distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$ is implemented by the VAE's generator, the variational posterior $q(\boldsymbol{z} \mid \boldsymbol{x})$ by the encoder and $p(\boldsymbol{z})$ is the latent variable prior. Both the encoder and generator are typically neural networks. During training we pass the sample $\boldsymbol{x}$ through the encoder network that parameterizes the normal distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$. By assuming that $p(\boldsymbol{z})$ is standard normal distributed, we can compute the KLD in closed form. Then we draw a sample $\boldsymbol{z}$ from $q(\boldsymbol{z} \mid \boldsymbol{x})$ (making use of the reparameterization trick [29]) and pass it to the generator network. The generator then parameterizes the distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$ – usually also a normal distribution – and the log-likelihood for $\boldsymbol{x}$ is evaluated. Alternatively, the generator directly predicts the estimate $\hat{\boldsymbol{x}}$ and we compute the mean-squared error with $\boldsymbol{x}$. This is equivalent to maximizing $\log p(\boldsymbol{x} \mid \boldsymbol{z})$. To turn the whole term into a minimization problem, it can be simply negated.

To generate new data at inference time, the encoder can be omitted. Instead, we just sample from the standard normal prior $\boldsymbol{z} \sim p(\boldsymbol{z})$ and then pass the noise vector $\boldsymbol{z}$ through the generator to get a sample $\hat{\boldsymbol{x}}$. Furthermore, instead of approximating $p(\boldsymbol{x})$, we can also learn a conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{c})$ by providing $\boldsymbol{c}$ as an additional input to the generator and encoder.

Autoregressive Neural Networks

An alternative to estimating a lower bound on $p(\boldsymbol{x})$ are autoregressive models. An autoregressive model is based on the principle that the target distribution $p(\boldsymbol{x})$ can be expressed as a joint probability distribution over the data points' individual features, i.e., $p(\boldsymbol{x}) = p(x_1, ..., x_n)$. This joint distribution can then, by the chain rule of probability,

be factorized into a product of conditional distributions:

$$p(x_1, ..., x_n) = \prod_i^n p(x_i \mid x_{>i}) \tag{2.9}$$

In this formulation, each conditional distribution over a specific feature x_i depends only on the features before it, which is an autoregressive property [24].

To implement such an autoregressive model, each conditional distribution is represented by a parametric probability density that is parameterized by a recurrent neural network. A recurrent neural network, such as an LSTM, can store information about all inputs that it received in its hidden state and carry it over long sequences. Hence, sequentially providing it with one feature x_i after another, it can regress the variables that parameterize the corresponding distribution, conditioned on all previous input values.

To generate new samples from $p(\boldsymbol{x})$, we only need to sample $\hat{x}_1$, then successively compute each conditional distribution and draw the next $\hat{x}_{i+1}$ until $\hat{\boldsymbol{x}}$ is composed. To obtain the first distribution, we initially either provide the network with placeholder values or learn the parameters of the first distribution $p(x_n)$ directly. Furthermore, following Eq. 2.9, such autoregressive models allow to compute likelihoods for any $\boldsymbol{x}$, also those that were not sampled from the model itself.

Generative Adversarial Neural Networks

Instead of using probabilistic principles, GANs [30] are trained in a game theoretic setup. A GAN consists of two competing networks, the generator and the discriminator. The generator produces samples $\hat{\boldsymbol{x}} = g(\boldsymbol{z})$ by transforming noise vectors $\boldsymbol{z}$. It tries to fool the discriminator into classifying its generated samples as real. The discriminator network $d(\boldsymbol{x})$, tries to distinguish between samples from the generator and a training dataset [15]. The function [15] that quantifies how good both networks perform can be defined as

$$v(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log d(\boldsymbol{x})\right] + \mathbb{E}_{z \sim \mathcal{N}} \left[\log(1 - d(g(\boldsymbol{z}))\right]. \tag{2.10}$$

Here, $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_d$ are the generator and discriminator network parameters, respectively. The first term of the equation evaluates how well the discriminator can classify samples from the training data as true, and the second term evaluates how good it is at classifying samples created by the generator as false. Therefore, to train the discriminator $v(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d)$ must be maximized. On the other hand, the second term must be minimized to train the generator, whereas the first term will be ignored in the optimization as it is constant w.r.t. $\boldsymbol{\theta}_g$. Hence, with the following optimization, we obtain an optimal generator g^*:

$$g^* = \arg\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} v(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d). \tag{2.11}$$

At training time, it is common to first optimize the discriminator for several steps ahead to stabilize the training and improve convergence. After that, the optimization

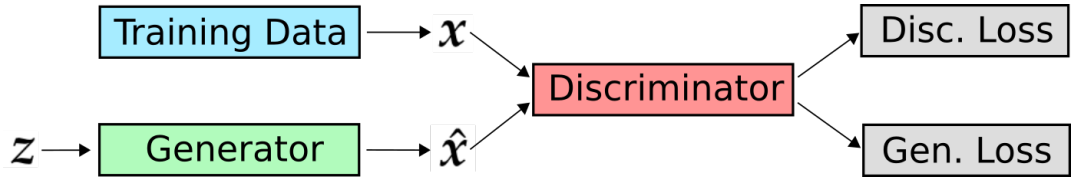


Figure 2.2: The basic architecture of a GAN during training. The generator receives noise and transforms it into a sample from the target distribution, trying to fool the discriminator. Then the discriminator has to differentiate between real samples from the training data and those from the generator.

alternates between the generator and the discriminator at each step. Fig. 2.2 shows how the components in a GAN are connected during training.

At convergence, the generator's samples are indistinguishable from real samples, and the discriminator outputs $1/2$ for each sample [15]. At inference time, the discriminator can be discarded, and only the generator is used. Because GANs are not required to learn the full distribution $p(\boldsymbol{x})$, they can focus on recreating specific examples very accurately. This specialization makes GANs especially suitable to generate sharp images [23] or other complex and highly multimodal data. As for the VAE, in a GAN additional inputs can be passed to the generator and discriminator networks, such that the generator learns to generate samples from a conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{c})$.

2.3 Normalizing Flows

Normalizing flows are a conceptual framework to build machine learning models that can mold a simple probability density into a more complex one by iteratively applying invertible transformations [31]. Normalizing flows can be used to describe how data is distributed, the detection of outliers and anomalies, and the sampling of new data from the learned distribution. In contrast to most other non-trivial generative models, the density $p(\boldsymbol{x})$ of a normalizing flow can be directly evaluated.

In particular, a normalizing flows learns a function that transforms a noise sample $\boldsymbol{u}$ from a simple base distribution $p_u(\boldsymbol{u})$ into sample $\boldsymbol{x}$ from the target distribution:

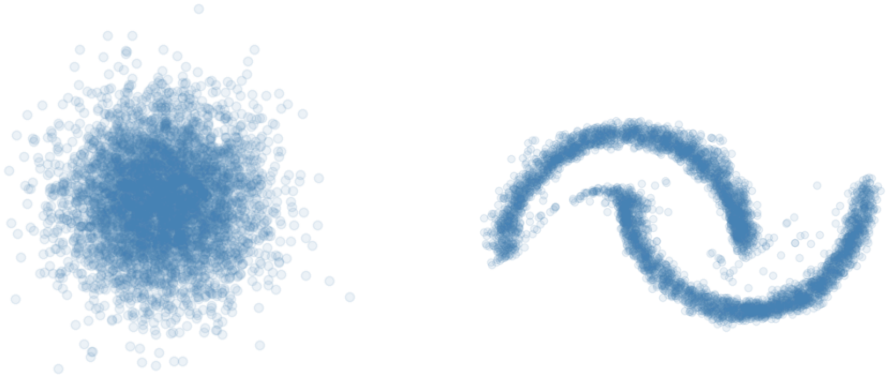
$$\boldsymbol{x} = f(\boldsymbol{u}) \quad \text{where} \quad \boldsymbol{u} \sim p_u(\boldsymbol{u}). \tag{2.12}$$

By defining the transformation $f(\boldsymbol{u})$ such that it is invertible and differentiable, the probability density of $\boldsymbol{x}$ can be obtained by a change of variables [31]:

$$p_x(\boldsymbol{x}) = p_u(\boldsymbol{u}) \left| \det J_f(\boldsymbol{u}) \right|^{-1}. \tag{2.13}$$

In the same manner, by the inverse function theorem, it is also possible to express $p_x(\boldsymbol{x})$ in terms of $\boldsymbol{x}$ and $J_{f^{-1}}$:

$$p_x(\boldsymbol{x}) = p_u(f^{-1}(\boldsymbol{x})) \left| \det J_{f^{-1}}(\boldsymbol{x}) \right|. \tag{2.14}$$



(a) Standard Normal **(b)** MAF

Figure 2.3: This example shows how a Masked Autoregressive Flow (MAF) [32] transforms samples from a standard normal distribution (a) to a multimodal target distribution (b).

The Jacobian determinant describes the extent to which the transformation compresses and expands the density of the base distribution to mold it into the target distribution and scales likelihoods accordingly [31]. Usually, we choose a standard normal distribution for the base distribution, and a neural network implements the invertible transformation. Several such transformations can be composed to make the flow more flexible, i.e.,

$$f(\boldsymbol{u}) = f_2 \circ f_1 \quad \text{with inverse} \quad f^{-1}(\boldsymbol{u}) = f_1^{-1} \circ f_2^{-1}. \tag{2.15}$$

Then the flow's overall Jacobian determinant becomes

$$\det J_{f_2 \circ f_1}(\boldsymbol{u}) = \det J_{f_2}(f_1(\boldsymbol{u})) \cdot \det J_{f_1}(\boldsymbol{u}). \tag{2.16}$$

It is crucial that the Jacobian determinant can be computed efficiently and, depending on the use case, the flow must be easy to invert. For example, a flow that is mainly used to learn a density and to evaluate $p(\boldsymbol{x})$, e.g., for outlier detection, only needs a fast inverse transformation from $\boldsymbol{x}$ to $\boldsymbol{u}$ and can be trained with maximum likelihood estimation by evaluating Eq. 2.14. At inference time, the same flow direction is used. In contrast, for a flow trained as a generative model, both directions must be fast. Like in the previous example, it is trained with Eq. 2.14, but at inference time, new samples are generated by sampling $\boldsymbol{u}$ and passing it through Eq. 2.12. Besides the mentioned properties, a normalizing flow must be expressive to represent complex distributions.

Fig. 2.3 shows an example of how a normalizing flow can transform samples from a two-dimensional standard normal distribution into a more complex target distribution. Even though the input distribution is unimodal, the flow learned an invertible function that maps the input samples to the multimodal half-moon distribution.

2.4 Sensor Coordinate Transformations

To predict motion in a distributed sensor system, we need tracks of objects that pass through the system. This tracking requires the fusion of sensor detections, for which relevant detections must be available in a common coordinate system. We denote this world coordinate system as $\boldsymbol{W}$.

However, each sensor detects objects in its own reference frame. In particular, at a specific time step, a vehicle or other object with true position $\boldsymbol{x}_w$ defined in $\boldsymbol{W}$ that is passing through the system generates detections $\hat{\boldsymbol{x}}_a, ..., \hat{\boldsymbol{x}}_z$ for each observing sensor $a, ..., z$. Here, each $\hat{\boldsymbol{x}}_a$ is defined in the detecting sensor's reference frame $\boldsymbol{S}_a$, which itself is defined relative to the world coordinate system $\boldsymbol{W}$. By computing $\boldsymbol{S}_a\hat{\boldsymbol{x}}_a$, we can transform the detection $\hat{\boldsymbol{x}}_a$ of sensor a in the global reference frame $\boldsymbol{W}$.

Typically, the local sensor coordinate systems $\boldsymbol{S}$ are not known while building a distributed sensor system. Determining these matrices correctly is referred to as extrinsic sensor calibration. In particular, given ground truth, the extrinsic calibration matrix for a single sensor can be determined by solving the following optimization problem:

$$\boldsymbol{S}^* = \arg\min_{\boldsymbol{S}} \frac{1}{n} \sum_{i}^{n} (\boldsymbol{x}_w^i - \boldsymbol{S}\hat{\boldsymbol{x}}^i)^2. \tag{2.17}$$

The optimal solution to this problem is the matrix $\boldsymbol{S}^*$, that transforms the sensor's detection into the global coordinate system, such that it matches the ground truth position $\boldsymbol{x}_w^i$ of the object. Furthermore, the resulting matrix $\boldsymbol{S}^*$ describes the pose of the corresponding sensor in the scene.

Alternatively, when the pose $\boldsymbol{S}_a$ of one sensor is known, it is possible to calibrate a second sensor to it by solving

$$\boldsymbol{S}_b^* = \arg\min_{\boldsymbol{S}_b} \frac{1}{n} \sum_{i}^{n} (\boldsymbol{S}_a\hat{\boldsymbol{x}}_a^i - \boldsymbol{S}_b\hat{\boldsymbol{x}}_b^i)^2. \tag{2.18}$$

This pair-wise calibration approach can be applied successively between all sensors in the system to obtain all required transformations and the overall system calibration.

Besides an extrinsic calibration, many sensor types also require an intrinsic calibration specific to the sensor's physical measurement principle. For example, the intrinsic calibration matrix of a camera computes the projection of a point in the camera's extrinsic coordinate system onto the image plane. This intrinsic calibration considers camera parameters such as focal length, center offset, and pixel skew. However, intrinsic calibration can usually be performed before sensor deployment and with standardized methods, unlike extrinsic calibration.

Chapter 3

Robustness and Generalization in Deep Motion Prediction

To develop a good-performing and reliable prediction model applicable in safety-critical scenarios, we need to understand its limitations. One of the most critical aspects of reliability in machine learning is the model's robustness to input variations and its capability to generalize to data not seen during training. Therefore, in this chapter, we analyze what inhibits the robustness and generalization of neural networks for motion prediction and explore how we can overcome these limitations.

3.1 Motivation

Accurate motion prediction is an essential capability for Intelligent Infrastructure Systems (IISs), autonomous vehicles, and robots to operate safely and to not endanger humans. In recent years, many models based on neural networks were proposed to address this problem [26], [34]–[37]. Neural networks are powerful function approximators and believed to take into account long motion histories and to learn how agents interact.

In this chapter, we show that – surprisingly – a simple Constant Velocity Model (CVM) can achieve state-of-the-art performance on this problem. We demonstrate this with an extensive evaluation using two popular benchmark datasets and compare the CVM with multiple baseline and four state-of-the-art prediction networks. Because the CVM does not require any information besides the agent's last relative motion and performs well across multiple different scenarios, its success raises questions about the generalization of neural networks and how well they utilize the additional information they receive.

For this reason, we analyze what affects their generalization, how neural networks process their inputs, and how these inputs impact their predictions. Our analysis reveals pitfalls in training neural networks and clarifies false assumptions about motion prediction itself. In particular, we analyze:

- **Environmental Priors**. Physical constraints and environmental semantics bias motion towards certain patterns. We show that neural networks implicitly learn such a prior, even though no environment information was explicitly provided. This

Parts of this chapter have been previously published in [33].

learned environmental prior has a strong negative impact on their generalization to new scenes.

- **Motion History**. It is a common belief that taking longer motion histories into account leads to more accurate future predictions. However, our analysis demonstrates that most of this information is redundant and consequently ignored by neural networks. Depriving a neural network of the long past does not lead to prediction degradation.

- **Pedestrian Interactions**. Interactions between traffic participants happen. However, our experiments show that interactions of pedestrians are too complex to predict for neural networks and, in most cases, do not have a significant influence on their trajectories. Furthermore, providing neural networks only with motion histories of surrounding pedestrians can negatively influence their performance.

The datasets we use in our experiments in this chapter focus on pedestrians because their trajectories are highly non-linear and less structured than those of vehicles that move in constrained environments. However, because IISs also observe other traffic participants, e.g., cars, trucks, and motorbikes, we will discuss the implications of this chapter's results for the prediction of vehicle trajectories in Sec. 3.4.4.

3.2 Related Work

The prediction of pedestrian motion has been addressed from various perspectives. In tracking algorithms, motion prediction is vital for robust statistical filtering. To track people in images, Pellegrini et al. propose Linear Trajectory Avoidance models for short-term predictions [38], [39]. Baxter et al. [40] extend a Kalman Filter with an instantaneous prior belief about where people will move, based on where they are currently looking at. Kooij et al. [41] describe the motion of vulnerable road users with a Dynamic Bayesian Network. Ess et al. [42], and Leigh et al. [43] use a Constant Velocity Model in combination with a Kalman Filter in their tracking approaches. Moreover, models that take into account grouping behavior have been explored for prediction [44], [45].

While tracking requires good short-term predictions, we focus on long-term predictions in this work. For long-term predictions, Becker et al. [46] use a recurrent encoder with a multilayer perceptron and achieve good results. Other contributions also take pedestrian interactions into account. Helbing et al. [47] propose the use of attractive and repulsive social forces to model these interactions. This approach was then extended and transferred to the prediction of pedestrians for autonomous robots [48]. Later, interaction-awareness was integrated into neural networks. Alahi et al. [36] train Long Short-term Memorys (LSTMs) [20] for pedestrian motion prediction and share information about the pedestrians through a social pooling mechanism. This mechanism gathers the hidden states of nearby pedestrians with a pooling grid. Xu et al. [49] propose the Crowd Interaction Deep Neural Network that computes the spatial affinity between

pedestrians' last locations to weight the motion features of all pedestrians for location displacement prediction. Vemula et al. [37] address interaction-aware motion prediction with their Social Attention model by capturing the relative importance between pedestrians with spatio-temporal graphs. Zhang et al. [34] propose a state refinement module for LSTM networks that iteratively refines interaction-aware predictions for all pedestrians through a message passing mechanism. For interaction-aware predictions between heterogeneous agents, Ma et al. [50] propose a graph-based LSTM that uses instance layers to take into account individual agents' movements and interactions, and category layers to exploit similarities between agents of the same type. To model distributions of future trajectories, generative neural networks have been used for interaction-aware pedestrian motion prediction. Sadeghian et al. [35] use a Generative Adversarial Neural Network (GAN) [30] that leverages the pedestrian's path history and scene images as context with a physical and social attention mechanism. Gupta et al. [26] propose the Social GAN with an extended social pooling mechanism that is not restricted to a limited grid around the pedestrian to predict as in [36]. Amirian et al. [51] use an InfoGAN for pedestrian motion prediction to avoid mode collapse.

Besides interaction-awareness, also the environment has been exploited for pedestrian motion prediction. Ballan et al. [52] extract navigation maps from birds-eye images and transfer them to new scenes with a retrieval and patch matching procedure to make predictions. Jaipuria et al. [53] propose a transferable framework for predicting the motion of pedestrians on street intersections, based on Augmented Semi-Nonnegative Sparse Coding. Lee et al. [27] predict the motion of vehicles and pedestrians by sampling future trajectory hypotheses from a Conditional Variational Autoencoder (VAE) [29]. They select the most reasonable trajectories by scoring them based on future interactions and consider the environment by encoding an occupancy grid map with a Convolutional Neural Network (CNN). Bartoli et al. [54] consider environmental context by providing an LSTM with distances of the target pedestrian to static objects in space, as well as a human-to-human context in the form of a grid map, or the neighbors' hidden encodings. Pfeiffer et al. [55] propose an LSTM that receives static obstacles as an occupancy grid and surrounding pedestrians as an angular grid. Aside from neural networks, also set-based methods [56], Gaussian Processes [57] and Reinforcement Learning algorithms [58] have been used for predictions that take into account the environment.

In contrast to the related work, we do not develop a new prediction architecture in this chapter. Instead, we show that the simple CVM can achieve competitive performance on popular long-term motion prediction benchmarks. Furthermore, we form hypotheses that could explain this observation and confirm these hypotheses in an extensive analysis.

3.3 Performance of the Constant Velocity Model

In this section, we first formally define the CVM and describe our experimental setup. Then we evaluate the performance of the CVM on two popular benchmark datasets and

(a) Hotel **(b)** UCY-Uni **(c)** Zara1

Figure 3.1: Different scenes from the ETH and UCY datasets. To generate trajectories, the authors annotated and tracked each pedestrian in the corresponding videos.

compare it to several baseline and state-of-the-art neural prediction models. Finally, we discuss both the quantitative and qualitative results of our experiments.

3.3.1 CVM Definition

Based on the assumption that the most recent relative motion $\boldsymbol{p}_i' = \boldsymbol{p}_i^t - \boldsymbol{p}_i^{t-1}$ of a pedestrian is the most relevant predictor for his or her future trajectory, the CVM is a simple but effective prediction method. It assumes that the pedestrian will continue to walk with the same velocity and direction as observed from the latest two time steps. Because it does not use any filtering, it is sensitive to measurement noise. In particular, it predicts the relative future trajectory for pedestrian i as

$$\hat{\boldsymbol{x}}_i' = (\boldsymbol{p}_i', ..., \boldsymbol{p}_i'), \tag{3.1}$$

where the number of $\boldsymbol{p}_i'$ is equal to the number of prediction steps.

3.3.2 Experiments

To evaluate the performance of the CVM and to compare it with other models, we use two datasets for pedestrian motion prediction, the ETH dataset [38] and the UCY dataset [59]. In total, they contain 1950 unique pedestrians and five scenes (see Fig. 3.1): ETH-Uni and Hotel (from ETH) and Zara1, Zara2, and UCY-Uni (from UCY). Each scene has a different walkway layout and contains non-traversable obstacles. The datasets are based on real-world video recordings from different scenarios, like university campuses and pedestrian walking zones. Both datasets have been heavily used to evaluate the performance of motion prediction models in recent contributions [26], [34]–[37], [46], [49], [51].

In our experiment, we observe the last 8 positions of a pedestrian's trajectory and predict the next 12 time steps. This corresponds to an observation window of 3.2 s and a prediction for the next 4.8 s, which is an established setting and used in other motion prediction papers as well [26], [34]–[37], [46], [51]. This means we slice trajectories with

a sliding window and step-size of one into sequences of length 20. We reject sliced trajectories with a length shorter than 10. By this, we guarantee an observation of 8 time steps and that the evaluated models must predict at minimum the next two time steps.

As proposed in [36], we train on four scenes and evaluate on the remaining one in a leave-one-out cross-validation fashion. This cross-validation ensures the evaluation of the model's generalization capability to new scenarios. Like in related contributions, we report errors in meters and evaluate all models with the following metrics:

- *Average Displacement Error (ADE)* — Average l^2-distance between all corresponding positions in ground truth and predicted trajectory.
- *Final Displacement Error (FDE)* — l^2-distance between the last position in ground truth and the last position in predicted trajectory.

3.3.3 Training

We trained each model – except state-of-the-art models and the CVM that does not require training – with the Adam Optimizer [17] with learning rate 0.0004, batch size 64, and for 35 epochs. All hyperparameters were determined empirically. As loss function we used the Mean Squared Error (MSE) that is defined as

$$\text{MSE}(\theta) = \frac{1}{N}\sum_{i}^{N}\|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|^2, \tag{3.2}$$

where $\boldsymbol{x}_i$ is the ground truth trajectory, $\hat{\boldsymbol{x}}_i$ is the predicted trajectory, and θ are the predicting model's parameters.

We randomly split the training scenes into a training set and a 10% validation set to detect overfitting. All models converged without overfitting and did not require further hyperparameter tuning. State-of-the-art models from other contributions were trained as described in respective papers, including specified data augmentations and loss functions.

3.3.4 Models

We compare the performance of the CVM with multiple baselines that are commonly used in contributions to the pedestrian motion prediction domain:

- *Constant Acceleration* (**CAM**) — Observes the last three positions of a pedestrian and assumes he or she continues to walk with the same acceleration.
- *Linear Regression* (**LIN**) — Multivariate multi-target linear regression model that estimates each component in the predicted trajectory as an independent linear regression. Each predicted variable depends on the full motion history.
- *Feed-forward Neural Network* (**FNN**) — Fully connected neural network that receives all eight motion history time steps as a flattened vector. It then applies two hidden

layers with 60 and 30 neurons, respectively. Both hidden layers are followed by Rectified Linear Unit (ReLU) activations. The final linear output layer has 24 outputs, which corresponds to 12 prediction time steps. We also evaluated bigger networks for this comparison and in the following analysis section. This lead to worse performance (stronger overfitting) in this comparison and the same conclusions in the analysis.

- *LSTM Network* (**LSTM**) — Stacked LSTM that receives single positions and linearly embeds them in a 32-dimensional vector. Three LSTM layers with 128 hidden dimensions and a linear output layer follow. We trained the LSTM with Teacher Forcing [60].

We further include four state-of-the-art models in our evaluation. These are the *RNN-Encoder-MLP* (**RED**) [46], the *LSTM with State Refinement* **SR-LSTM** [34] and for generative models *Social GAN* (**S-GAN**) [26] and the *SoPhie GAN* (**SoPhie**) [35].

Because S-GAN and SoPhie were evaluated by drawing 20 samples and taking the predicted trajectories with minimum ADE (minADE) and minimum FDE (minFDE) for evaluation into account, we add an extended version **CVM-S** of the CVM for comparability. For CVM-S we add additional angular noise to its predicted direction, which we draw from $\mathcal{N}(0, \sigma^2)$ with $\sigma = 25^\circ$ and evaluate its error in the same fashion.

While for SR-LSTM and SoPhie GAN we report the original papers' results, for S-GAN we use a pre-trained[1] version of their best performing model, which is not interaction-aware as reported in [26]. Note that this is an improved version compared to what the authors report in their paper. To evaluate RED, we carefully re-implemented and trained the model as proposed by the authors.

3.3.5 Results

In this section, we discuss the results of our prediction experiments. In particular, we first discuss the prediction errors of the compared models in a quantitative fashion and then compare qualitative prediction examples.

Quantitative. In Tab. 3.1 we show the prediction errors for all evaluated models on each scene. On average, the best performing model for both minADE and minFDE was CVM-S, which is the CVM with sampling. It also outperformed state-of-the-art generative models S-GAN and SoPhie. As explained in the previous section, CVM-S, S-GAN and SoPhie were evaluated by considering only the errors of the best predicted samples. For this reason, the other models are discussed separately.

Of the models without sampling, the CVM outperformed all other models as well, including state-of-the-art RED and SR-LSTM. Its advantage was especially strong for the Hotel scene. Among the basic neural networks, Feed-Forward Neural Network (FNN) outperformed LSTM. We hypothesize that an error accumulation effect for the LSTM

[1] github.com/agrimgupta92/sgan

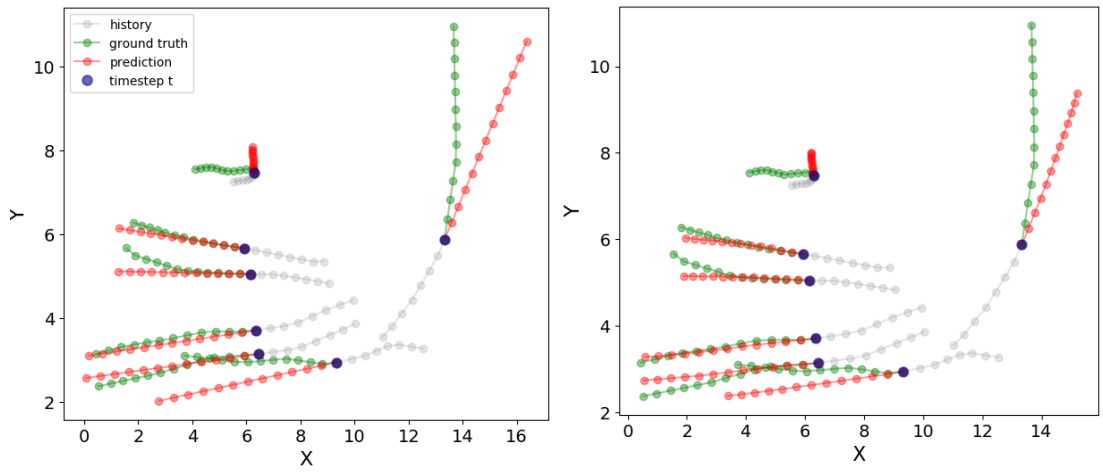
Model	ETH-Uni	Hotel	UCY-Uni	Zara1	Zara2	AVG
CAM	1.35 / 3.29	0.95 / 2.41	0.79 / 2.03	0.59 / 1.50	0.50 / 1.30	0.84 / 2.11
LIN	0.58 / 1.11	0.39 / 0.81	0.60 / 1.19	0.44 / 0.93	0.41 / 0.83	0.48 / 0.97
FNN	0.67 / 1.32	1.59 / 3.12	0.69 / 1.38	0.39 / 0.81	0.38 / 0.77	0.74 / 1.48
LSTM	0.54 / 1.04	2.91 / 6.07	0.72 / 1.60	0.34 / 0.74	0.43 / 0.95	0.99 / 2.08
RED	0.60 / 1.14	0.47 / 0.94	0.47 / 1.00	0.34 / 0.76	0.31 / 0.70	0.44 / 0.91
SR-LSTM	0.63 / 1.25	0.37 / 0.74	0.51 / 1.10	0.41 / 0.90	0.32 / 0.70	0.45 / 0.94
CVM	0.58 / 1.15	0.27 / 0.51	0.46 / 1.02	0.34 / 0.76	0.31 / 0.69	0.39 / 0.83
SoPhie	0.70 / 1.43	0.76 / 1.67	0.54 / 1.24	0.30 / 0.63	0.38 / 0.78	0.54 / 1.15
S-GAN	0.59 / 1.04	0.38 / 0.79	**0.26** / **0.49**	**0.18** / **0.32**	**0.18** / **0.34**	0.32 / 0.60
CVM-S	**0.43** / **0.80**	**0.19** / **0.35**	0.34 / 0.71	0.24 / 0.48	0.21 / 0.45	**0.28** / **0.56**

Table 3.1: Displacement errors of the CVM, common baselines and state-of-the-art models. The errors are shown as ADE / FDE for the models that make a single prediction (upper block) and Minimum Average Displacement Error (minADE) / Minimum Final Displacement Error (minFDE) for those that make 20 predictions (lower block).

could be responsible for this, as it predicts the next step based on its output for the last step. The CAM performed the worst, which shows that, especially over long prediction horizons, the assumption of continual acceleration or deceleration is detrimental. To our surprise, LIN performed better than LSTM, contrary to what [36] and [26] reported. We believe this discrepancy can be attributed to the data augmentation they used for all their models. The good performance of LIN can be explained with the high bias and hence strong generalization of linear regression models.

Qualitative. Fig. 3.2 shows single predictions of the CVM and RED for the same UCY Uni scene. The worst prediction CVM made in Fig. 3.2a is for the pedestrian at the top left, who suddenly makes a sharp turn. This behavior is not predictable based on a pedestrian's motion history. The rightmost pedestrian walks in a gradual curve. As the CVM makes linear predictions, it can not extrapolate this behavior. However, often these trends are not reliable, and the trajectory curvature suddenly changes. For example, the bottommost pedestrian is first taking a turn to the left but then changes his or her trajectory back to the right, which is difficult to foresee. Overall, the linear predictions of the CVM are good approximations of the pedestrians' future trajectories. This is further confirmed by the fact that the second-best single prediction model RED (Fig. 3.2b) learned to predict trajectories that are mostly linear as well.

In Fig. 3.3 we show predictions of the CVM-S and the second-best multiple trajectory prediction model S-GAN for the same scene. For both models we sampled 20 trajectories as described in Sec. 3.3.4. The width of the prediction cone of the CVM-S in Fig. 3.3a can be controlled with σ during sampling. CVM-S can predict accurate linear approximations



(a) CVM **(b)** RED

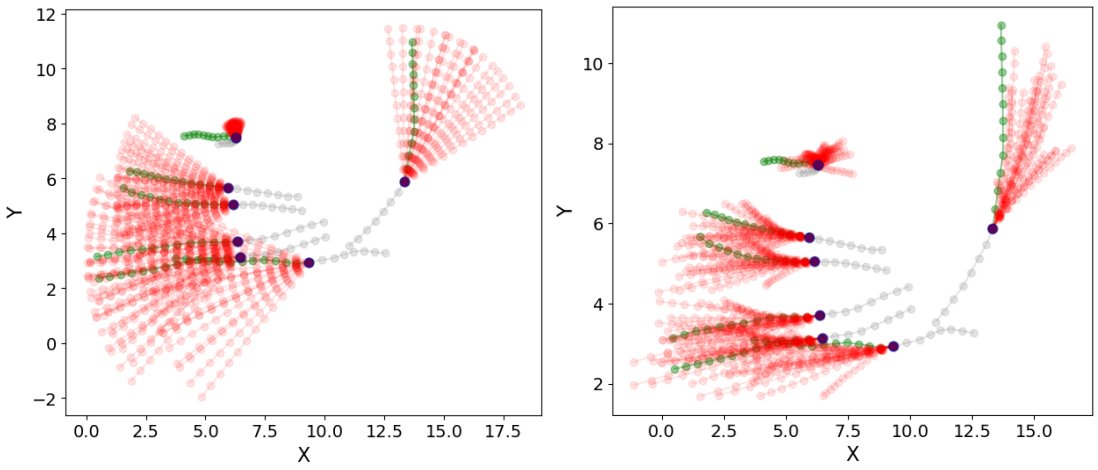
Figure 3.2: Single trajectory predictions of the CVM (a) and the second best model RED (b). Similar to the predictions of CVM, RED learned to predict approximately linear trajectories, which indicates that this is a robust strategy.

of the pedestrians' future trajectories, even for those pedestrians that walk in a curved fashion. The predictions of S-GAN in Fig. 3.3b appear to be clustered into maneuvers, like walking straight, turning right, or left. However, the pedestrians do not always seem to follow these patterns.

In summary, the low errors of our CVM variants indicate that – on average – people tend to walk at constant speeds. Furthermore, sudden turns are difficult to predict, which explains the good performance of highly diverse predictions, for example, the random sampling in CVM-S and the S-GAN that was trained with the variety loss.

3.4 Generalization and Robustness Analysis

Our experiments show that the CVM performs strongly on pedestrian motion prediction, despite its simplicity and lack of additional information besides motion histories. Its performance indicates that the neural networks it outperforms are either unable to use the additional information they are provided with to their advantage or that this information is not as relevant as commonly believed. In this section, we develop explanations for this result. In particular, we analyze the influence of environmental priors, the motion history, and pedestrian interactions on neural network performance. For our analysis we use the FNN and the RED [46] from Sec. 3.3.4. Both models have simple training dynamics, i.e., they converge reliably. This simplicity enables us to disentangle the effects of our modifications on the models' performance that we observe in our experiments from noise caused by instabilities and randomness in the training process. While FNN is a basic neural network, RED is a state-of-the-art model with



(a) CVM-S **(b)** S-GAN

Figure 3.3: The same scene with multiple trajectory predictions of CVM-S (a) and the second best model S-GAN (b). While the CVM-S predicts a random trajectory cone, S-GAN seems to predict trajectories clustered along maneuvers.

good performance.

3.4.1 Learning Environmental Priors

The environment of each scene puts constraints and bias on how pedestrians can move within it. Constraints can be physical, for example, certain areas like closed buildings cannot be traversed. Bias can be caused by the semantics of a scene, e.g., in a parking lot of a shopping center, people are likely to either walk towards the shopping center or away from it.

We argue that even though this environmental information was not explicitly provided to the networks in Sec. 3.3.2, they implicitly learn such prior. Each area of a scene corresponds to a specific numeric range of input coordinates. This allows a neural network to associate these areas with certain motion patterns. But even if we prevent the network from making this associations, it would still learn motion patterns that are typical for the whole scene.

To verify this hypothesis, we compare the models FNN and RED from Sec. 3.3.4 that we train without data augmentations with two modifications that dampen the effects of learning environmental priors. For the first modification (**Relative**) we do not feed the network with absolute positions as inputs, but instead with the pedestrian's past relative motion, which we compute analog to $\boldsymbol{x}_i'$ in Sec. 3.3.1. This ensures that the model can not learn to associate certain areas in the scene with a specific motion pattern. For the second modification (**Rotations**), we use relative motion and additionally add random rotations to the training trajectories to reduce directional bias. For this purpose we sample angles from $\mathcal{N}(0, \sigma^2)$ with $\sigma = 180°$. Note that we only apply one rotation to

Model	Metric	Basic	Relative	Rotations
FNN	**ADE Hotel**	1.59	0.45	**0.30**
	FDE Hotel	3.12	0.95	**0.55**
	ADE Avg	0.74	0.44	**0.42**
	FDE Avg	1.48	0.93	**0.87**
RED	**ADE Hotel**	1.59	0.45	**0.30**
	FDE Hotel	2.80	0.92	**0.56**
	ADE Avg	0.79	0.44	**0.41**
	FDE Avg	1.49	0.92	**0.86**

Table 3.2: Effects of learning environmental priors. Our proposed augmentations significantly increase the models' generalization and hence performance.

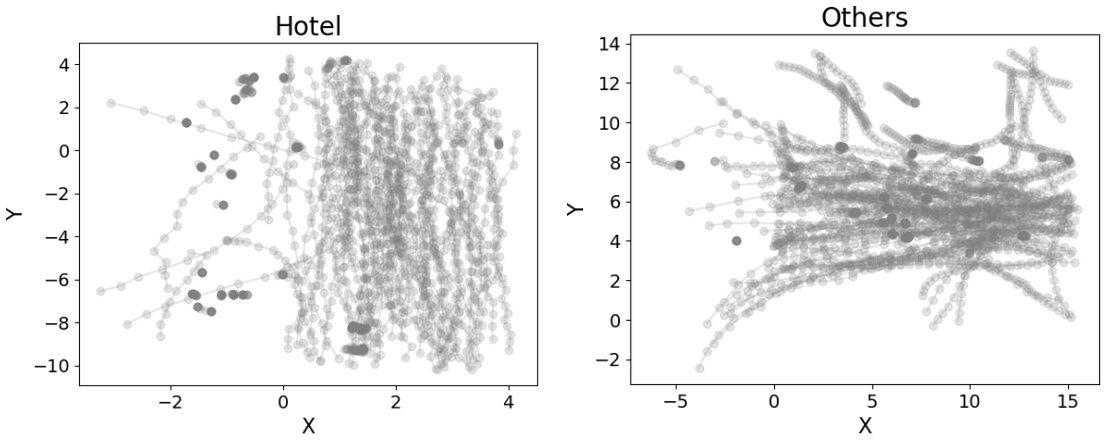
each sample in the training dataset, such that the resulting dataset has the same size for all three training variations. This ensures a fair comparison.

The ADE and FDE for each variant are displayed in Tab. 3.2. For the rows with **Avg** we report the average errors of the leave-one-out experiment as in Sec. 3.3.2. Our results show that Relative, as well as the additional Rotations strongly improved the models' performance. This effect is even intensified for scene Hotel, which we therefore report additionally. To understand why Hotel was stronger influenced by learning environmental priors, we plotted a uniform sub-sample of the trajectories of the Hotel scene, as well as of all other scenes combined in Fig. 3.4. It shows that most trajectories in the other scenes run horizontally, whereas in Hotel most trajectories run vertically. This explains our observation, and we conclude that the networks learned this motion pattern as a prior which negatively influences their generalization capability.

While the benefits of data augmentation have been reported earlier [35], [46], our analysis shows that the reason for this is that it helps to prevent learning environmental priors. The awareness of this problem is also necessary to understand certain phenomena, for example, we hypothesize that learning environmental priors is the primary reason for the bad performance of LSTMs reported in [37], instead of the missing interaction-awareness as the authors suggest. To train the LSTM, the authors used absolute positions and applied no data augmentation.

3.4.2 Long-term Motion Histories

It is believed that neural networks can use long motion histories to make better predictions. In Sec. 3.3.2 we provided our models with a history of eight time steps, which is common practice in the domain of pedestrian motion prediction [26], [34]–[37], [46], [51]. However, the performance of the CVM that only uses the last two time steps to



(a) Hotel **(b)** Other scenes

Figure 3.4: Trajectories from scene Hotel and the other scenes combined. We uniformly subsampled the datasets for better visibility. The majority of trajectories in Hotel are oriented vertically, whereas those in the other scenes mostly run horizontally.

make predictions suggests that for pedestrian motion prediction, long histories are not as relevant as believed. To isolate the effects of learning environmental priors and the motion history, we use the data augmentations from Sec. 3.4.1 in this experiment.

To evaluate if the models utilize the full motion history to make predictions, we compute gradient norms for each time step with respect to the predicted trajectory. In particular, after training, we keep the network static and summarize predicted trajectories in a scalar value by summing up the absolute values of the network's predicted displacements. As our network $f(\boldsymbol{o}_i)$ is a function of the agent's motion history, we can compute gradients

$$\nabla_i^t f = \left[\frac{\partial f(\boldsymbol{o}_i)}{\partial x_i^t}, \frac{\partial f(\boldsymbol{o}_i)}{\partial y_i^t} \right] \tag{3.3}$$

for each time step t in history $\boldsymbol{o}_i$. Then we compute the norm $\|\nabla_i^t f\|$ of each gradient and evaluate it for all test set trajectories with respectively trained models. We sum the gradient norms for all trajectories per time step and normalize the resulting values to a distribution, such that they sum up to one. This distribution expresses how much influence each time step in motion history $\boldsymbol{o}_i$ has on average on the output trajectory.

We found that for model FNN the latest relative motion has an influence of 68.2% on the predicted trajectory, while time step $t-1$ contributes only 8.1%. The other five time steps in the relative motion history influenced the prediction with 3.8% – 6%, and their influence did not decrease monotonically but fluctuated. These fluctuations contradict the intuition, that all time steps contain predictive information with decreasing relevance. For model RED, the influence of the latest time step was 80.3% and time step $t-1$ only 8.3%, and thus the influence drop is even stronger. This stronger drop is likely because

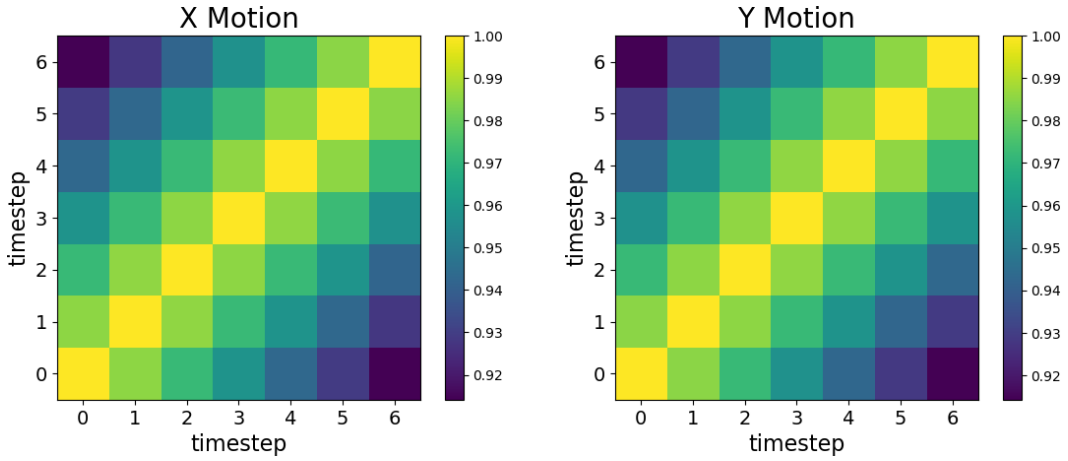


Figure 3.5: Pearson correlation coefficients between all time steps of the observed relative motion histories for the X and Y dimension. All time steps are highly correlated.

of the recurrent encoder of RED. In summary, the non-monotonic fluctuations of earlier time steps' influences, combined with the substantial drop from time step t to $t-1$ indicates that distant time steps in the history are not predictive.

We further computed linear correlation coefficients for the X and Y dimensions of all time steps in the relative motion history. The resulting correlations are displayed in Fig. 3.5. All time steps are highly correlated, with correlation coefficients ranging from 0.91 to 1.0. The closer the time steps are, the more they are correlated. While some correlation of positions in the motion history is expected, this high correlation means that the time steps contain mostly shared and thus redundant information. A high correlation of features is an undesirable property in regression problems such as motion prediction. For the neural networks we evaluated, this redundant information likely acts as noise rather than signal. This explains why the networks strongly rely on the latest time step for their predictions.

To empirically confirm our findings, we analyzed if the performance of the neural networks deteriorates by successively depriving them more of the pedestrian's motion history. In particular, we train the networks with relative histories of sizes $[7, ..., 1]$, but let them always predict the next 12 time steps as usual.

Tab. 3.3 shows the networks' performance when providing them with the entire motion history and a history of size one. We further evaluated all intermediate history sizes, and the performance of the models stayed approximately the same for all settings. There was no monotonic decrease in performance during history deprivation, but the performance fluctuated. We report the sample standard deviations of these fluctuations in Tab. 3.3 as well. These small fluctuations can be attributed to randomness in the network's training process, i.e., weight initialization and stochastic gradient descent. Our findings confirm that, contrary to popular belief, a long motion history is not more predictive than a short history for pedestrian motion prediction.

Model	Metric	Full History	History Size One	σ
FNN	**ADE Avg**	0.42	**0.41**	0.002
	FDE Avg	0.87	**0.86**	0.005
RED	**ADE Avg**	0.41	**0.40**	0.003
	FDE Avg	0.86	**0.84**	0.005

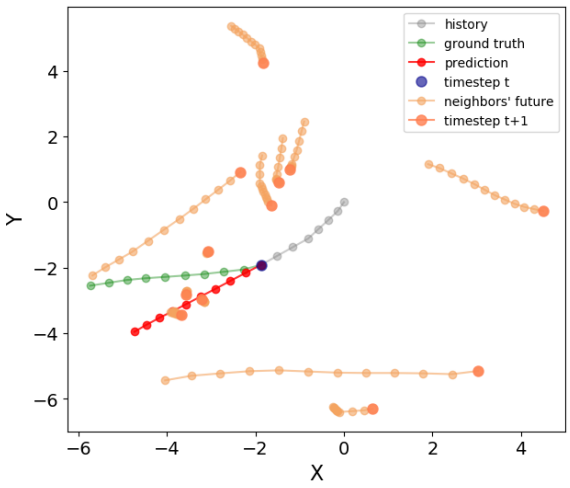
Table 3.3: Depriving the networks of motion history. No negative impact on the networks' predictive power can be observed.

3.4.3 Utilizing Interactions

To behave in an interaction-aware manner, an agent must anticipate the future motion of its neighbors. Only then is it possible to plan his or her trajectory to avoid potential collisions. This implies that a neural network that predicts an interaction-aware trajectory for a pedestrian i must implicitly and simultaneously predict the future trajectories of the pedestrian's neighbors. The CVM does not receive information about surrounding pedestrians and is not making interaction-aware predictions. Its strong performance hints that interactions do not strongly influence the pedestrians' trajectories, or the state-space of possible interactions is too big and complex for neural networks to make robust predictions that reliably decrease the expected error.

To analyze this, we compare three variations of training our models FNN and RED. For the experiments in this section, we modify RED and provide its decoder with additional position information about the target pedestrian's neighbors by concatenation to the target encoding. The FNN receives the additional positions directly with its input. We again apply the data augmentations from Sec. 3.4.1 for both models.

In the first version (**Basic**), the models do not receive any neighborhood information. In the second version (**History**), the models receive the last eight history steps – including time step t – of the pedestrian's 12 closest neighbors. Note that also state-of-the-art models like [26], [36] or [37] receive information about the neighbors' past, but indirectly through specialized pooling modules. In the third version (**Future**), the models are provided with 12 true future positions of the pedestrian's 12 closest neighbors, starting from time step $t+1$. Hence, in this version, the models have perfect information about the neighbors' future trajectories and should utilize it if this information is relevant for making predictions. We choose 12 as the number of observed neighbors because the average number of neighbors across all scenes is 11.34 and provide all neighbor positions relative to the position $\boldsymbol{p}_i^t$ of the target pedestrian i. We order the observed neighbors by distance to pedestrian i at time step t in ascending order and pad missing neighbors with zeros. We do not include neighbors with partial trajectories in the observed time window, as this harms the prediction performance of History and Future. We re-train each model variant separately as described in Sec. 3.4.2.



©IEEE 2020 [61].

Figure 3.6: Prediction of the model FNN that received the neighbors' future trajectories as input. Despite this additional information it predicted a colliding trajectory.

Model	Metric	Basic	History	Future
FNN	**ADE Avg**	**0.42**	0.47	0.44
	FDE Avg	0.87	0.93	**0.86**
RED	**ADE Avg**	**0.41**	0.45	0.44
	FDE Avg	**0.86**	0.93	0.90

©IEEE 2020 [61].

Table 3.4: Influence of neighborhood information on predictions. Even ground truth knowledge about the future does not provide a significant advantage.

Tab. 3.4 shows the results of our experiments for all three training variants. Both models performed the worst when they received information about the neighbors' histories. This is likely because pedestrian interactions are too complex, and the model cannot determine robust solutions while internally predicting for all pedestrians simultaneously. Instead of being useful, the additional information about the neighbors' histories acts as noise. Surprisingly, even the models we provide with the neighbors' future trajectories do not outperform those with no neighborhood information. They are relatively on par, and for RED the version Basic performs even slightly better. These results explain why the CVM can perform well without taking into account any information about the pedestrian's neighbors. The fact that none of the models can exploit the neighbors' actual future trajectories shows that interactions between pedestrians are either not predictable or that, on average, the impact of interactions on the pedestrians' trajectories and thus the prediction error is small.

Fig. 3.6 shows a failed prediction of model FNN that was trained with Future. The

model predicted a path that would cause a collision with the standing neighbors despite fully observing them. Besides such failures, we also observed predictions that could be interpreted as interaction-aware, but these were so infrequent that rather chance and not interaction-awareness caused them. Furthermore, most true trajectories do not involve obvious interaction-aware behavior, which makes observing interaction-awareness additionally difficult.

Our analysis indicates that interactions between pedestrians are less relevant than commonly believed for making accurate predictions. Furthermore, interaction-aware predictions are likely too complex to solve only based on neighbors' motion histories. Our results are consistent with the observations made by [26] that including interaction-awareness by providing the model with the neighbors' motion histories does not lead to performance gains and can even be detrimental.

3.4.4 Implications for Vehicle Trajectory Prediction

Because the most popular state-of-the-art models for motion prediction are evaluated and compared with pedestrian datasets, we used these datasets in this chapter for our experiments and analysis. However, the results of our analysis allow us to draw conclusions for the prediction of other traffic participants, especially vehicles. In this section, we want to draw these conclusions and argue their validity.

Environmental Priors. As we have shown in Sec. 3.4.1, neural networks trained for pedestrian motion prediction learn environmental priors that reflect the environments of the scenes used for training. This implies that neural networks – even when trained only on motion histories – do not automatically learn a generic motion model for the agent type they were trained for.

We can also explain this behavior from a probabilistic perspective. A dataset that contains trajectories can be interpreted as a set of samples drawn from a random variable that follows the true trajectory distribution. This distribution is heavily shaped by the environments of the underlying scenes. A simple example for this would be a dataset that only contains trajectories recorded on a left curved road, where pedestrians walk beside the road and vehicles drive on it. Each agent in such a scene would move on a curved trajectory. Hence, the distribution would not contain straight trajectories, sharp turns, or direction changes. A probabilistic model that learns such a distribution will never predict anything but curved trajectories.

Some neural networks were specifically designed to learn distributions, e.g., GANs or VAEs. However, even models that only make a single prediction are usually trained with distance measures such as the MSE. When using the MSE, it is trivial to show that the trained model learns to predict the expected value of the underlying distribution. Therefore, training the model like this can be interpreted as fitting a Normal distribution with fixed variance to the data. Consequently, the model will also learn the source distribution and with it described environmental patterns.

This explanation for learning environmental priors is valid for whether a network is trained to predict trajectories for pedestrians, cyclists, or vehicles. In the case of

vehicles, these effects may even be exacerbated as these move in much more constrained and structured environments than pedestrians, which increases the significance of observable patterns. However, because the environments of vehicles are so structured, when explicitly provided with information about it and with sufficient variation in the dataset, the network may be enabled to exploit this information systematically.

Motion History. In our analysis, we showed that long motion histories are not more useful than short ones to make accurate predictions about a person's future motion. We believe that in the case of vehicles, this holds too. For a car driving towards a crossing, it is unnecessary to know where it came from to predict where it will go. The most critical information in its motion history to predict if it will make a turn is its change in speed and the lane it is on at the time of prediction. Its lane can be readily determined by its current position.

The most relevant parameters of motion are direction, directional change, velocity, acceleration, and jerk. All of these can be computed from just four observed time steps, i.e., in relative coordinates a history size of three. Furthermore, vehicles move in environments that often dictate where they can go in the future and where they came from, which further reduces the relevance of long motion histories for vehicle motion prediction.

Interactions. Interactions between pedestrians are hard to predict, and on average, their influence on the pedestrians' trajectories is most likely rather small. Pedestrians that walk on colliding trajectories do not usually make large circles around each other to avoid a collision but manage this with subtle speed adjustments or by turning their upper body to move forward in dense crowds. However, vehicles drive on constrained roads where typically a lane is not much wider than a vehicle. If one vehicle breaks in close distance to the one following him, the vehicle behind it must either break as well or overtake if possible. Such a breaking maneuver can even propagate over multiple vehicles that drive behind each other. From this example, it becomes clear that interactions between vehicles most likely form much stronger patterns that are easier identifiable and usable to make better predictions.

While we only formulated hypotheses in this section, we demonstrate that vehicles can utilize interactions and environment information in Sec. 4.5.4, and show in Sec. 6.3.2 that also vehicle motion prediction is possible with short histories.

3.5 Conclusion

In this chapter, we have shown with extensive experiments that the CVM can outperform state-of-the-art models in motion prediction. Due to its strong performance on commonly accepted benchmarks and its simplicity, the CVM should in the future be included as a standard baseline for motion prediction. Establishing strong baselines is vital for advancing research, especially given a trend toward complex models [62]. Furthermore, simple methods such as the CVM help to establish a better understanding of the problem at hand. This has been demonstrated in other domains, such as image classification [63]

or captioning [64], as well.

Inspired by the result of our CVM experiments, we analyzed why neural networks can not use their computational power and the additional information they receive to their advantage. We found out that neural networks learn environmental priors that negatively influence generalization and showed how data augmentation helps to alleviate this problem. Furthermore, we demonstrated that a long motion history is not required for making accurate predictions, and it mainly contains redundant information that neural networks ignore. Our analysis also indicates that interactions between pedestrians are less relevant than commonly believed. Based on the neighbors' motion histories, interactions are too complex to predict reliably and likely do not significantly influence their trajectories in most cases. Lastly, we argued via analogies and examples why these insights are also relevant for predicting the motion of other traffic participants. However, for accurately predicting the motion of vehicles interactions may be more relevant and their environment could provide strong constraints for possible future trajectories.

Our results in this chapter suggest that research on motion prediction should set a stronger focus on theoretic and probabilistic fundamentals to develop models that outperform simple baselines such as the CVM. Furthermore, it is necessary to conduct careful ablations to determine which information neural networks utilize and benefit from. Even though humans know that they react to each other while walking, and it seems natural to include interaction information in a pedestrian prediction model, this information does not necessarily translate to better empirical results.

Chapter 4

Probabilistic and Context-Aware Prediction Models

The previous chapter shows that simple baselines can outperform modern neural prediction models, and their generalization is limited when not trained with caution. This concerns not only models that make single predictions but also state-of-the-art generative prediction models, which performed worse than randomly sampling from a Constant Velocity Model (CVM). However, we assume that a model capable of accurately learning the true distribution of future trajectories must make better predictions than a simple baseline. Therefore, in this chapter, we develop a novel context-aware generative prediction model that addresses these limitations, and we propose methods to enhance its robustness. Furthermore, we will show in our experiments that our model is suited to make reliable predictions, as it is required for the use in Intelligent Infrastructure Systems (IISs).

4.1 Motivation

For autonomous agents like vehicles and robots, it is essential to accurately predict the movement of other agents in their vicinity. Only with this information can collisions be avoided and interactions become safe. However, trajectories can never be predicted with absolute certainty, and multiple future outcomes must be taken into account to plan safe actions.

To address this problem, research on generative models for motion prediction has recently attracted attention. An ideal generative model is expressive and can learn the true underlying trajectory distribution it is trained for. Furthermore, it allows the assignment of a likelihood value to each prediction. Knowing how likely specific trajectories are is necessary to prioritize because it is infeasible for an agent to take into account all possible future behaviors of surrounding agents. Mainly because the space of future trajectories is continuous and the number of possibilities is infinite.

However, most methods do not have all of these desirable properties. For example, Generative Adversarial Neural Networks (GANs) have been used extensively for motion

Parts of this chapter have been previously published in [65].

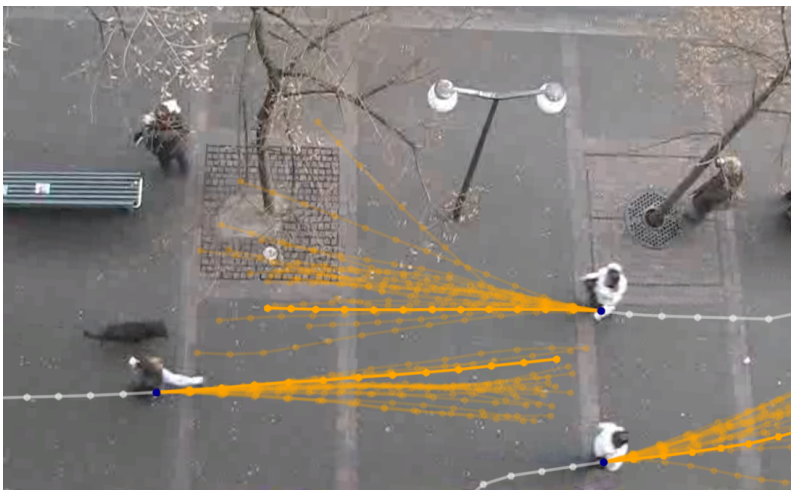


Figure 4.1: Trajectory predictions of our model (orange). More likely trajectories are drawn more opaque. The distributions our model learned are highly multimodal.

prediction [26], [35], [51], but suffer from mode collapse and are not guaranteed to learn the true distribution of the data [66], [67]. The Variational Autoencoder (VAE) is a popular type of generative model as well [27], [68]–[70] and approximates the true distribution with a lower bound. Unfortunately, likelihoods cannot be calculated directly with VAEs and must be estimated with computationally expensive Monte Carlo methods. Other contributions try to overcome the problem of missing likelihoods with the use of parametric density functions, most commonly normal distributions [71], [72]. This often requires unrealistic independence assumptions and provides only limited expressive power.

This chapter proposes a novel motion prediction model that addresses the above-mentioned issues. In particular, our model FloMo is based on normalizing flows that we condition on observed motion histories, and encoded interaction and environment information. It is expressive and able to learn complex multimodal distributions over future trajectories (see Fig. 4.1). With FloMo, trajectories can be efficiently sampled, and likelihoods are computed in closed form. These tractable likelihoods allow us to train our model with Maximum Likelihood Estimation (MLE) instead of using an inferior proxy loss. Because training normalizing flow with trajectory data is prone to likelihood divergence, we apply a novel noise injection method that significantly stabilizes training and enables the use of our model's likelihoods in downstream tasks. Furthermore, we propose a new data augmentation that helps our model generalize better to unknown data and improves its performance. We show with an extensive evaluation on two popular motion prediction datasets that our method achieves state-of-the-art performance, and we demonstrate the effectiveness of our interaction and environment encoding on a realistic vehicle motion dataset. Lastly, we show qualitatively and quantitatively that the likelihoods FloMo produces are meaningful.

4.2 Related Work

In Ch. 3 we confirmed that classic prediction approaches are still valuable. However, after successes on various other computer vision problems, neural networks have become the most popular type of model for motion prediction. While many approaches using neural networks focus on the prediction of single trajectories [73]–[75], in this chapter, we focus on generative models that learn motion distributions.

With a generative model, it is possible to sample multiple predictions from the learned distribution to reflect various possible outcomes. Two such models based on GANs that are provided with additional context information to learn motion distributions were proposed by Sadeghian et al. [35] as well as Gupta et al. [26]. Varadarajan et al. [76] propose MultiPath++, that uses multi-context gating fusion and latent anchor embeddings to output a Gaussian Mixture Model (GMM) distribution. The Trajectron++ model of Salzmann et al. [28] combines a conditional VAE, Long Short-term Memorys (LSTMs) and spatio-temporal graphs to produce multimodal trajectory predictions. The autoregressive model of Srikanth et al. [77] uses recurrent convolutions and intermediate semantic representations to predict a likelihood grid over future positions. Inspired by BERT, Giuliari et al. [78] propose to use a transformer architecture for motion prediction. Xue et al. [70] propose the Scene Gated Social Graph that models the relations between pedestrians with a dynamic graph that is used to condition a VAE. Mohamed et al. [79] model social interactions with a spatio-temporal graph on which they apply graph convolutions and a temporal Convolutional Neural Network (CNN) to make predictions. Instead of directly predicting trajectories, Mangalam et al. [80] use a conditional VAE to first predict trajectory endpoints and a recursive social pooling and prediction module to infer socially compliant trajectories. The prediction model of Pajouheshgar et al. [81] is fully convolutional and outputs a discrete probability distribution over image pixels.

A different approach to learning complex probability distributions is using normalizing flows. While originally developed for density estimation [82], normalizing flows have recently been applied to various data generation problems [83]–[86]. In the area of motion prediction, normalizing flows have rarely been used. To generate trajectories for a planner, Agarwal et al. [68] sample from a conditional β-VAE [87] that uses a Neural Autoregressive Flow [88] as a flexible posterior. Bhattacharyya et al. [69] use a conditional Flow VAE with condition and posterior regularization to predict trajectories. In their recently published work [89], they use a Block Autoregressive Flow based on Haar wavelets to learn distributions for motion prediction and also adapted FlowWaveNet [83] for motion prediction. Ma et al. [90] recently showed how to find those trajectories sampled from affine flows that are both likely and diverse to make predictions.

The method we propose is a flow-based context-aware generative prediction model that can learn complex multimodal distributions. It allows tractable likelihood computation and can be trained directly with MLE. Most existing generative models only possess some of these properties. In contrast to the flow-based prediction models proposed in concurrent works [89], [90], the flow we use is based on splines and hence is more flexible, which our results demonstrate. Furthermore, we propose a novel noise injection

method that significantly stabilizes training and a data augmentation transformation that further improves our model's generalization and performance. Our extensive experiments show that our model achieves state-of-the-art results on popular motion prediction datasets and that the likelihoods it produces are meaningful and can be used to modulate how concentrated our model's predictions are. Furthermore, we also show that FloMo outperforms the generative CVM, which we utilized in the previous chapter to demonstrate the weaknesses of other recent state-of-the-art generative neural prediction models.

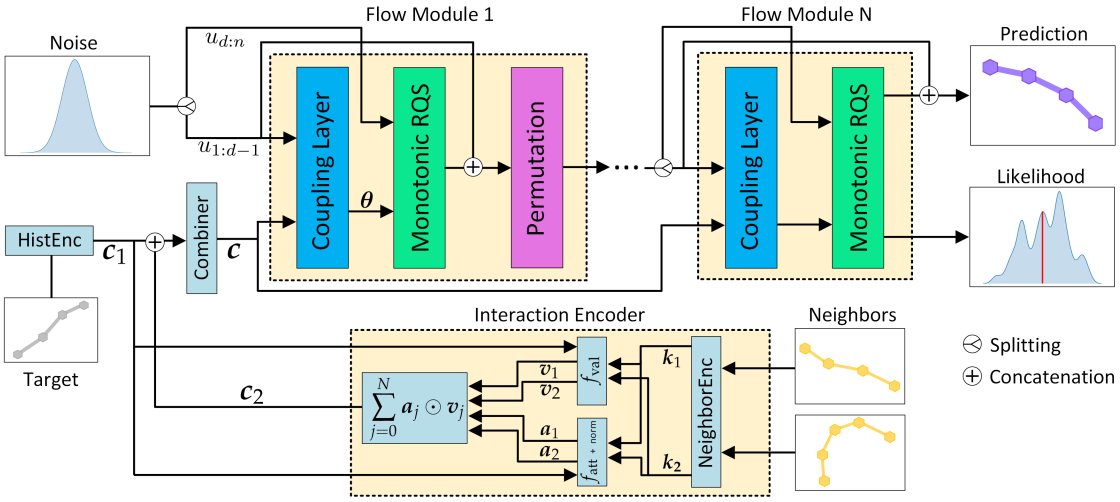
4.3 Motion Prediction as Density Estimation

As we already outlined in Sec. 2.1, the goal of motion prediction is to predict future trajectory $\boldsymbol{x}$ based on the agent's past trajectory $\boldsymbol{o}_i$, and possibly additional information like the past movements of other agents $\mathcal{O} = \{\boldsymbol{o}_j : i \neq i\}$ and the state of the environment $\mathcal{E}$. From the perspective of generative modeling, it then is the goal to learn a conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{o}, \mathcal{O}, \mathcal{E})$ that describes for a specific situation how the agent may move. Multiple concrete predictions can then be made by simply sampling future trajectories $\hat{\boldsymbol{x}} \sim p(\boldsymbol{x} \mid \boldsymbol{o}, \mathcal{O}, \mathcal{E})$.

One way to learn such a distribution is to use normalizing flows (see Ch. 2.3). The described predictions can then be simply computed by sampling $\boldsymbol{u}$ from a standard normal distribution $p_u(\boldsymbol{u})$ and transforming it through the flow into the prediction $\hat{\boldsymbol{x}}$. Because neural networks usually implement normalizing flows, they can be readily provided with conditional information. This effectively turns the learned distribution into a conditional distribution, as it is required for motion prediction. For a good flow-based motion prediction model, it is crucial that the applied transformation is flexible and can represent complex patterns. Furthermore, it is important that the flow can be computed fast in both directions, as we will show in the remainder of this chapter.

4.4 Spline-based Flow Prediction Model

The objective of our model is to learn the conditional motion distribution $p(\boldsymbol{x} \mid \boldsymbol{o}, \mathcal{O}, \mathcal{E})$. However, in practice we encode $\boldsymbol{o}$, $\mathcal{O}$, and $\mathcal{E}$ into a compact representation $\boldsymbol{c}$ and hence we can equivalently learn $p(\boldsymbol{x} \mid \boldsymbol{c})$. We learn this distribution by utilizing normalizing flows. To make a prediction, we then sample $\boldsymbol{u}$ from a standard normal base distribution and pass the noise sample through our model, that we provide $\boldsymbol{c}$ as an additional input variable for conditioning. The output of our model is a sampled trajectory prediction $\hat{\boldsymbol{x}}$. By evaluating Eq. 2.13, we can directly compute the likelihood of each sample in the same network pass. A high-level overview of our architecture is given in Fig. 4.2. The main components of our model are a motion history encoder, an interaction encoder, and neural spline flows as proposed by Durkan et al. [91]. These flows consist of conditional coupling layers [92] and monotonic spline transformations [93]. Note that for encoding environment information, we use a module with the same architecture as for encoding



©IEEE 2021 [65].

Figure 4.2: Our model is composed of multiple flow modules, each containing a coupling layer for conditioning, a monotonic rational-quadratic spline (RQS) transformation and – except the last module – a permutation layer. It receives an encoded observed trajectory, a noise vector, and encoded interaction information and outputs a prediction sample along with its likelihood.

interactions. Therefore we do not display it in Fig. 4.2 or discuss it separately.

The following sections explain each component of our model in detail, including how we prepare our data to achieve stable training, our objective function, and a novel trajectory augmentation transformation that we apply to increase generalization and performance.

4.4.1 Motion History Encoder

The first module of our model is the motion history encoder, which encodes the observed trajectory $\boldsymbol{o}$ of the target agent. Before we encode $\boldsymbol{o}$, we subtract from each position $\boldsymbol{p}^t \in \boldsymbol{o}$ its preceding position, i.e. $\boldsymbol{p}'^t = \boldsymbol{p}^t - \boldsymbol{p}^{t-1}$. This means instead of encoding absolute coordinates, we encode relative displacements, which has proven to be beneficial for motion prediction [46], [61]. From now on, we will denote the resulting relative observed trajectory as $\boldsymbol{o}'$ and its encoding as $\boldsymbol{c}_1$. We implement the encoder as a recurrent neural network with three Gated Recurrent Units (GRU) [94] and a hidden state size of 16. Before we pass each displacement step to the encoder, we embed it with a linear layer in a 16-dimensional vector. The output of the last GRU is then passed through an Exponential Linear Unit (ELU) [95] and again linearly transformed while keeping 16 output dimensions. We determined these hidden and embedding sizes empirically. Because the ELU function is non-zero everywhere, it helps to avoid dying neurons in the network recursion. The recurrent architecture of our encoder enables it to work with input trajectories of various lengths.

4.4.2 Interaction Encoder

To make our model aware of other agents that move in the local neighborhood of the target agent i, we encode their trajectory information with an interaction encoder in vector $\boldsymbol{c}_2$. In particular, we interpret the motion histories of all neighbors $\mathcal{O}$ up to time step t as a set. Because the elements in a set are unordered, it is beneficial to use an order invariant aggregation function [96] to encode the contained trajectories.

Simple order invariant aggregation functions, such as *sum*, *average* or *max* are not trainable and hence limited in their expressive power. While the set aggregation functions proposed by Zaheer et al. [96] can be trained, they treat every element in the set equally, and the resulting descriptive vector is not normalized by the number of elements in the set. However, for interaction-aware motion prediction, it is necessary to identify those agents that affect the target agent the most, and the resulting encoding must be independent of the number of surrounding agents. This assumption can be justified with a simple example: When several neighbors slow down the target agent, usually one of the neighbors – the slowest – dominates this influence. Furthermore, our function must account for the fact that different neighbors can affect the motion of the target agent in different ways. The agent in front of the target agent could slow him down, while the agent to the right of him could alter his movement direction towards the left. Both of these influences should be taken into account independently. To address these requirements, the aggregation function we propose is based on feature-wise attention [97] and uses a query, key, and value mechanism [98].

In particular, our module first subtracts the most recent position $\boldsymbol{p}_i^t$ of the target agent from each observed neighbor trajectory $\boldsymbol{o}_j \in \mathcal{O}$ and encodes the resulting relative trajectories independently with a neighbor motion encoder. We denote the set of relative observed neighbor trajectories as $\mathcal{O}'$. This encoder has the same architecture as the one described in Sec. 4.4.2, but its own weights. The resulting vectors are the keys $\boldsymbol{k}_j$, with one for each neighbor in the target agent's vicinity. The query $\boldsymbol{q} = \boldsymbol{c}_1$ is the encoding of the target agent's motion history itself. By comparing each key with the query, we can then compute a vector that describes the influence of the corresponding neighbor on the target agent. These vectors correspond to the values $\boldsymbol{v}_j = f_{\text{val}}(\boldsymbol{k}_j, \boldsymbol{q})$ and the function f_{val} is implemented by a Feed-Forward Neural Network (FNN). The next step is then to aggregate all computed values to obtain $\boldsymbol{c}_2$, which summarizes how its neighbors influence the target agent.

In most attention mechanisms, a scalar for each value $\boldsymbol{v}$ is computed, the scalars are normalized, and then a weighted sum over all values is computed to summarize them. In contrast to that, we attend feature-wise, such that each neighbor can influence only those particular dimensions that are the most relevant. For example, one neighbor could mainly influence the target agent's direction and nothing else, while another could primarily slow the target agent down. To achieve this, we first compute an unnormalized

score vector $\boldsymbol{s}_j = f_{\text{att}}(\boldsymbol{k}_j, \boldsymbol{q})$ for each key and then compute attention weights as follows:

$$a_j^d = \frac{\exp(s_j^d)}{\sum_{k=0}^{N} \exp(s_k^d)} \quad \text{for all} \quad d < \dim(\boldsymbol{s}_j) \quad \text{and} \quad j. \tag{4.1}$$

The superscript d denotes the dimension we normalize with our softmax. Hence, the sum of a particular dimension across all vectors $\boldsymbol{a}_j$ is one. Then we aggregate all values by computing their sum, weighted with the attention vectors, i.e.,

$$\boldsymbol{c}_2 = \sum_{j=0}^{N} \boldsymbol{a}_j \odot \boldsymbol{v}_j. \tag{4.2}$$

To obtain our final conditioning $\boldsymbol{c} = f_{\text{comb}}(\boldsymbol{c}_1 \oplus \boldsymbol{c}_2)$, we concatenate $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ and then pass it through a combiner network that outputs a 32-dimensional vector. As we mentioned earlier, in the case we also encode environment information, we express it in relative coordinates too (denoted as $\mathcal{E}'$), use the same attention-based encoding architecture, and additionally concatenate the resulting environment encoding $\boldsymbol{c}_3$. We implemented the functions f_{val}, f_{att} and f_{comb} as FNNs with four hidden layers of 32 neurons and a 32-dimensional linear output layer. Between the hidden layers we use Exponential Linear Unit (ELU) activation functions. We empirically determined this parameterization to be suitable for our problem.

4.4.3 Conditional Coupling Layer

One way to design a normalizing flow is to modularize it into a transformation and a conditioner [88]. The conditioner takes the input $\boldsymbol{u}$ and parameterizes the transformation that in turn transforms $\boldsymbol{u}$ into $\boldsymbol{x}$. In our work, it is important that our flow is fast to evaluate both in the forward and inverse direction. For sampling trajectories, we must transform forward from $\boldsymbol{u}$ to $\boldsymbol{x}$, but during training, we have to compute the likelihood of $\boldsymbol{x}$ in the inverse direction (Eq. 2.14). Furthermore, it would be desirable for our model to allow the computation of likelihoods for trajectories that an agent could take but that were not sampled. This also requires the inverse direction.

For the flow to be fast to invert, both the transformation and conditioner must be fast to invert. To achieve this for the conditioner, we use coupling layers [91], [92] to implement our flow. Coupling layers are just as fast to invert as to compute forward. Our coupling layer computes the output $\boldsymbol{x}$ as follows:

$$\begin{aligned} x_{1:d-1} &= u_{1:d-1} \\ \boldsymbol{\theta} &= \text{NN}(u_{1:d-1} \oplus \boldsymbol{c}) \\ x_i &= \tau(u_i; \boldsymbol{\theta}_i) \text{ for } i \geq d. \end{aligned} \tag{4.3}$$

First, we split the input $\boldsymbol{u}$ in half and assign the first part $u_{1:d-1}$ directly to the output. Then we concatenate $u_{1:d-1}$ with context encoding $\boldsymbol{c}$ (see Sec. 4.4.2) and feed it to the conditioner network that computes the parameters $\boldsymbol{\theta}$. Using $\boldsymbol{\theta}$ to parameterize the

invertible transformation τ, we transform the second half $u_{d:n}$ of $\boldsymbol{u}$ element-wise to the remaining corresponding outputs. The resulting Jacobian matrix is lower triangular, and hence its determinant can be easily computed as the product of its diagonal elements [92]. By concatenating $\boldsymbol{c}$ to the conditioner input, we make our flow conditional on the observed target trajectory and context information, such that it learns the target density $p(\boldsymbol{x} \mid \boldsymbol{c})$.

We implement the conditioner as a regular FNN with five hidden layers. Each layer has 32 neurons and is followed by an ELU activation. This configuration worked well empirically. Because half of the inputs are not transformed in a coupling layer, it is crucial to stack several such flow modules and randomly permute the input vectors between the modules. As permutations are volume-preserving, the Jacobian determinant of such a permutation layer is simply 1.

4.4.4 Monotonic Spline Transforms

Transformations used in normalizing flows must be expressive, invertible, and differentiable. In motion prediction, expressive power is crucial to represent complex distributions, and only fast invertibility allows the computation of likelihoods for external trajectories at runtime and short training times. However, the most expressive flows, e.g., neural flows [88], cannot be inverted analytically, and we have to resort to iterative methods like bisection search [31]. On the other hand, flows that are fast to invert often use simple transformations, e.g., affine or linear transformations, and hence are not very expressive.

Recently, Durkan et al. [91] proposed using monotonic rational-quadratic splines (RQS) [93] as flow transformations. This type of flow becomes both expressive and fast to invert in conjunction with coupling layers. The spline transformation described in the following corresponds to the function τ in Sec. 4.4.3.

The spline is defined by K different rational-quadratic functions that pass through $K+1$ knot coordinates $\left\{\left(x^k, y^k\right)\right\}_{k=0}^{K}$. These knots monotonically increase between $\left(x^0, y^0\right) = (-B, -B)$ and $\left(x^K, y^K\right) = (B, B)$. In accordance with Durkan et al., we assign the spline $K-1$ arbitrary positive derivatives $\left\{\delta^k\right\}_{k=1}^{K-1}$ for the intermediate knot connection points and set the boundary derivatives $\delta^0 = \delta^K = 1$ to match the linear 'tails' outside of the rational-quadratic support $[-B, B]$. This support is a hyper-parameter and is set manually. With these parameters, the spline is smooth and fully defined. The neural network parameterizing it can learn the knot positions and boundary derivatives during training.

The spline transformation is then applied element-wise, e.g., to a given scalar input x_{in}. If x_{in} is outside the support, the identity transformation is applied. Otherwise, the correct knot bin is determined first and then

$$\begin{aligned} s_k &= \left(y^{k+1} - y^k\right) / \left(x^{k+1} - x^k\right) \\ \xi &= \left(x_{\text{in}} - x^k\right) / \left(x^{k+1} - x^k\right) \end{aligned} \tag{4.4}$$

are computed. After this, the forward transformation

$$\frac{\alpha^k(\xi)}{\beta^k(\xi)} = y^k + \frac{\left(y^{k+1} - y^k\right)\left[s^k\xi^2 + \delta^k\xi(1-\xi)\right]}{s^k + \left[\delta^{k+1} + \delta^k - 2s^k\right]\xi(1-\xi)} \tag{4.5}$$

defined by the k^{th} bin can be evaluated. For the inverse transformation, derivatives to compute the Jacobian determinant and further details, we refer the reader to [91].

In practice, the knot coordinates and derivatives come from the conditioner network. For each input dimension i, its output $\boldsymbol{\theta}_i = \left[\boldsymbol{\theta}_i^w, \boldsymbol{\theta}_i^h, \boldsymbol{\theta}_i^d\right]$ is simply partitioned into vectors of length K, K and $K-1$ for the knot widths and heights, as well as the knot derivatives. To compute the actual knot coordinates, $\boldsymbol{\theta}_i^w$ and $\boldsymbol{\theta}_i^h$ are multiplied by 2B and their cumulative sums starting from $-B$ are computed.

Finally, the sampled output of our model (after the last spline transformation) represents the predicted trajectory as relative displacements. Using relative displacements limits the numeric range of the output and is important to stay within the support $[-B, B]$ of the spline transformations. We denote this estimated relative displacements as $\hat{\boldsymbol{x}}'$. To convert it back to absolute coordinates, we compute the cumulative sum over all positions $\boldsymbol{p}' \in \hat{\boldsymbol{x}}'$, starting from the last observed absolute position $\boldsymbol{p}_i^t$ of the target agent. Because the relative observed trajectory $\boldsymbol{o}'$, the relative context information $\mathcal{O}'$ and $\mathcal{E}'$, and the relative future trajectory $\boldsymbol{x}'$ can be unambiguously mapped back to $\boldsymbol{o}$, $\mathcal{O}$, $\mathcal{E}$, and $\boldsymbol{x}$, it holds that $p(\boldsymbol{x}' \mid \boldsymbol{o}', \mathcal{O}', \mathcal{E}') = p(\boldsymbol{x} \mid \boldsymbol{o}, \mathcal{O}, \mathcal{E})$. Hence we learn the target distribution.

Furthermore, like in [99], before making a prediction we rotate the whole scene around $\boldsymbol{p}_i^t$, such that the last relative displacement $\boldsymbol{p}_i^t - \boldsymbol{p}_i^{t-1}$ of the target agent is aligned with the vector $(1, 0)$. After sampling our model, we rotate the predicted trajectories back with the corresponding inverse rotation. This transformation simplifies the distribution our model must learn and makes it rotation invariant. This is especially important for models that should generalize as well as possible because directional environmental priors (see Sec. 3.4.1) are suppressed. Because rotations are volume-preserving, we do not have to consider this transformation in our flow's likelihood computation.

4.4.5 Preventing Manifolds

Whenever data is distributed such that it – or a subset of it – is residing on a lower-dimensional manifold, this leads to infinite likelihood spikes in the estimated density. Consider the two-dimensional example with joint density $p(x, y)$, where x is normally distributed and $y = x$. The distribution resides on a line and for $\int\int p(y \mid x)p(x)\mathrm{d}y\mathrm{d}x = 1$ to hold, the likelihoods where y is defined must be infinite.

In practice, this problem also arises when certain dimensions in the dataset samples frequently take on equal values or when one dimension frequently takes the same value. Because we predict relative displacements $\boldsymbol{x}'$ instead of absolute coordinates with our model, this can happen if pedestrians stand still (values become zero) or move with constant velocity for multiple time steps (values are equal). During training, this can

cause numerical instabilities, loss volatility, and the overestimation of specific samples' likelihoods.

Furthermore, assigning inflated likelihoods that do not correlate with better predictions can cause severe problems in downstream tasks. To decide which predicted sample to prioritize, it is reasonable to compare their likelihoods. Alternatively, it is possible to approximate the underlying distribution by drawing many samples and normalizing their likelihoods with functions such as softmax. However, the learned likelihood artifacts could cause making wrong choices by comparing likelihoods, or when normalizing with softmax, it could cause the resulting distribution to be very pronounced around the inflated likelihood and very flat everywhere else. Hence, decision-making based on such information becomes more difficult.

To mitigate this problem, and inspired by [100], we define three hyper-parameters α, β and γ. While training, when transforming $\boldsymbol{x}'$ to $\boldsymbol{u}$ through the inverse of our flow, we augment $\boldsymbol{x}'$ before our first flow module with

$$\begin{aligned} \boldsymbol{x}'' &= \alpha \boldsymbol{x}' \\ x_i''' &= x_i'' + \epsilon_{\beta i} \quad \text{for all} \quad x_i'' = 0 \\ x_i''' &= x_i'' + \epsilon_{\gamma i} \quad \text{for all} \quad x_i'' \neq 0. \end{aligned} \tag{4.6}$$

We sample noise vectors $\boldsymbol{\epsilon}_\beta$ and $\boldsymbol{\epsilon}_\gamma$ from zero-centered normal distributions with standard deviation β and γ, respectively. However, we only apply noise during the training phase and not at inference time. In the forward pass, we always compute $\boldsymbol{x}' = \alpha^{-1}\boldsymbol{x}''$ after our last flow module to normalize predicted trajectories. By adding the noise during training, we essentially lift data off potential manifolds. Generally speaking, we apply less noise to zero-valued dimensions and more to non-zero displacement vectors. Scaling $\boldsymbol{x}'$ with α allows us to inject more noise while controlling the impact of the noise on the trajectory. It is important to pick all hyperparameters, such that the noise injected trajectory $\boldsymbol{x}'''$ stays within spline support $[-B, B]$. As a result, our noise injection method reduces the general magnitude of likelihoods, reduces likelihood outliers, and stabilizes training, as we will show in Sec. 4.5.6.

4.4.6 Objective Function

Because our model makes it easy to compute likelihoods for training examples, we simply train it with MLE. In particular, we minimize the negative log-likelihood

$$\text{NLL} = -\frac{1}{N}\sum_{i=1}^{N} \log(p(\boldsymbol{x}_i \mid \boldsymbol{c}_i)). \tag{4.7}$$

As maximizing the data's likelihood under the model is equivalent to minimizing the Kullback-Leibler divergence (KLD) of the model distribution and the data distribution, we can be sure that our model learns the true distribution. This is superior compared to using proxy losses, such as those applied in VAEs and GANs that don't provide these

types of guarantees.

4.4.7 Trajectory Augmentation

To increase the diversity of our data, we augment trajectories by randomly scaling them. In particular, for each trajectory, we sample a scalar in the range $[s_{\min}, s_{\max}]$ from a truncated normal distribution. Before multiplying the trajectory element-wise with the scalar, we first center the trajectory by subtracting its mean position to avoid translating it with the scaling operation and then move it back. Scaling a trajectory does not influence its direction and motion pattern but simulates varying movement speeds. It is crucial to stay within realistic limits by applying this transformation, and the correct choice for the sampling interval depends on the used data. In the case that we use context information, it is scaled accordingly.

4.5 Experiments

To evaluate our model, we use the the publicly available ETH [38] and UCY [59] datasets, the Stanford Drone [101] dataset and the Argoverse [99] dataset. All datasets are based on real-world video recordings and contain complex motion patterns. The ETH/UCY datasets are evaluated jointly and focus on pedestrians recorded in city centers and at university campuses. They cover five scenes with four unique environments and 1950 individual pedestrians. The larger Stanford Drone dataset contains 10300 individual traffic participants. It covers roads, and besides pedestrians, it also includes other agent types like cyclists and vehicles. The extensive Argoverse dataset contains primarily vehicles trajectories and was created based on 1000 hours of driving data recorded in two major cities. All datasets are heavily used in the motion prediction domain [26], [28], [35], [36], [61], [89], [102]. For each dataset in our experiments, we use the evaluation regime that is the most common in literature. In the following sections, we explain each experiment setup in detail and present results.

4.5.1 Metrics

Because our model is learning a distribution, we must evaluate multiple predictions for each test set example to assess its performance. In our experiments, we will – depending on the dataset – make use of a subset of the following metrics that all are suited to evaluate generative prediction models:

- *Minimum Average Displacement Error (minADE)* — Error of the sample with the smallest average l^2-distance between all corresponding positions in the ground truth and the predicted trajectory.
- *Minimum Final Displacement Error (minFDE)* — Error of the sample with the smallest l^2-distance between the last position in the ground truth and the last position in the predicted trajectory.

- *Oracle Top 10% (OR)* — Average error of the top 10% best predicted trajectories at different time steps. It has been shown that this measure is robust to random guessing and simply increasing the number of drawn samples does not affect it [69].

- *Miss Rate (MR)* — Percentage of examples for which no prediction endpoint was within a certain threshold radius (2 m in this work), and hence the target was missed.

4.5.2 ETH and UCY Datasets

To evaluate the ETH/UCY datasets, we train on four scenes and evaluate the remaining one, as we also did in Sec. 3.3.2. We slice each trajectory with a step-size of one into sequences of length 20, of which 8 time steps are observed, and 12 must be predicted. This corresponds to an observation window of 3.2 s and a prediction of 4.8 s. In our evaluation, we only use those trajectories with a length of at least 10 time steps, i.e., at least two time steps to predict. Following the standard protocol for these datasets, each model was allowed to predict 20 trajectory samples in this evaluation.

Training

For training our model, we only take into account trajectories of full length because padding would cause issues as described in Sec. 4.4.5. For the ETH/UCY datasets, we trained our model with the Adam Optimizer [17], learning rate 0.001, and batch size 128 for 150 epochs. We randomly split a 10% validation set that we use to detect overfitting. Furthermore, we define the support for each spline flow as $B=15$ and use 8 knot points. In total, we stack 10 flow layers in our model. To parameterize our model's noise injection, we set $\alpha=10$, $\beta=0.2$ and $\gamma=0.02$ and for our scaling transformation we use $\mu=1$, $\sigma=0.5$, $s_{\min}=0.3$ and $s_{\max}=1.7$. All hyper-parameters described were determined empirically by testing.

Models

We compare three versions of our model with a variety of state-of-the-art prediction models. For our model, we use one basic version without advanced modifications (**FloMo**), one with with our scaling augmentation (**FloMo-S**) and one with scaling and our interaction module (**FloMo-I**). We do not use environment information as the datasets – besides videos – only contain trajectories. All other models we compare with, except the **CVM-S** [61], are based on neural networks. **S-STGCNN** [79], **SGSG** [70] and **Trajectron++** [28] utilize neural networks in combination with graphs. $\mathbf{TF}_q$ [78] is based on the transformer architecture, and **S-GAN** [26] and **SoPhie** [35] are GANs.

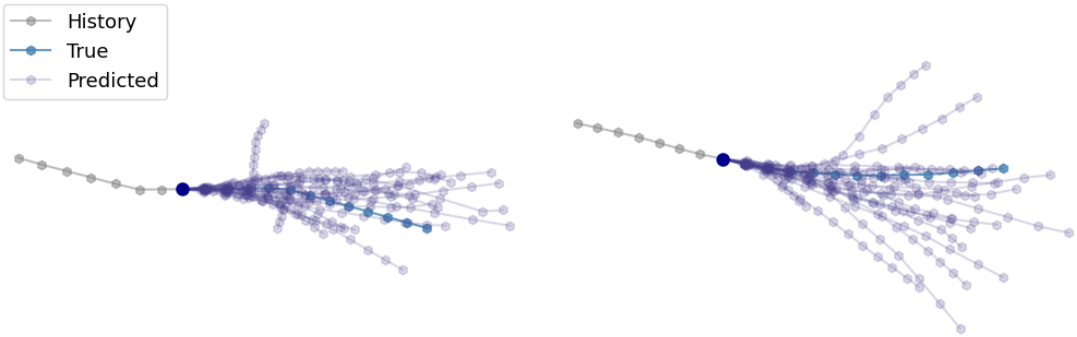
Model	ETH-Uni	Hotel	UCY-Uni	Zara1	Zara2	AVG
SGSG	0.54 / 1.07	0.24 / 0.45	0.57 / 1.19	0.35 / 0.79	0.28 / 0.59	0.40 / 0.82
S-STGCNN	0.64 / 1.11	0.49 / 0.85	0.44 / 0.79	0.34 / 0.53	0.30 / 0.48	0.44 / 0.75
S-GAN	0.59 / 1.04	0.38 / 0.80	0.27 / 0.49	0.18 / 0.33	0.19 / 0.35	0.32 / 0.60
Sophie	0.70 / 1.43	0.76 / 1.67	0.54 / 1.24	0.30 / 0.63	0.38 / 0.78	0.54 / 1.15
CVM-S	0.44 / 0.81	0.20 / 0.35	0.34 / 0.71	0.25 / 0.49	0.22 / 0.45	0.29 / 0.56
Trajectron++	0.39 / 0.83	**0.12** / **0.21**	**0.20** / **0.44**	**0.15** / **0.33**	**0.11** / **0.25**	**0.19** / 0.41
TF$_q$	0.61 / 1.12	0.18 / 0.30	0.35 / 0.65	0.22 / 0.38	0.17 / 0.32	0.31 / 0.55
FloMo	0.58 / 0.95	0.16 / 0.23	0.25 / 0.45	0.19 / 0.34	0.17 / 0.31	0.27 / 0.46
FloMo-S	**0.32** / **0.52**	0.15 / 0.22	0.25 / 0.46	0.20 / 0.36	0.17 / 0.31	0.22 / **0.37**
FloMo-I	**0.32** / **0.52**	0.16 / 0.23	0.27 / 0.46	0.19 / 0.34	0.18 / 0.32	0.22 / **0.37**

Table 4.1: Displacement errors for scenes in the ETH/UCY datasets and on average. We compare our model to six state-of-the-art models. Each model predicted 20 trajectory samples and the errors are shown as minADE / minFDE.

Results

For the ETH/UCY datasets, we compare the performance of our model with state of the art in Tab. 4.1. From the versions of our model that we included in the evaluation, it becomes clear that, on average, the version using our scaling transformation (FloMo-S) performs significantly better than the plain FloMo. This difference in performance is almost entirely due to scene ETH-Uni. Most likely, the pedestrians in this scene move at different speeds compared to the other scenes. Including interactions did not have a significant influence on our model's performance. This is not surprising based on the results of our analysis in Sec. 3.4.3, which shows that in pure pedestrian datasets like ETH and UCY, interactions are hard to predict.

In comparison with state-of-the-art models, except for the Trajectron++, our model significantly outperforms all other models on average errors, both in terms of Minimum Average Displacement Error (minADE) and Minimum Final Displacement Error (minFDE). Compared to the Trajectron++, our model performs better on the ETH-Uni scene, while on the other scenes, the Trajectron++ achieves lower errors, especially for minADE. For the minFDE, both models perform close on all scenes, except ETH-Uni and Zara2. In total, the Trajectron++ achieves lower errors averaged over the whole trajectories with a minADE of 0.19. However, FloMo-S performs better on the endpoints prediction, where it achieves a minFDE of 0.37. Hence, the prediction performance of both models can be considered approximately equivalent. Furthermore, note that unlike the Trajectron++ our model is tractable and allows direct likelihood computation. The comparable performance of both models could indicate that the noise floor for predictions on the ETH/UCY datasets is approached.



(a) Zara01 **(b)** UCY-Uni

Figure 4.3: Prediction examples for pedestrians in the scenes Zara01 and UCY-Uni from the ETH/UCY datasets.

In Fig. 4.3 we show prediction examples of FloMo-S for the Zara01 and UCY-UNI scenes. Our model predicts a high diversity of possible future movements, including different directions, speeds, and acceleration changes. This is because pedestrians move chaotically at times, and hence, the model learns wide distributions. In both cases, at least one predicted trajectory covers the ground truth trajectory accurately.

4.5.3 Stanford Drone Dataset

For the Stanford Drone dataset, we randomly split it into training and test set but ensure that both sets do not contain parts of the same video sequences. We observe a motion history of 20 time steps and predict the next 40 time steps, which corresponds to 2 s observation and 4 s prediction, respectively. As for the ETH/UCY datasets, during evaluation our model must predict at least two time steps, i.e., we reject all trajectory slices shorter than 22 time steps. Because the Stanford Drone dataset is large, we use this minimum trajectory length of 22 as our step-size for slicing.

For comparability with related work, we follow [27], [89] and scale the dataset trajectories by a factor of $1/5$. In this experiment, we compare our model with both tractable and intractable state-of-the-art models. We evaluate both model types in terms of displacement errors and do not rely on comparing tractable models with log-likelihoods. While our model's likelihoods are meaningful, as we will show in Sec. 4.5.5, the overall log-likelihood for trajectory datasets is largely dominated by manifold artifacts and hence not ideal for assessing prediction performance.

Training

We train our model with the same architecture and setting as for the ETH/UCY datasets. However, in this experiment, we split a 5% validation set from our training set to detect

Model	@1 s	@2 s	@3 s	@4 s
STCNN	1.20	2.10	3.30	4.60
FlowWaveNet	0.70	1.50	2.40	3.50
HBA-Flow	0.70	1.40	2.30	3.20
FloMo	**0.26**	0.63	1.03	1.29
FloMo-S	0.27	**0.56**	**0.90**	**1.27**
FloMo-I	0.30	0.62	0.99	1.38

Table 4.2: Errors for the Stanford Drone dataset, evaluated with a five-fold cross-validation, the Oracle Top 10% metric and 50 predicted trajectories. All models are tractable and allow exact likelihood computation.

overfitting and set $\alpha = 3$, $\beta = 0.002$, $\gamma = 0.002$. For our scaling augmentation, we use $\mu = 1$, $\sigma = 0.2$, $s_{\min} = 0.8$, and $s_{\max} = 1.2$.

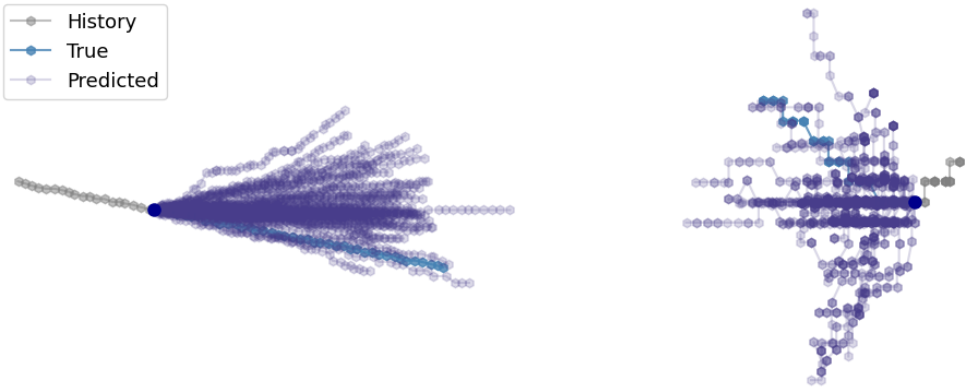
Models

To evaluate the Stanford Drone dataset, we include the same three versions of FloMo as in the previous section. The exact inference models we include in our comparison are the **STCNN** [81], the **FlowWaveNet** [83], [89] that has been adapted for motion prediction, and the **HBA-Flow** [89]. The latter two – from concurrent work – are like our model based on normalizing flows.

In our comparison with primarily intractable models we include the **S-GAN** [26], **SoPhie** [35], the **CF-VAE** [69], **PECNet** [80], and again the HBA-Flow, as we evaluate with a different metric. While S-GAN and SoPhie are based on GANs, CF-VAE and PECNet use conditional VAEs as their core network. In the CF-VAE, the prior of its VAE is based on a normalizing flow.

Results

The results for our comparison with tractable prediction models are shown in Tab. 4.2. In this experiment, we evaluated with five-fold cross-validation and let each model predict 50 samples. As evaluation metric, we used the Oracle Top 10% (OR) that we described earlier, evaluated at four different time steps. This is in accordance with how [81] and [89] evaluated. Our results show that our model significantly outperforms all other tractable models at each time step, with a margin of 60% at 4 s over the second-best model HBA-Flow. These results demonstrate that our model captures the true underlying distribution better than the other tractable models. The contribution of our scaling augmentation was moderate but overall positive for the Stanford Drone dataset, except for the evaluation at one second.



(a) (b)

Figure 4.4: Prediction examples for the Stanford Drone dataset. Despite the different structure of both trajectories, our model's predictions match the ground truth.

We also compared our FloMo on the Stanford Drone dataset with intractable models. In this experiment, we just split the dataset a single time, let each model predict 20 future trajectories, and evaluated with the minADE and the minFDE metrics. The results of this experiment are shown in Tab. 4.3 and confirm those of the previous one. FloMo-S significantly outperforms all compared models, with a margin of 74% in minADE and 72% in minFDE, compared to the best intractable state-of-the-art model PECNet.

Including interactions did not improve our model's performance in both our evaluations. This can be explained by the fact that pedestrians and cyclists dominate the Stanford Drone dataset, but the number of contained vehicles is relatively small. As we have shown earlier in Sec. 3.4.3, predicting interactions between pedestrians is infeasible. Including interactions contributed even negatively. Most likely, our model overfitted specific interaction patterns in the training data that did not translate to the test set. These negative results are aligned with those reported by Gupta et al. [26], where including interactions increased the error of their model S-GAN on the ETH/UCY dataset.

In Fig. 4.4 we show two qualitative prediction examples of our model for the Stanford Drone dataset. In both examples, our model closely tracks the ground truth trajectory. Note the difference between how the trajectories in both examples are shaped. The points in trajectory 4.4a are distributed more smoothly and continuously, while those in 4.4b almost seem to be discretized. This is an artifact of the Stanford Drone dataset's labeling that perhaps was performed by different people. Our model learned these patterns and, in both cases, made predictions that are distributed accordingly. This demonstrates our model's capability to model complex distributions and various complex motion patterns.

Model	minADE	minFDE
S-GAN	27.23	41.44
SoPhie	16.27	29.38
CF-VAE	12.60	22.30
HBA-Flow	10.80	19.80
PECNet	9.96	15.88
FloMo	2.92	5.02
FloMo-S	**2.60**	**4.43**
FloMo-I	3.59	6.19

Table 4.3: Evaluation results for Stanford Drone with a single dataset split, 20 predicted trajectories and the minADE / minFDE metrics. Here we also include intractable models.

4.5.4 Argoverse Dataset

With the experiments on the Argoverse dataset, it is not our goal to outperform state-of-the-art methods. Instead, we aim to demonstrate that integrating interaction and environment information into our model is possible. Because the dataset is focused on vehicles on regular roads, we further want to confirm the hypothesis we formulated in Sec. 3.4.4, that interactions are relevant for vehicle prediction and the explicit use of environment information in a diverse dataset can improve predictions.

The Argoverse dataset consists of a fixed training-, validation- and test set split. In our evaluation, we observe 20 time steps and predict the next 30 time steps. Because the dataset was recorded with 10 Hz, this corresponds to 2 s observation and 3 s prediction, respectively. In this evaluation, we only use trajectories of full length, both during the training and the test phase.

Training

As in the previous experiments, we train our model with the Adam Optimizer, learning rate 0.001, and batch size 128. However, for the Argoverse dataset, we train for 70 epochs. We define our model in the same fashion as before, but use noise parameters $\alpha = 2$, $\beta = 0.02$, and $\gamma = 0.01$. Furthermore, we omit our scaling transformation in this case because each scene (not the sample to predict) evaluated is also part of the training set. Hence, both training set and test set stem from the same distribution, and thus improving generalization is not a concern. Furthermore, because we also use environmental information, we would have to scale the environment too. However, road dimensions are standardized, and therefore this would make our model learn unrealistic prediction cases.

Models

Because we do not apply our scaling augmentation, we only include the basic version **FloMo** of our model, **FloMo-I** with the interaction module, and a new version **FloMo-I-M**. In FloMo-I-M we additionally encode map information, and concatenate it to the input of our model's combiner network. In particular, Argoverse provides centerlines for each lane as polylines. We split these polylines into segments of length 10, encode each segment with a FNN (with four 32 neuron hidden layers, ELU activations, and a linear output) in a 32-dimensional embedding, and aggregate with the same feature-wise attention module that we use for encoding interactions (but separate weights).

Furthermore, we evaluate with the **CMV-S**, and include the best baseline models **LSTM+map(prior) 1-G,n-C** and **NN+map(prune)** that were proposed in the Argoverse paper [99]. Both models utilize map features, i.e., predict along map centerlines or prune predictions based on drivable area compliance, respectively.

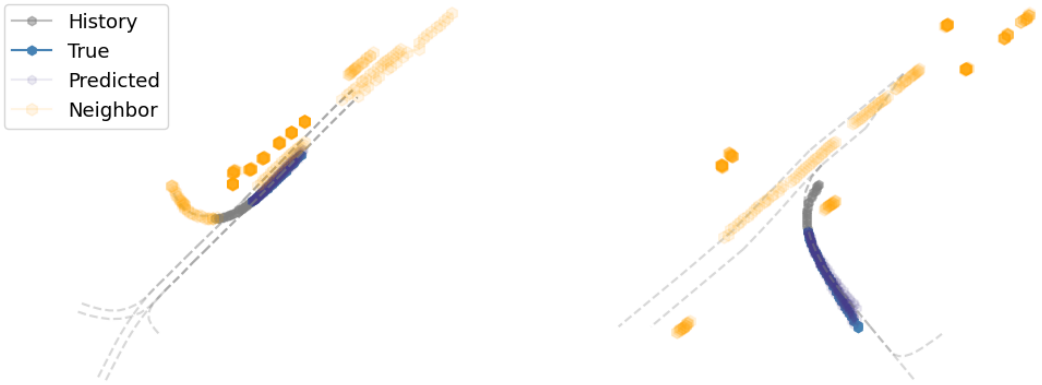
Results

Model	minADE	minFDE	MR
CVM-S	4.31	8.72	0.88
LSTM+map(prior) 1-G,n-C	2.08	4.19	0.67
NN+map(prune)	1.68	3.19	0.52
FloMo	1.65	3.08	0.52
FloMo-I	1.43	2.38	0.39
FloMo-I-M	**1.38**	**2.26**	**0.35**

Table 4.4: Results of our evaluation on the Argoverse dataset. Each model was allowed to make 6 predictions. Adding interactions and environment information to FloMo significantly reduced its prediction error.

In Tab. 4.4 we show our evaluation results for the Argoverse dataset. Amongst the versions of our model, FloMo-I performs better than FloMo, which confirms our suspicion from Sec. 3.4.4 that interactions between vehicles are much more predictable and information about neighbor vehicles could contribute to better prediction results. Providing the model with road centerline markings further reduced the model's prediction errors, even though not as significantly as the interaction information. Compared to the other models, FloMo-I-M outperforms both the simple CVM-S and the baselines provided by the Argoverse paper.

Fig. 4.5 shows two predictions of FloMo-I-M for the Argoverse dataset. In both cases, the target vehicles are driving in a curve, and our models accurately predict their future trajectories. Compared to the predictions for pedestrians in Fig. 4.3 and Fig. 4.4, the



(a) (b)

Figure 4.5: Prediction examples for the Argoverse dataset. The dashed lines represent the road's centerline markings that our model receives as inputs.

predicted trajectories are much more concentrated around the most likely outcome. This indicates that vehicle movements are more predictable and less chaotic than those of pedestrians. Because our model accurately predicts how strong the target vehicle will turn in both cases, we believe it was able to use the lane information that we provided it with to its advantage.

4.5.5 Likelihoods

To verify that the likelihoods our model provides are relevant, we generate 20 trajectory predictions with our basic FloMo model for each example in the ETH/UCY test sets and rank them in descending order by likelihood. Then we compute the expected Average Displacement Error (ADE) and Final Displacement Error (FDE) for each likelihood ranking position across all test sets. As for the evaluation in Sec. 4.5.2, for each test set evaluation, we use the FloMo trained on the remaining scenes. Fig. 4.6 shows graphs of how the expected errors change with likelihood ranking. As expected, a higher likelihood (lower rank) corresponds to lower errors for both ADE and FDE. This proves that the likelihoods computed by our model are meaningful and can be used for decision-making.

To qualitatively demonstrate how likelihoods relate to the predicted trajectories, in Fig. 4.7a we show 20 regularly predicted trajectories, and in Fig. 4.7b a top-k prediction for the same example. For the top-k prediction, we sample 100 trajectory candidates and only keep the 20 most likely ones. The regular predictions are much more spread out. Our model predicts sudden turns, acceleration, or deceleration. The top-k predictions are more concentrated around the true and most likely outcome of the pedestrian's movement. Furthermore, the predicted velocities are more regular. These results demonstrate that an autonomous agent can utilize the likelihoods our model provides to decide which predictions it should prioritize for its planning.

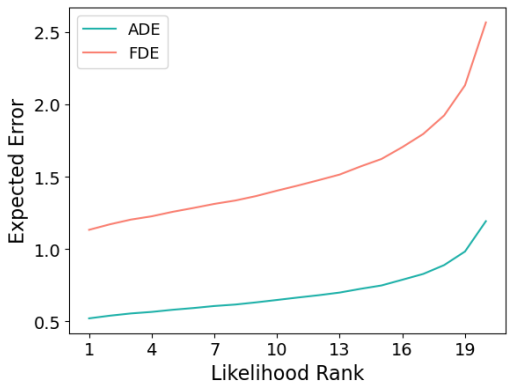
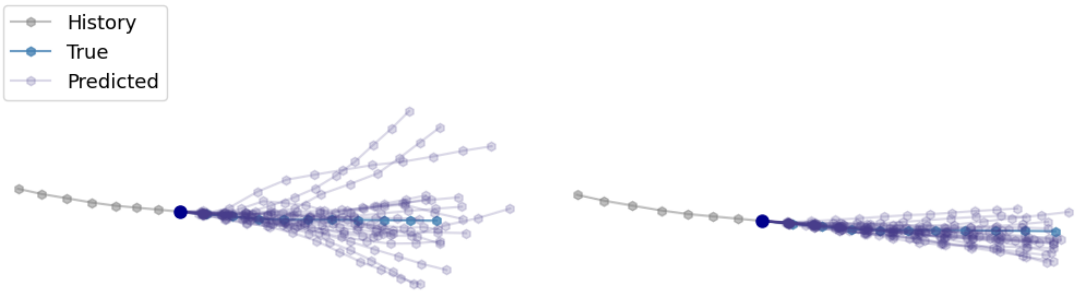


Figure 4.6: We ranked our model's predictions for the ETH/UCY datasets by likelihoods and computed the expected ADE and FDE for each ranking position. Higher likelihoods are correlated with lower errors.



(a) Regular predictions **(b)** Top-k predictions

Figure 4.7: Comparison of our model's regular predictions with top-k predictions for the same sample.

4.5.6 Noise Ablation

To understand the impact of injecting noise in our model's training process, we compare prediction results with and without it. In this comparison, we use the model FloMo-S and report average errors for the ETH/UCY datasets, using the same evaluation regime described in Sec. 4.5.2. Tab. 4.5 shows the impact of our noise injection on the prediction errors of our model. Both versions perform equivalently well, which means that our noise injection does not significantly affect prediction performance. We hypothesize that this is because the extremely inflated density points are sparsely distributed. Instead of spanning regions in the distribution's density, they only resemble Dirac delta impulses with a small probability.

However, injecting noise in the training of normalizing flows for motion prediction has other benefits. In particular, it stabilizes the training process of our model and

Model	minADE	minFDE
Noise	0.22	0.37
No Noise	0.22	0.37

Table 4.5: Ablation of our noise injection for the model FloMo-S and the ETH/UCY datasets. The reported errors are averaged over all scenes. The injection of noise does not significantly influence prediction performance.

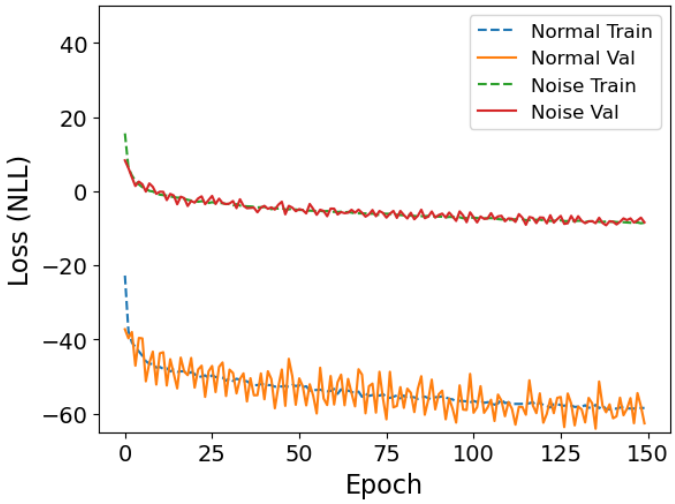
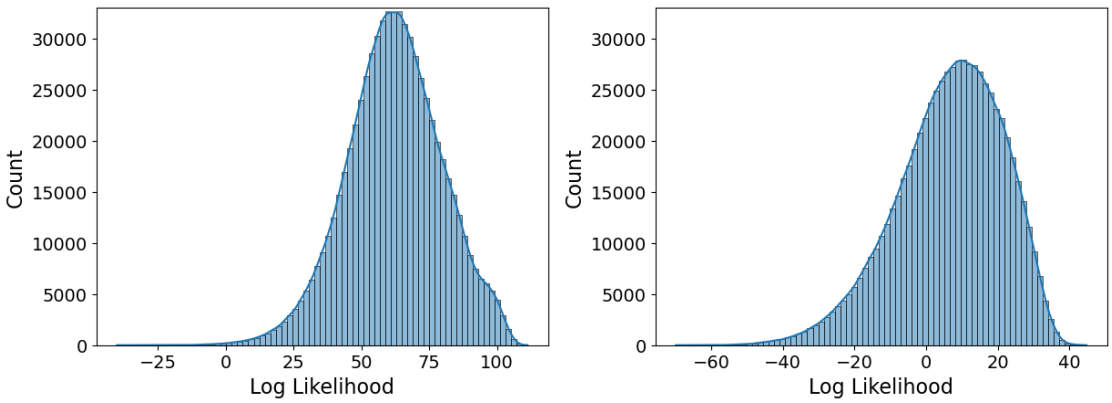


Figure 4.8: Comparison between normally training our model and with our proposed noise injection. The training becomes more stable and likelihoods stay in a reasonable range.

helps to avoid numeric problems. In Fig. 4.8 we show how the loss of our model behaves when trained with (upper curves) and without (lower curves) noise injection. Without noise injection, the loss is very volatile, especially for the validation dataset, and the likelihoods produced by our model are very large. Because we use the negative log-likelihood loss function (see Sec. 4.4.6), these large likelihoods lead to an artificially low overall loss. However, this low loss does not correlate with a better trained model. Instead, these large likelihoods can lead to numeric problems during training. We occasionally experienced overflows and training crashes due to too large values. By injecting noise, the magnitudes of the likelihoods are significantly reduced because samples that originally lay on manifolds get smaller likelihood values assigned. They stop dominating the training, which also reduces the volatility of our validation loss.

Besides a better training process, injecting noise also enables the use of likelihoods in downstream tasks. For example, for safely planning its future actions, an autonomous agent should consider a wide variety of future scenarios. This can be achieved by



(a) No Noise **(b)** Noise Injection

Figure 4.9: Distribution of likelihoods without and with our noise injection, generated by sampling 20 predictions for each test set example in the ETH/UCY datasets.

drawing a large number of prediction samples. To choose which of these samples are the most relevant to be included in its planning, it can compare the samples' likelihoods. However, when the likelihoods of specific samples are artificially inflated, this comparison fails. Besides that, likelihoods are often normalized with functions like softmax to approximate the true underlying distribution. However, when some likelihoods are overproportionally large, the resulting distribution could be very pronounced around the prominent likelihoods but flat elsewhere. This would render the distribution uninformative for decision-making. In Fig. 4.9 we predicted 20 samples for each test set example in ETH/UCY and computed a histogram over all likelihoods. It can be seen that the likelihoods without noise injection are very large (note that we display likelihoods in log scale). The step on the right end of distribution Fig. 4.9a shows that a significant amount of samples are likelihood outliers. The likelihood range becomes more reasonable with noise injection, and extreme values become infrequent, i.e., the likelihoods are more regularly distributed, as Fig. 4.9b shows. These observations support our argument that with noise injection, our model's likelihoods become more practical for use in downstream tasks.

4.6 Conclusion

In this chapter, we proposed a motion prediction model based on spline flows that can learn distributions over the future motion of agents. Furthermore, we developed a fine-grained attention module that enables our model to account for interactions between agents and showed that this module can also be used to encode environment information. Our model enables the direct computation of likelihoods, which is necessary for autonomous agents to prioritize predictions. This is an important feature, because the continuous distribution over future trajectories has infinitely many possible outcomes. By drawing a sufficient amount of samples from such distribution and prioritizing

them, an autonomous agent can plan its actions to take the most relevant outcomes into account and minimize its risk for collision.

With our scaling transformation, we can augment the model's training data such that different movement speeds are simulated, and the data becomes more diverse and hence our model generalizes better to distribution shifts. Furthermore, because training on trajectory data directly causes loss volatility and numerical instabilities, we proposed a method of injecting noise, such that training is stabilized, but the motion information in the trajectories is preserved. Besides the training itself, this is especially important for the use of likelihoods in downstream tasks.

To evaluate our model, we conducted extensive experiments. We showed that our model achieves state-of-the-art performance on two popular benchmark datasets and performed an extensive ablation study that shows that all of our model's components contribute positively.

To evaluate our model, we conducted extensive experiments. We showed that our model achieves state-of-the-art performance on two popular benchmark datasets regarding displacement errors. Furthermore, we performed ablations and showed that our interaction and environment encodings significantly improved prediction performance on a large vehicle dataset. For all experiments we also provided qualitative prediction examples. With additional analyses, we demonstrated that our model's likelihoods are meaningful and that our noise injection enables their use in downstream tasks, e.g., for decision making in autonomous agents.

Chapter 5

Intelligent Infrastructure Providentia

The motion prediction methods we developed in this work aim to support the Intelligent Infrastructure System (IIS) Providentia. Providentia is a large-scale distributed sensor system for the assistance of autonomous and legacy vehicles on the highway. In this chapter, we motivate why an IIS like Providentia is a valuable vehicle support. We describe how we designed the system in terms of its architecture and evaluate its perception performance under real-world conditions.

5.1 Motivation

The environmental perception and resulting scene understanding of an autonomous vehicle are limited by sensor ranges and object detection performance. Even in the vicinity of the vehicle, the existence of occlusions can lead to incomplete information about its environment. The resulting uncertainties pose a safety threat to other road users and the autonomous vehicle itself. The vehicle must then reduce its driving speed to mitigate the resulting risks, which slows down traffic. Furthermore, this incomplete information results in impaired driving comfort, as the vehicle must spontaneously react to unforeseen scenarios.

IISs can alleviate these problems by providing autonomous vehicles – as well as conventional vehicles and drivers – at operating time with complementing information about each road user and the overall traffic situation [14], [106], thereby greatly extending their perception range as well. In particular, an IIS can observe and detect all road users from multiple superior perspectives, and with extended coverage compared to that of an individual vehicle. Providing a vehicle with this additional information gives it a better and spatially extended understanding of its surrounding scene and enables it to plan its maneuvers more safely. Moreover, an IIS with the described capabilities enables a multitude of additive services that can further support decision making.

Building such a system involves many challenges, such as the right choice of hardware and sensors, and their optimal deployment and utilization in a complex software stack. Its perception must remain reliable and robust under a wide variety of weather, light, and traffic conditions. Ensuring this reliability requires a combination of multimodal sensors,

Parts of this chapter have been previously published in [103]–[105].

Figure 5.1: One of the Providentia measurement points on the A9 highway in Germany. The two radars directed towards the north are installed on the other side of the gantry bridge and are therefore not visible from this perspective.

redundant coverage with overlapping Field of Views (FoVs), accurate calibration [107], and robust detection and data fusion algorithms.

In this chapter, we propose a concrete and scalable architecture for such an IIS. While the initial idea for this system has already been sketched earlier [108], [109], the architecture we propose is the result of the real-world build-up experience we made with Providentia (see Fig. 5.1). Our description includes the system's hardware and the software to operate it. In particular, we discuss the choice of sensors, the network architecture, and the deployment of edge computing devices to enable fast and distributed processing of heavy sensor loads. We outline our software stack and the detection and fusion algorithms used to generate an accurate and consistent model of the world, which we call the digital twin. The digital twin includes position, velocity, type, and a unique identifier for every observed vehicle. By providing this digital twin to an autonomous driving research vehicle, we demonstrate that it can be used to extend the limits of the vehicle's perception far beyond its onboard sensors. Furthermore, this digital twin serves as the input for the trajectory prediction methods we presented in the previous chapters.

For autonomous vehicles to trust the digital twin for maneuver planning, its accuracy and reliability must be known. However, a thorough evaluation requires precise ground truth about the traffic situation. This is non-trivial to obtain. To solve this issue, we took aerial images of the traffic in our testbed to generate an approximate ground truth and use it to evaluate our system. We describe in detail the methods used for this evaluation, present the results, and analyze the system's performance in real-world applications. As our evaluation methodology is not specific to our system, it can serve as a general framework for the evaluation of IISs.

5.2 Related Work

First ideas for assisting vehicles, as well as monitoring and controlling traffic with an IIS have already been developed in the *PATH* [110] and *PROMETHEUS* [2] projects. Recently, with the growing efforts of industry and research to realize autonomous driving, the need for IISs that can support autonomous vehicles has further increased. Several new projects have therefore been initiated to develop and research prototypical IISs. However, their focuses differ widely, and few detailed system descriptions are available.

Communication. Some IISs projects primarily focus on the communication aspects between the vehicle and infrastructure, and sometimes additionally vehicle-to-vehicle communication. The research project *DIGINETPS* [111] focuses in particular on the communication of traffic signal information, parking space occupancy, traffic density, and road conditions to vehicles. Similarly, the *Veronika* [112] project provides traffic signal information to vehicles, intending to reduce emissions and energy consumption. The *Antwerp Smart Highway* [113] is built along a 4 km highway strip and equipped with roadside communication units on gantry bridges. In contrast to our work, its research focuses on vehicle-to-everything communication and distributed edge computing. Similarly, the goal of the *New York City Connected Vehicle Project* [114] is to improve safety and reduce the number of crashes by providing drivers with alerts via Dedicated Short-Range Communication. The *M-City* [115] project built an artificial test facility to evaluate the performance of connected and automated vehicles. Research that uses this test facility primarily focuses on communication and partially covers roadside perception.

Roadside Perception. The primary goal of roadside perception systems is the enhancement of autonomous vehicle safety. The system in the *Test Area Autonomous Driving Baden-Württemberg* [116] is perceiving a cross-road with two cameras and creates a digital twin. It also provides functionality to evaluate autonomous driving functions in a realistic environment. However, this system is much smaller than Providentia and cannot operate at night as it only uses cameras. In the *MEC-View* project, an IIS consisting of cameras and lidars mounted on streetlights creates a real-time environment model of an urban intersection that is amongst others fused into a vehicle's onboard perception system [117]. Furthermore, the local highway operator in Austria is transforming its existing road operator system into an IIS [118]. It aims to actively support autonomous vehicles and to enable the validation of autonomous vehicle perception.

IIS Algorithms. Instead of the IIS itself, many research contributions propose methods of making algorithmic use of the information provided by an IIS, or optimizing their function. Concerning communication networks, Jeffrey Miller [119] proposes an architecture for efficient vehicle-to-vehicle and vehicle-to-infrastructure communication, while Igor Kabashkin [120] analyses the reliability of bidirectional vehicle-to-infrastructure communication. In the project *KoRA9* [121], Geissler et al. [122] formulate an optimization problem that maximizes sensor coverage to locate suitable sensor placements in an IIS. Popular areas of research in the field of computer vision that are related to IIS

include traffic density prediction [123], [124] and vehicle re-identification [125], [126]. Other topics involving information provided by an IIS include danger recognition [127] and vehicle motion prediction [128], [129].

In this chapter, we focus on the overall system architecture and implementation of a large-scale IIS that generates a digital twin of the current traffic. Our system aims to complete and extend a vehicle's perception and provide information that enables the implementation of various applications. In particular, we intend to use its digital twin for our motion prediction. In the literature – to the best of our knowledge – detailed technical descriptions of systems with similar size and capabilities like ours are not publicly available.

Furthermore, to ensure our system's performance is suited for the intended purposes, we conduct a thorough evaluation by considering the overall traffic rather than trajectories from a single test vehicle. Thereby we account for various vehicle types, colors, and driving behaviors. In particular, we evaluate the spatial accuracy and the detection rate, i.e., the system's performance concerning missing vehicles and false detections. To the best of our knowledge, our work is the first to provide such quantitative results on the performance of a large-scale IIS for fine-grained vehicle perception.

5.3 Providentia System

Providentia is a large-scale distributed sensor system that provides a real-time and reliable digital twin of the current road traffic at any time or day of the year for use in various applications. We built the system on the A9 highway, close to Munich. This highway consists of two physically separated driving directions, with up to 5 lanes in each direction. In this section, we describe the design of the Providentia system in detail, including its hardware, software architecture, and the algorithms used for detection, calibration, data fusion, and position refinement.

5.3.1 Hardware and Software Setup

To build Providentia, two gantry bridges – separated by a distance of approximately 440 m – have been equipped with sensors and computing hardware. Each of these gantry bridges represents one measurement point in our system, as shown in Fig. 5.1. To achieve high perception robustness, we use sensors of different measurement modalities that cover the entire stretch between our measurement points redundantly. Fig. 5.2 illustrates the overall setting of the system with the redundant coverage of the highway.

Each measurement point comprises eight sensors with two cameras and two radars per viewing direction. In each direction, one radar covers the right-hand side while the other covers the left-hand side of the highway. The cameras have focal lengths of 16 mm and 50 mm to enable them to capture both the far and near ranges while covering the entire width of the highway. Our system can operate in varying traffic, light, and weather conditions by combining sensors with different measuring principles. Besides having redundant coverage with the sensors on each measurement point, we

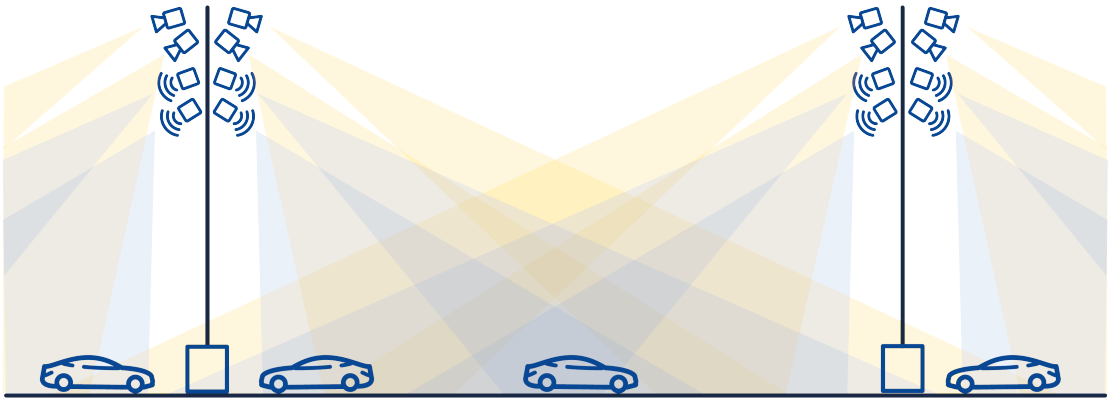


Figure 5.2: Schematic illustration of the Providentia sensor setup, with overlapping FoVs for redundancy.

also selected the positions of the two measurement points such that their total FoVs overlap as well. This further increases redundancy and thus robustness and allows smooth transitions while tracking vehicles as they move from one measurement point to another. In addition, covering the highway stretch from different viewing directions helps to resolve detection errors and occlusions.

The system employs specialized 24 GHz traffic monitoring radars from SmartMicro, of the generation UMRR-0C, with a type 40 antenna. They provide detections at an average frequency of 13.2 Hz. They are designed explicitly for stationary traffic monitoring and have a good object separation capability, even of closely spaced objects. Furthermore, they have a high detection range of up to 350 m − 450 m, depending on the object size and driving direction. Each radar covers up to 256 objects and seven lanes for the side of the highway it specializes on. All of these properties are necessary for traffic detection on high throughput highways.

The cameras we use are Basler acA1920-50gc, taking color images at an average frequency of 25 Hz. After testing various other cameras, we selected this model primarily because it can provide raw images with a very short processing time and latency, which is necessary for creating a real-time digital twin. The raw images allow us to define the image compression level ourselves, such that artifacts are minimized, and our detection algorithms become as accurate as possible.

All the sensors at a single measurement point are connected to a Data Fusion Unit (DFU), which serves as a local edge computing unit and runs with Ubuntu 16.04 Server. It is equipped with two INTEL Xeon E5-2630v4 2.2 GHz CPUs with 64 GB RAM and two NVIDIA Tesla V100 SXM2 GPUs. All sensor measurements from the cameras and radars are fed into our detection and data fusion toolchain running on this edge computing unit. This results in object lists containing all the road users tracked in the FoV of that measurement point. Each DFU transmits this object list to a backend machine via a fiber-optic network, where they are finally fused into the digital twin that covers the entire observed highway stretch.

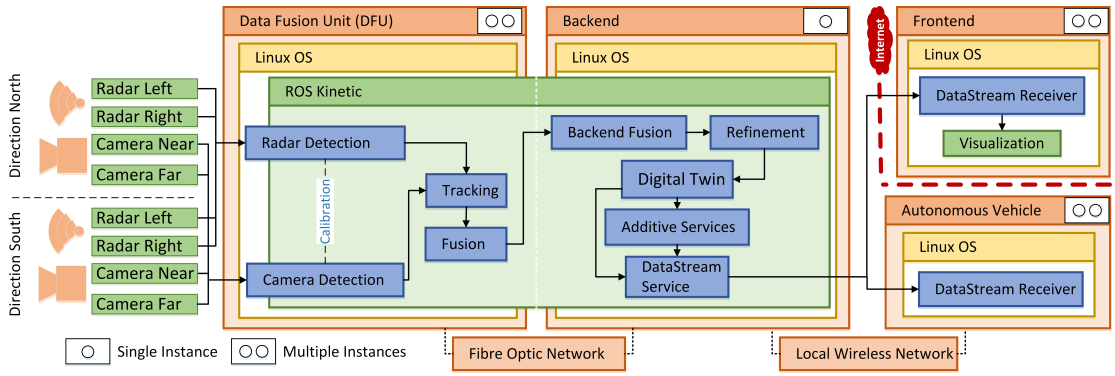


Figure 5.3: Platform architecture of the Providentia system.

The full architecture is shown in Fig. 5.3. We use the Robot Operating System (ROS) [130] on all computing units to ensure seamless connectivity. The final digital twin is communicated either to autonomous vehicles or to a frontend, where it can be visualized as required for drivers or operators.

5.3.2 Object Detection

The first step towards creating the digital twin is to detect and classify the vehicles on the highway. Our traffic radars are ex-factory capable of detecting objects and providing time-stamped positions and velocities in their respective local sensor coordinate systems on street level. We transform this output of each radar into our system's global Cartesian coordinate system using the radar calibration parameters in preparation for data fusion.

Detection and classification of objects in the camera images are performed by the DFU edge devices next to the highway. The system's cameras publish time-stamped images that are tagged with a unique camera identifier. To ensure scalability and safety, even in the event of camera failures, our modular object detection pipelines subscribe to each image stream separately. Furthermore, the object detection pipelines are constantly monitored to supervise and analyze detection performance. The modular services are automatically restarted in the event of any failures. All camera streams are processed in parallel, and our object detection can work with various detection networks. This flexibility allows us to configure the object detection to optimally balance between low computation time and high accuracy, depending on the requirements that the application of our system poses.

At the time of writing, we use the YOLOv4 [131] architecture as our detection network in the object detection pipelines. In addition to regressing two-dimensional bounding boxes with a confidence score, this network classifies the detected vehicles into car, truck, bus, or motorcycle types. Our detection modules then publishes the captured object detections in our system.

To compute three-dimensional positions for the vehicles from the camera detections in the images, it is unfavorable to use stereo vision techniques in our system. Our

cameras that look in the same direction are placed closely together, significantly differ in their focal length, and their FoVs overlap only partially. Instead, we use the vehicles' bounding boxes to cast a ray through their lower-edge midpoint and intersect it with the street-level ground plane that we know from our camera calibration. We transform the resulting vehicle positions into our system's global Cartesian coordinate system in the same manner as the detections of the radars. The resulting measurements are then ready to be fused into a consistent world model and fed into the data fusion pipeline, starting with a tracking module.

5.3.3 Calibration

Precise calibration of each sensor and measurement point is necessary to transform all sensor measurements from their respective local system to a common global coordinate system. Only then can we perform data fusion and ultimately generate the digital twin. As the first step, we intrinsically calibrated all cameras individually prior to their installation using a common checkerboard calibration target and the camera calibration package provided by ROS. In particular, the function we used is minimizing reprojection errors with the Levenberg-Marquardt optimization algorithm. During the build-up of the system, all radars were calibrated with their ex-factory supplied calibration software. This software assists in ensuring that each radar is mounted with an ideal orientation at its perfect operating point, such that the radar optimally covers the part of the highway it specializes on. To determine these parameters, the software requires the radar's mounting height and a local map as input. As a result, the radar is enabled to transform all measurements internally and to output them in a Cartesian coordinate system with its origin placed directly underneath the radar on street level. The XY-plane of this coordinate system approximates the street.

The overall extrinsic calibration of the system after having installed all sensors is non-trivial. Not only does our system possess a high number of sensors and degrees of freedom, but it also makes use of sensors with heterogeneous measurement principles. Once we calibrate our system and know all the sensors' positions and orientations on each gantry bridge, we can transform their measurements into a common global coordinate system. Our system can then provide the digital twin in a standardized reference frame to the outside world, as we know the measurement points' orientation towards north and the Global Positioning System (GPS) coordinates of reference points on the gantry bridges from official surveying and an HD map.

To obtain approximate starting points for each sensor for our extrinsic calibration algorithms, we manually measure the relative translation of all sensors and the reference point on each gantry bridge separately using modern laser distance meters. Thereby, we measure orthogonal respectively parallel to the gantry bridge and the street plane for reference. Note that the gantry bridges are accessible with a walking corridor on top (see Fig. 5.1), which allows taking manual measurements. While the radars' orientations are already approximately known from set-up, we determine all cameras' yaw angles relative to the highway direction with a compass. Then we measure their pitch and roll

(a) Initial Calibration **(b)** Improved Calibration

Figure 5.4: After our initial calibration with physical measurements (a), we refine the inter-sensor calibration by projecting the radar detections in the camera image and optimize the alignment with the observed vehicles (b).

angles relative to the horizontal street plane with a digital angle finder with spirit levels.

We then refine the camera poses using vanishing point methods that utilize parallel road markings to find vanishing points [132]. To fine-tune the inter-sensor calibration, we manually minimize the projection and re-projection errors for all sensor pairs, both in the image planes (see Fig. 5.4) and on street level in the three-dimensional coordinate system. Furthermore, we incorporate the lane information from the HD map as an additional reference. The final calibration step is to refine the alignment of the measurement points with each other. By transforming the detections of both measurement points into the same coordinate system, we can manually associate them and minimize their distance to find an optimal overall calibration. This results in a global coordinate system for our IIS, where all sensor detections can be transformed into. In this coordinate system, the final digital twin is created and then transformed to GPS.

5.3.4 Data Fusion

A large-scale sensor system like ours poses many challenges when it comes to sensor data fusion. On the highway, we potentially observe a large number of vehicles that need to be tracked in real-time. Therefore, the data fusion system has to scale for hundreds of vehicles. In addition, the number of targets is unknown in advance, and our fusion must be robust to clutter and detection failures. Conventional filtering methods that handle each observed vehicle separately, such as multiple Kalman filters or multiple hypotheses tracking [133], require to explicitly solve a complex association problem between the system's sensor detections and tracked vehicles. This severely limits scalability. For this reason, we use the Random Finite Set (RFS) framework [134], [135], specifically the Gaussian Mixture Probability Hypothesis Density (GM-PHD) filter [136]. This filter avoids the explicit data association step and has proven to balance our runtime and scalability constraints well. Additionally, it handles time-varying target numbers, clutter, and detection uncertainty within the filtering recursion.

To add tracking capabilities to our GM-PHD filter, we extended it with ideas taken

from Panta et al. [137]. In particular, we make use of the tree structure that naturally arises in the GM-PHD filter recursion and implement appropriate track management methods. With the resulting tracker, we track the measurements of each sensor in parallel. As our sensor model, we use a zero-mean Gaussian white noise observation model, and as our motion model, a vanilla Constant Velocity Model (CVM). As we have shown in Sec. 3.3 – despite being simple – the constant velocity model is a competitive prediction model while being very fast to evaluate. All parameters for our sensor and scenario specifications were tuned empirically.

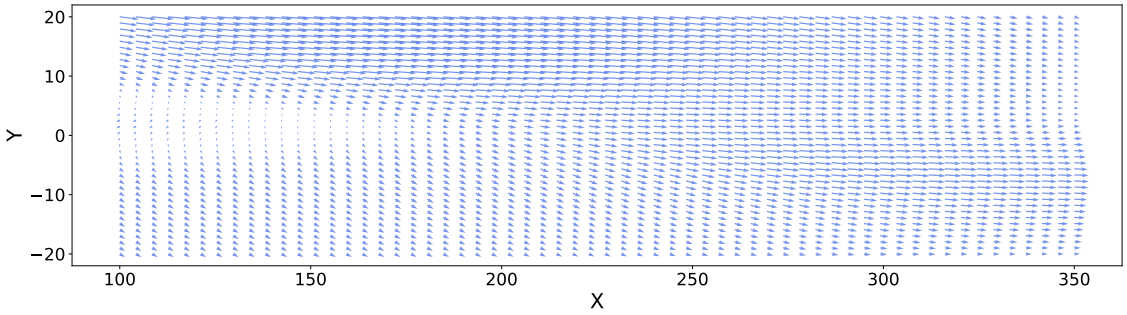
To fuse the tracked data from different sensors and measurement points, we adapt the method from Vasic et al. [138] that is based on generalized covariance intersection [139]. In order to ensure scalability and easy extension of our system setup, we implement a hierarchical data fusion concept, in which we first perform independent local sensor fusion at each measurement point leading to vehicle tracklets. Second-level fusion of all measurement points is then performed on the backend. This step generates the consistent model of the highway scene covered by our system, which we refer to as the digital twin.

Switching between different fusion setups is possible, depending on the sensors that should be used. With this, the system can be adapted to changing lighting conditions, where the different sensor types complement each other to varying extents. Apart from fusing all sensors, our system also allows to only fuse the cameras or to only fuse the radars. For example, at night, it switches to only using the radars.

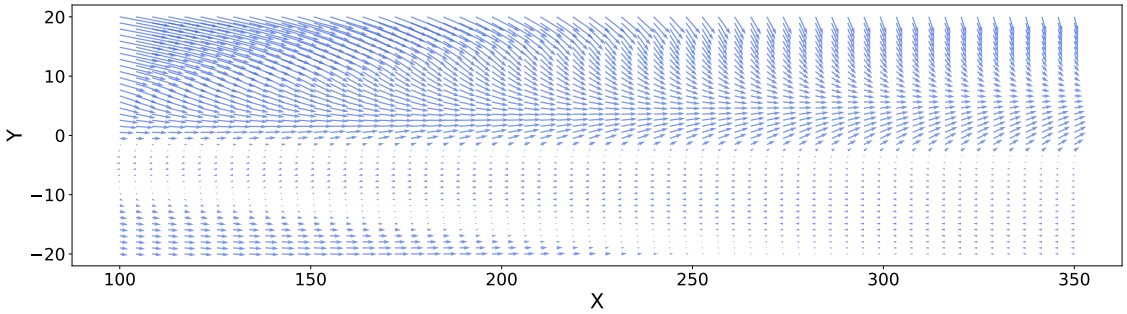
5.3.5 Position Refinement

After the data fusion, the detected vehicle positions in the digital twin tend to be placed either towards the vehicles' front or rear. The reason for this is the way we compute positions in world coordinates from two-dimensional bounding box detections by ray casting (see Sec. 5.3.2). The exact placement is dependent on the cameras' perspectives. Other systematic positioning errors our system is affected by are calibration inaccuracies and approximation errors of the street geometry.

To address these problems jointly, we apply a regression-based position refinement using a Feed-Forward Neural Network (FNN) after our data fusion. It receives the position of a vehicle in the digital twin and predicts the systematic offset to its actual center position based on its current location on the highway. Adding this offset to the input position, we estimate the vehicle's true position more accurately and reduce the systematic positioning errors described. We have trained the neural network using a random split of our ground truth data, which we associate with vehicle detections from our fusion to obtain a training dataset (see Sec. 5.5.1 and Sec. 5.5.2 for more details). We train one network for operation during the day and one for the night. The networks share the same architecture and have five hidden layers with 32 neurons each. To train them, we use the the Adam optimizer [17], batch size 16, learning rate 0.001, and 0.1 weighted l^2 regularization for 1000 epochs.



(a) Refinements at day

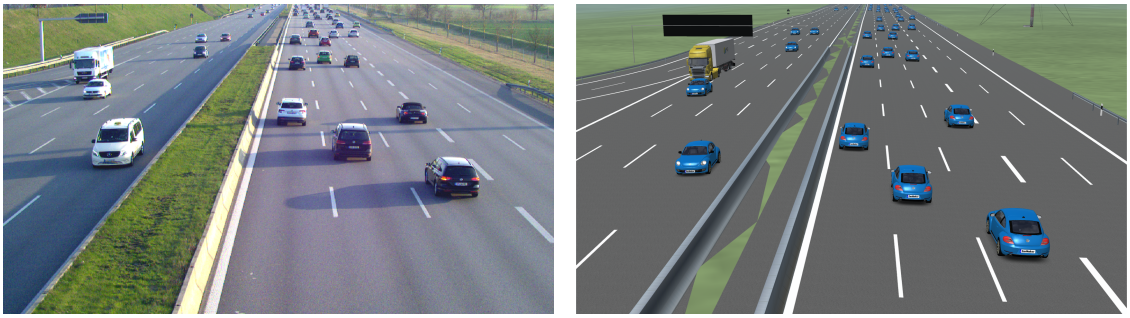


(b) Refinements at night

Figure 5.5: Our refinement module learns a displacement correction for each position of the road to eliminate systematic positioning biases. The images show the corrective vector fields our models learned for day and night.

To demonstrate what these refinement networks learn, we sampled a grid of positions over our testbed and evaluated each position with our trained networks for day and night, respectively. Fig. 5.5a shows the refinement vectors for our system at day, when we use cameras. The strongest corrections happen towards the top lanes of the highway. This effect can be explained by the placement of our cameras around the -5 mark on the Y axis (see Fig. 5.6a). Hence, their perspective on the vehicles that drive on the opposite side of the highway is very oblique, and determining these vehicles' positions with the ray casting we described in Sec. 5.3.2 leads to systematic errors in the position estimates.

Our system's radars are placed similarly and perform best when detecting positions of vehicles that drive closely to their viewing direction. Therefore, they are prone to errors when detecting vehicles that drive laterally displaced, i.e., also on the opposite highway side of where our sensors are installed (see Fig. 5.5b). However, as we explained earlier, not all corrections can be explained by the sensors' perspectives only, but also calibration inaccuracies and approximation errors of the road geometry affect systematic displacements in our system.



(a) Camera Image **(b)** Digital Twin

Figure 5.6: Qualitative example of how our system captures the real world (a) in a digital twin (b). We recreate the scene with generalized models for different vehicle types. During operation, all information is sent to the autonomous vehicle in the form of a sparse object list.

5.4 Digital Twin

The digital twin represents the main output of the Providentia system. It consists of the position, velocity, and type of every vehicle observed, with each one assigned a unique tracking identifier. The contained positions can be transformed to GPS coordinates, depending on the application to which the digital twin is sent. It can be used by vehicles on the highway stretch to improve their decision-making and to implement additional services that the infrastructure itself can provide. Such services might include motion prediction for each vehicle, congestion recognition, lane recommendations, and collision warnings. This section illustrates our system's ability to capture the traffic and demonstrates its potential for extending an autonomous vehicle's perception of the scene.

Our qualitative examples were captured in our testbed under real-world conditions. Currently, our system redundantly covers a stretch of about 440 m of the road, which corresponds to the distance between the two measurement points. Fig. 5.6 shows an example of a digital twin that our system created of traffic on the highway. It is a visualization of the information sent to autonomous vehicles to extend their perception. Our system manages to detect the vehicles passing through the testbed reliably. This is only possible by fusing multiple sensor perspectives. The update rate for the digital twin depends on the fusion setup and the type of the used object detection network. It varies between 13.1 Hz when only using the radars at night and 24.6 Hz when using the cameras with the YOLOv4 architecture at day.

We also transmitted our system's digital twin to our autonomous driving research vehicle *fortuna* [140] to extend its environmental perception and situation understanding. Vehicles perceive their environment through lidars, which have limited measurement ranges, and the point cloud density in the distance becomes increasingly sparse. Vehicular cameras can capture a more distant environment than lidars, but objects that are too far away appear small on the image and cannot be reliably detected. Furthermore,

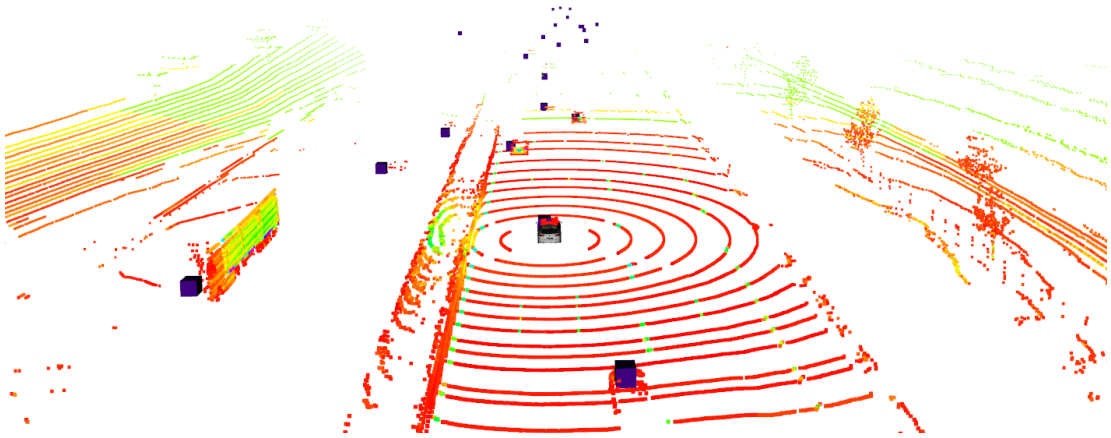


Figure 5.7: An autonomous driving research vehicle driving through our testbed. The dots visualize the vehicle's lidars measurements, and the purple cubes represent the vehicles perceived by the Providentia system. While the vehicle's own lidar range is severely limited, its perception and resulting scene understanding are extended into the far distance using information from our system.

the vehicle's low perspective is prone to severe occlusions. Fig. 5.7 illustrates how an autonomous vehicle driving through our system perceives its environment. The violet cubes represent vehicles detected by our system. At a distance of approximately 80 m the point cloud density of our vehicle's lidars drops significantly. However, our system's digital twin extends the vehicle's environmental perception to up to 400 m. In principle, a system such as ours can extend the perception of a vehicle even further since we designed it with scalability in mind. The maximum distance is only limited by the existing number of built-up measurement points.

5.5 System Performance Evaluation

Knowing the accuracy and detection rate of our system's digital twin is crucial to decide what applications can be realized with it. For example, using the digital twin for maneuver planning in autonomous vehicles or predicting future trajectories requires high position accuracy. In contrast, position accuracy is less important for detecting traffic jams. Knowing the statistical certainty and uncertainty of the system's measurements also makes it possible to define safety margins that vehicles have to consider when using the provided information.

However, evaluating the system's digital twin is a challenging task. To merely evaluate the detection performance of individual sensors is insufficient to judge the system's performance, as the calibration between the sensors and the fusion algorithms are of paramount importance for the quality of the digital twin and must therefore also be included in the evaluation. End-to-end evaluation of the system requires ground truth information of the traffic on the testbed over an extended period. This implies having

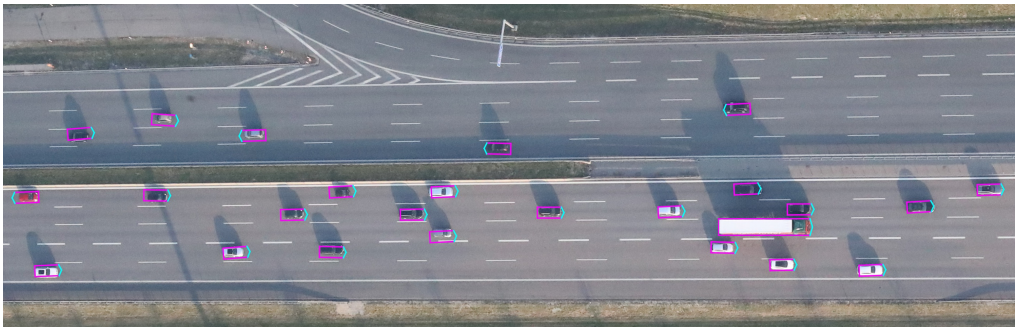


Figure 5.8: Crop of an aerial image including vehicle detections, taken with the helicopter's left-hand camera that captures part of our testbed.

the exact positions of all the vehicles on the observed stretch of the highway. Labeling the images from the cameras within the system is not sufficient, as it would only provide ground truth information in image coordinates but not in the real world. Using a single localized test vehicle also has limits, as the system must handle a wide variety of vehicle colors and shapes in practice. Furthermore, the usefulness of simulations is limited as well. In reality, the system is subject not only to various lighting and vibration effects but also to drivers' decisions, which are hard to model.

That is why we approximate the required ground truth by recording aerial images of the testbed. These have an ideal – almost orthogonal – top-down perspective of the highway. This perspective avoids all inter-vehicle occlusions, and due to their regular contours, vehicles are easy to detect and distinguish. This section will describe how we captured and processed these images to generate ground truth data suitable for evaluating our system. We also explain the evaluation itself and discuss the results and their implications for the performance of our system.

5.5.1 Ground Truth Generation

The Providentia testbed was recorded using a 4k camera system mounted on an H135 helicopter to generate such an aerial view ground truth. Both the camera system and the Providentia system were synchronized with GPS time. The 4k camera system consists of three Canon EOS 1D-X Mark II cameras oriented in different viewing directions, each recording images at a resolution of 20.2 megapixels. The cameras to the left and right covered the northern and southern parts of the testbed, respectively, with an overlapping FoV. The third, nadir-looking camera of the system was not used. The cameras captured images simultaneously at a rate of one image per second at a flight altitude of 450 m above ground, covering an area of 600 m $\times$ 250 m. With a focal length of 50 mm, each image pixel corresponds to 6 cm on the ground. In addition, an IGI Compact MEMS GNSS/IMU system was used to estimate the position and orientation of the sensors during flight to enable georeferencing of the images captured. Bundle adjustment with tie points and ground control points was performed to optimize the georeferencing accuracy.

We used a neural network that has been trained and evaluated with the EAGLE dataset [141] for the object detection in all aerial images. To compute the positions of the detected vehicles on the road, we cast rays through all four bounding box corner positions in the aerial image and intersected them with a lidar terrain model of the highway surface to compute local Universal Transverse Mercator (UTM) coordinates. To obtain the final ground truth data that we use in our evaluation, we computed the center position of each vehicle. Furthermore, we filtered out all vehicles that were detected outside of the Providentia system's field of view or were detected twice in the overlap of the two cameras' FoVs.

The quality of the obtained ground truth depends on the accuracy of the object detections in the aerial images and the positioning accuracy. The detection quality is the subject of ongoing research [142] and is not entirely perfect yet. The network we used occasionally missed vehicles, especially trucks, slightly misplaced bounding boxes, or detected false positives. We manually corrected these errors by re-labeling the concerned bounding boxes to obtain perfect ground truth. The positioning accuracy in world coordinates depends on the georeferencing accuracy of the images. Specifically, it depends on the calibration accuracy of the 4k system, the quality of the tie and ground control points used for bundle adjustment, and the accuracy of the underlying terrain model. The overall absolute accuracy on the present dataset lies in the centimeter range. The accuracy of this georeferencing is demonstrated in [143].

We recorded the ground truth data with a medium traffic volume during the day. Fig. 5.8 shows a captured aerial image with vehicle detections. Even though the testbed was not always fully covered by the helicopter's cameras, we captured enough vehicles to perform a reliable and statistically significant evaluation. In total, we generated over 2 minutes of ground truth data containing 2125 valid vehicle observations within our testbed. Additionally, each detection contains a classification that distinguishes between cars and trucks. In total, our dataset contains 95 % cars and 5 % trucks.

5.5.2 Evaluation Methodology

To compare our digital twin with the ground truth data, we first transform the aerial detections from UTM into the Cartesian world coordinate system used by Providentia, and in which the digital twin is defined. In the next step, we match all ground truth frames, i.e., all detections in each pair of left-hand and right-hand camera images, with those frames of the digital twin with the closest corresponding timestamp. Note that the digital twin has a higher frequency than the ground truth with 1 Hz, and they are slightly offset to each other. We account for the time difference between matched frames by extrapolating the Providentia detections with a constant velocity prediction.

To assess the overall performance of our system, we evaluate the classification accuracy, spatial accuracy, and detection rate of the digital twin. For computing these metrics, it is necessary to establish associations between the detected vehicles in the digital twin and the ground truth. For this purpose, we use the Hungarian algorithm [144], which guarantees an optimal one-to-one assignment. We use a weighted Euclidean distance to

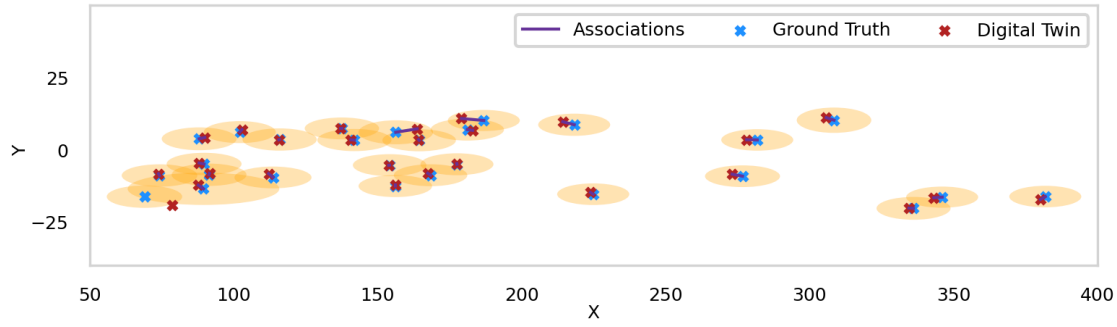


Figure 5.9: Associations between detections from our system and ground truth data. Only detections within the displayed ellipses around each vehicle in the ground truth data are associated. Actual associations are marked with a line between corresponding detections. Note that the different sizes of the ellipses depends on the ground truth vehicle sizes.

compute the cost matrix, emphasizing the longitudinal distance between vehicles rather than their lateral distance. This accounts for the fact that the estimates in the digital twin have a greater variance in the driving direction because of their shape and kinematics (see also the RMSE results in Sec. 5.5.3). While vehicles driving on nearby lanes can be close, the estimate of one should not be associated with the ground truth of another.

Furthermore, we make the weights dependent on the vehicles' dimensions because the detection of a larger vehicle is more likely to be placed further away from its center. In particular, we define the longitudinal weight as the vehicle length plus 8 m and the lateral weight as its width plus 1.7 m. We determined these additive components empirically. Thresholding this weighted distance represents an ellipse around the ground truth object, with its major axis aligned with the driving direction (see Fig. 5.9). Based on these ellipses, we perform a gating with threshold 1. By this, we reject all associations outside of the respective ellipse to avoid wrong associations between pairs of false negatives and false positives. Overall, these parameters lead to accurate and reasonable associations.

We can then compute our system's classification accuracy with the correctly established associations. This accuracy is the percentage of vehicles to which our system assigned the correct class label. We differentiate between cars and trucks as these are the class labels of the objects contained in our ground truth data. Besides an overall classification accuracy, we also report our system's classification accuracy for both classes individually. Furthermore, we also use such a class-dependent evaluation in the following metrics.

To evaluate the spatial accuracy of the digital twin, we compute the Root-mean-square Error (RMSE). It represents the standard deviation of the error between the vehicle positions in the digital twin and the ground truth positions in meters. It is a good summary measure of the positioning errors in the digital twin. In particular, for computing the RMSE, we split the established associations into a 75% test set and 25%

training set. We use the split training set for training our position refinement network (see Sec. 5.3.5) and use 2% of the training set for validation. We ensure to split the smaller subset of trucks with equal percentages. This splitting helps avoid skewed outcomes of random splitting, like that the test set contains no trucks.

In addition to the classification and spatial accuracy, the detection rate of our system is essential for evaluating its overall performance. Appropriate metrics for this are precision and recall. The precision of our system states what percentage of vehicles detected by our system were actually present. Its recall refers to the successfully detected percentage of vehicles in the testbed. To evaluate these detection metrics consistently with each other and with the RMSE computation, we compute the true positives, false positives, and false negatives based on all established associations. In particular, we first associate all ground truth detections with the digital twin. Then we determine the FoV of our Providentia system and count all associated ground truth vehicles within this FoV as true positives. Those ground truth detections within this FoV that got not associated are false negatives. To compute the false positives, we must consider that the helicopter was moving and occasionally only covered parts of our testbed. This could lead to counting correct vehicle detections in our digital twin as false positives because the ground truth did not capture them. To avoid this, we project the current camera FoV of the ground truth data for each frame on the Providentia testbed and intersect it with the Providentia FoV. Then, we take all Providentia detections within this resulting intersected FoV that have not been associated with a ground truth detection as false positives.

We evaluate the performance of the Providentia system at the day and night by evaluating the digital twin it creates with either only using camera detections as inputs or with only using radar detections, respectively. Using only radars is a valid method for simulating night measurements, since radar performance is independent of lighting conditions. Hence, the radar-only based digital twin performs the same way day and night, given the same traffic. Like this, we can use the same traffic scenes that we recorded during the day for our day and our night evaluation. In both types of evaluations, we consider the area enclosed by the two measurement points that we cover redundantly.

5.5.3 Results

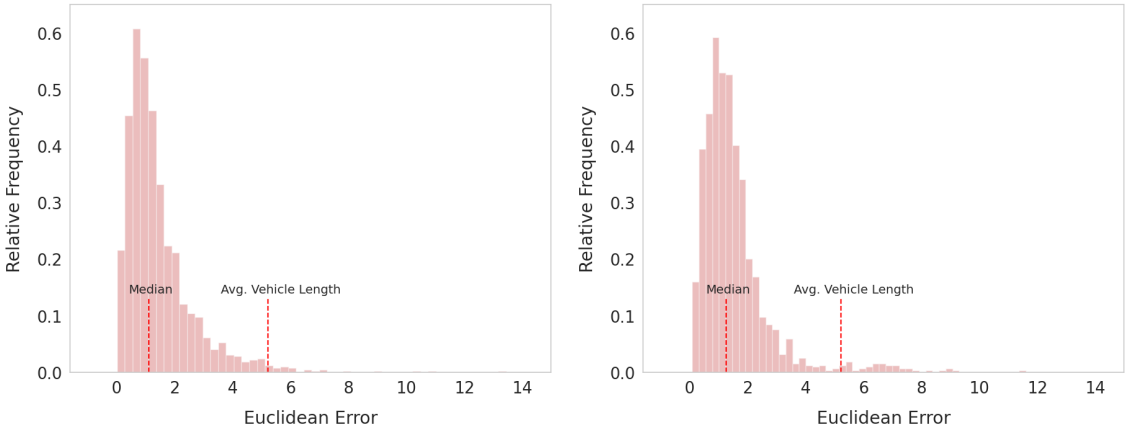
All results of our evaluation are summarized in Tab. 5.1, separated by day and night as well as by vehicle classes. In the following, we discuss the performance of our system for all evaluated metrics in detail.

Classification Accuracy

Our system classifies 96.2 % of the vehicles correctly during the day and 95.6 % during the night. At day, trucks are perfectly classified, while the accuracy for cars is 96.0 %. Most classification errors stem from vans that were assigned to the car class in our

Table 5.1: Evaluation results of the Providentia digital twin

Mode	Class	Classification	RMSE	$RMSE_x$	$RMSE_y$	Precision	Recall
Day	**Total**	96.2 %	1.88 m	1.81 m	0.49 m	99.5 %	98.4 %
	Car	96.0 %	1.79 m	1.72 m	0.48 m	99.6 %	98.5 %
	Truck	100.0 %	3.12 m	3.05 m	0.66 m	97.2 %	96.4 %
Night	**Total**	95.6 %	2.00 m	1.82 m	0.83 m	99.0 %	94.1 %
	Car	98.2 %	1.54 m	1.31 m	0.81 m	99.0 %	93.9 %
	Truck	50.5 %	5.64 m	5.55 m	1.01 m	98.0 %	97.1 %



(a) Errors at Day **(b)** Errors at Night

Figure 5.10: Distribution of positioning errors in the digital twin. Our system is very accurate in most cases. The errors are within the average vehicle length in 98.0 % of the cases at day and in 99.6 % of the cases at night. In 50 % of the cases the error is less than 1.02 m at day and less than 1.26 m at night.

ground truth but get easily confused with trucks by our camera detection network. When such misclassifications in the camera detections happen consistently to a vehicle, our tracker cannot compensate for the errors. However, these errors could be reduced by retraining our detection network with an additional van class, such that it learns to differentiate better. At night, using our radars, our system has a high classification accuracy of 98.2 % for cars but struggles at classifying trucks. They are mistaken for cars half of the time. However, this is expected because the classification of the radars is only based on reflection characteristics and rough length estimates, but no visual clues can be used.

Spatial Accuracy

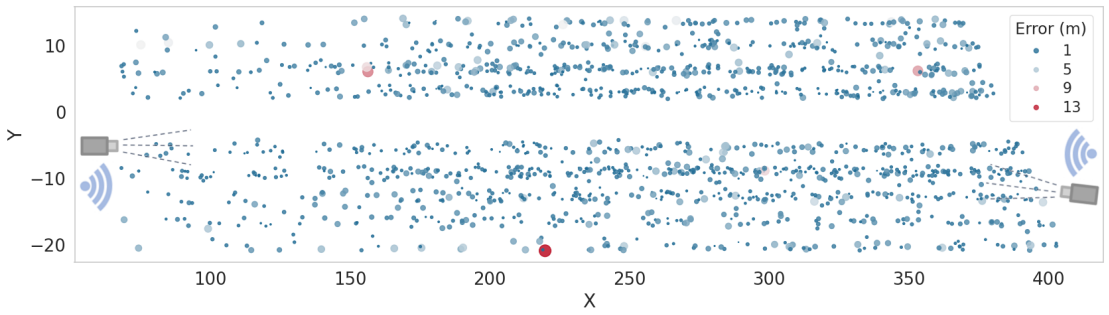
Concerning the spatial accuracy of the digital twin, we achieve an overall RMSE of 1.88 m during the day and 2.00 m at night (see Tab. 5.1). In both cases, the major component of the RMSE stems from the longitudinal direction and is 1.82 m and 1.81 m, respectively. In the lateral direction, our system is very precise with an error of only 0.49 m during the day. At night, this error component increases to 0.83 m because the radars' lateral position estimates are subject to greater noise, especially at larger distances. In both cases, the high lateral accuracy allows us to determine the lane of each vehicle reliably.

A significant component of both the longitudinal and lateral positioning errors is due to the current lack of information about the actual extents of the objects in our system. Because our camera detections are two-dimensional bounding boxes in the image plane, the vehicles' lengths are not explicitly considered for estimating their position, and their widths can only be approximated with perspective errors. The radars provide estimates of the vehicles' centers that result from the detection point, which is corrected by average vehicle extents. These average extents are associated with the corresponding vehicle class, which is inferred from its reflection characteristics. We partially compensate this source of systematic errors for both sensor types with our position refinement regression but cannot entirely remove it without knowing the vehicles' exact extents. Therefore, our system has difficulties handling especially those vehicles which extents strongly deviate from the average of their respective class. As a result, our system has a bias and places detections either more towards the rears or fronts of the vehicles as it should, depending on the perspective. However, in the ground truth, the position of an object is specified at its center. As the ground truth vehicles driving through the testbed throughout our evaluation have an average length of approximately 5.18 m, the extreme placement of a detection at the rear or front would already cause a displacement of 2.6 m to the center. Given this placement issue, our spatial accuracy is promising both day and night.

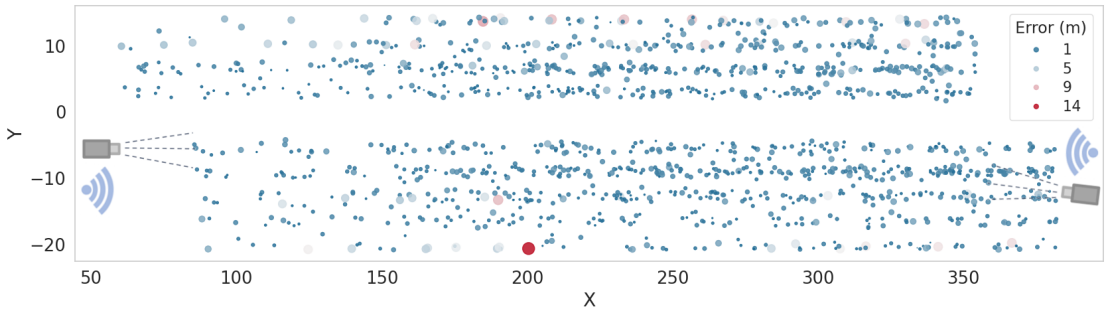
Fig. 5.10 shows the distributions of our overall positioning errors from which we computed the RMSE. During the day, in 50 % of the cases, our error is less than 1.10 m and at night less than 1.23 m. Furthermore, in 98.0 % of cases at day and 99.6 % at night, the errors of our detections are within the average vehicle length. For trucks, the error is within the average length in all cases.

In general, by separating the positioning errors for both vehicle classes, in Tab. 5.1 it can be seen that the RMSE for cars is smaller than for trucks during the day and at night. Based on our previous analysis, this is expected since trucks have greater extents which leads to larger estimation errors for their centers. Our system performs significantly better during the day than at night for trucks. This result can be explained by the radar's high misclassification rate for trucks. A consistently false classification of a truck as a car leads to an underestimation of the vehicle's center offset from the detection point. For cars, our system performs slightly better at night because of smaller longitudinal errors in the radar detections and the radars' accurate center corrections.

To see how the positioning errors are distributed over our testbed, in Fig. 5.11 we plotted the error for each corresponding ground truth detection. At day and night,



(a) Errors at Day



(b) Errors at Night

Figure 5.11: Positioning errors in the digital twin for each ground truth vehicle on the highway, along with the schematic measurement point positions. The FoV differences between day and night result from the changing sensor setups. During day and night severe errors are rare and mostly due to oblique perspectives and greater vehicle extents.

significant positioning errors over 5 m are rare and often belong to large vehicles. Additionally, larger positioning errors mainly occur on the top and bottom lanes. There, the sensors have more oblique perspectives on the vehicles. Hence, ray casting through the lower-edge midpoints of the bounding boxes deviates more from the actual middle of the vehicles in the camera detections. The radars are more accurate at measuring the positions of vehicles that drive closely to their viewing direction, as their angular resolution deteriorates towards larger angles. Furthermore, the road is only approximately a plane and deviates the most towards the outer lanes. Hence, there the projection errors for both the camera detections and the radar detections are the greatest.

Our position refinement module corrects a large number of systematic positioning errors. Even though small errors caused by oblique camera perspectives remain, they were significantly reduced. Without the regression, we also had an accumulation of errors for the camera positioning towards the middle of the testbed. In this area, the vehicles are far from all cameras, and their resolution in the images is the smallest, resulting in higher uncertainty in the bounding box estimates. The way we compute the vehicle positions from the camera detections, i.e., by intersecting rays with the road (see Sec. 5.3.2), is sensitive to such inaccuracies over large distances because the rays intersect the ground plane at a flatter angle. Our radars had the largest errors for

the vehicles on lanes of the opposite driving direction of where they were installed, which the regression also addressed well. In total, the position refinement module reduced the RMSE during the day from 3.47 m to 1.88 m and at night from 2.64 m to 2.00 m. While the data-driven correction significantly affects errors both day and night, the camera detections benefit stronger from it. This shows that the cameras suffer more from systematic errors, e.g., caused by unfavorable perspectives in some road areas. Furthermore, the smaller error reduction for the radars can be explained by their correction of the center offset with average vehicle extents, and it shows the need for this correction in the camera detections. Despite the significant improvements, some random error sources could not be fully corrected in both the data fusion and regression, e.g., measurement noise and sensor vibrations that make the detections susceptible to calibration inaccuracies.

Detection Rate

Evaluating the detection rate, our system achieves an overall precision of 99.5 % during the day and 99.0 % at night (see Tab. 5.1). This means that we have very few false positives, and almost all vehicles detected by our system exist and are correct. The few false positives during the night can be explained with the radars tending to split larger trucks in two detections that are classified as truck or car. In most cases, this splitting is compensated by our tracker, but it is not always possible. Quite similarly, larger trucks are sometimes split into towing vehicles and trailers by the camera detections, explaining the slightly lower precision for the truck class. As for the recall, our system achieves 98.4 % during the day. Hence, we detect 98.4 % of all ground truth vehicles on the highway and only miss 1.6 % of them. At night, our recall is 94.1 % which is not as high as during the day, but the vast majority of the vehicles are still detected. This decrease can be better explained when differentiating between cars and trucks. The recall for trucks with 97.1 % compared to 93.9 % for cars is significantly higher. Trucks have a larger surface to reflect radar signals, and thus the likelihood of missing a truck is smaller than for a car. During the day, our system is slightly more accurate at detecting cars, most likely, because cars are more frequent than trucks in the training data of our object detection networks.

It is important to note that we analyzed the precision and recall of our system on a frame-by-frame basis. Hence, when we do not detect a vehicle, it does not imply that it is passing through the testbed completely undetected. This did not happen. Instead, certain vehicles may be briefly lost at specific moments in time due to temporary detection failure or occlusions caused by larger vehicles. This indicates that incorporating an occlusion handling mechanism in the tracking can further optimize the overall performance.

In general, our system achieves a high degree of reliability in terms of spatial accuracy, detection rate, and classification accuracy. The accurate positioning of our system also allows us to estimate the vehicles' motion directions and speeds with high precision. Our results further show that it is highly beneficial to use cameras during the day instead of a radar-only system. Their detection rate and classification accuracy are higher than that

of radars. Furthermore, their update frequency is almost twice as high, and their spatial accuracy is similar. However, also our radars show good performance, which makes our system reliable even at night. Hence, both sensor types complement each other and enable our system to run at any time of the day. By incorporating methods to determine spatial vehicle extents, our system's positioning accuracy could be further improved, and a more balanced and larger training set could lead to even higher detection and classification performance, thus further increasing the system's reliability.

5.6 Lessons Learned

Only technological advances in recent years made it possible to develop a system like Providentia. This especially concerns computing power, artificial intelligence, and data fusion algorithms. However, despite these advances, it is a complex task to build a functional IIS for fine-grained vehicle perception such as our system.

Before building up the actual IIS, we recommend conducting many different field tests to gather a good understanding of the challenges involved. These studies are necessary due to the diverse nature of problems posed by every region or road. To name a few examples, selecting appropriate sensors and their mounting locations heavily depends on the road segment's length, its curvature, and the direction of observation. For our research testbed, one important aspect was to ensure that it is as diverse as possible. To achieve this, we chose a highway section at one of Germany's traffic hot spots that leads into a highway interchange and, in addition, is equipped with ramps that lead towards a close-by city. Therefore, many interesting driving maneuvers take place, e.g., various lane changes, vehicle interactions, and sometimes even accidents. Moreover, our testbed is exposed to various traffic conditions, from light traffic to heavy traffic jams. Nevertheless, mixtures of light traffic and traffic jams also occur. For example, when the two lanes that are branching off towards the intersecting highway get congested, vehicles may still pass at high speeds on the inner lanes.

Based on our experiences, building up the hardware for such a system is technically demanding and requires a team with a broad skillset. It was necessary to design tailored brackets to equip the gantry bridges with our sensors, deploy the sensors and computing units in a weather-resistant manner, install cabling to get a high-speed internet connection at the highway, and answer various legal and safety questions. However, these efforts and costs could be significantly reduced in the future as we perceive the ongoing trend to technologically modernize roads and highways to observe and actively optimize traffic in many countries. This modernization could synergize with systems like Providentia.

After the initial construction, our system has been running for over two years at the time of writing, and its hardware required only minor maintenance. What is more difficult is the system's calibration because the high number of multimodal sensors results in many degrees of freedom. Furthermore, the system gradually decalibrates over time, even after a precise initial calibration. Causes of this decalibration are primarily

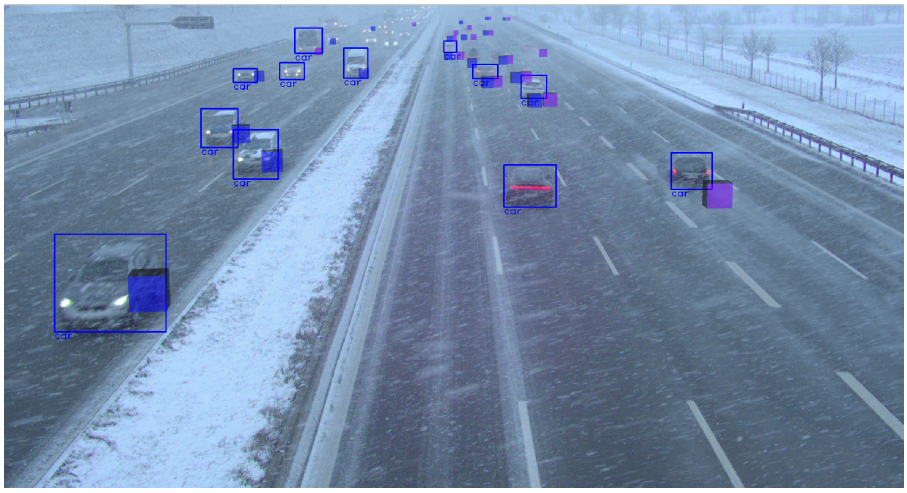


Figure 5.12: Detections of our system in a blizzard. Camera detections are marked with bounding boxes and the detections of our radars with cubes. Our radars detect distant vehicles more reliably under these weather conditions.

temperature changes, oscillations of the measurement points due to wind, and vibrations caused by large passing vehicles. These effects slightly change the sensors' positions relative to the road over time. Hence, the calibration must be regularly adjusted to avoid performance deterioration. In practice, during our project, we recalibrated our system when inaccuracies became visually apparent. Even though these are only slight readjustments once the system has been thoroughly calibrated initially, we recommend using suitable auto-calibration methods to reduce maintenance efforts for future applications. As the oscillations also cause the gantry bridges to slightly swing around their equilibrium, auto-calibration methods that run online and compensate for all immediate oscillations would further improve the system's performance and reliability in the future.

During our research project, we were able to assess the performance of our system in harsh weather qualitatively. As shown in Fig. 5.12, even in snowstorms, our system can detect the vehicles passing through our testbed reliably. To quantitatively evaluate the system's performance under such conditions and identify cross-over points at which it is ideal to switch between different detection and fusion modes, the development of evaluation methods that do not rely on aerial observation is required. Furthermore, self-diagnosis could instantly detect sensor failures or faults within the fusion system, e.g., caused by a deteriorated calibration. For this purpose, approaches like the one from Geissler et al. [145] could be applied.

Regarding its extension, we have built our system such that it is scalable (see Sec. 5.3). Many of our developed concepts and algorithms can be transferred when increasing the number of measurement points. However, one crucial aspect of this scalability is the system's middleware. This middleware should be given considerable thought before building a more extensive system than ours. In our system, we used the ROS. Due to its convenience, it enabled short development time, but we also experienced problems using

it, for example, message delays and network bottlenecks. Typical sources of bottlenecks in ROS are the transfer of images through TCP based serialized transport. This may affect the real-time performance of a scaled system and render ROS inappropriate. Another downside is the missing backward compatibility of ROS messages. Once data has been recorded in an old message format, there are problems replaying it with adapted new messages.

Besides a fast middleware, the quality of the entire system depends upon precise knowledge about when various events occur. We found an appropriate strategy for time synchronization based on a dependable master clock to be essential. It could be beneficial to choose sensors that support synchronization with such an external master clock in a new system.

5.7 Conclusion

To improve the safety and comfort of autonomous vehicles, one should not rely solely on onboard sensors, but their perception and scene understanding should be extended by adding information available from a modern IIS. With its superior sensor perspectives and spatial distribution, an IIS can provide information far beyond the perception range of an individual vehicle. This can resolve occlusions and lead to better long-term planning of the vehicle.

While much research is currently being done on specific components and use-cases of IISs, information on building up an entire system is sparse. This chapter described how a modern IIS could be designed and successfully built. This includes its hardware and sensor setup, detection algorithms, calibration, data fusion, and position refinement. We have shown that our system can achieve good results at capturing the traffic of the observed highway stretch and that it can generate a reliable digital twin in near real-time. We have further demonstrated that it is possible to integrate the information captured by our system into the environment model of an autonomous vehicle to extend its limited perception range.

Our extensive quantitative evaluation has shown that our system is characterized by both a high classification and spatial accuracy, as well as a high detection rate, day and night. The primary purpose of our system is to enhance the perception of autonomous vehicles in the testbed. However, based on our evaluation results, it is evident that a system like ours could be used for various other applications too. Such applications could be detecting emerging traffic jams, detecting wrong-way drivers and immobile vehicles, or traffic flow management with lane and speed recommendations. Beyond this, the system could be used as a reference for testing, evaluating, and developing autonomous driving functions. And it is a vast data source for developing data-driven algorithms. Furthermore, our results show that the positioning accuracy of our system is sufficient for the application of trajectory prediction methods, like those we developed in Ch. 4.

Chapter 6

Motion Prediction in Providentia

In the previous chapter, we described how we designed and built the Providentia system and evaluated it. In this chapter, we explain how we integrated our motion prediction model that we developed in Ch. 4 into the system, and we extensively evaluate its performance. In particular, we conduct experiments that show how accurately our model can predict vehicle motion based on our system's digital twin and what runtimes it achieves. These measures are important for a safety-critical system like Providentia and necessary to know for what these predictions can be used.

6.1 Motivation

Motion prediction in an Intelligent Infrastructure System (IIS) like Providentia has several advantages compared to that performed locally on intelligent vehicles. Most importantly, as we explained in Ch. 5, the perception range of Providentia is much larger than that of a single vehicle, and because its sensors cover the road redundantly from many elevated perspectives, it is less prone to occlusions. This leads to a more complete state estimate of the road, i.e., more traffic participants are included in this estimate compared to what a single vehicle can observe. Furthermore, it is unlikely that specific observations of a vehicle's trajectory are missing in a system like ours. Missing positions in an agent's motion history – especially the more recent one – can cause problems, because changes in direction or speed may go unnoticed and cannot be accounted for in the prediction. Furthermore, in our evaluation we also showed that the positioning in Providentia is very accurate. This means that the trajectories our system records contain only small amounts of noise.

Because of this trustworthiness and precision, our system serves as a good data source for reliable motion prediction. However, it is not clear how motion prediction models are best utilized in IISs and what performance can be expected. This is important to know, especially if the resulting predictions are provided to vehicles that pass through the system and they rely on it. Furthermore, for additive services and reasoning within the system, accurate predictions are crucial too. Examples of such services are collision warnings and lane recommendations.

To answer these questions, in this chapter, we first describe how we prototypically integrated our prediction model in Providentia and how we prepare the digital twin

to make it usable for predictions. Then we thoroughly evaluate how our model FloMo performs on real-world Providentia data. In particular, we evaluate different configurations of our model in terms of their prediction accuracy and determine how our model can be optimized for use in our system. Lastly, we measure the runtime of our model and ablate it such that it to determine the trade-off between our model's speed and accuracy. A prediction model in a system like Providentia is only a small part of its whole perception and inference process, and the faster it can run, the less likely it becomes a bottleneck in the system.

6.2 Prediction Integration

To make predictions for the vehicles captured in our digital twin, we prototypically integrated our model into the Providentia system. Our system perceives the road at each time step and creates a sparse object list (see Ch. 5). This object list contains information such as positions, velocities, and unique ids for each vehicle in the testbed. The ids originate from our system's tracking module that follows each vehicle from frame to frame while passing through our testbed. Because the frame rate of our system is approximately constant, the time deltas between subsequent frames are approximately constant as well, which is an important factor for good motion prediction.

To make predictions, we collect the motion history of each agent up to the desired time step in the past. We achieve this by collecting incoming positions for each vehicle id. New incoming positions are added to their respective track, and those outside the observation window are discarded. The size of this history observation window is a hyperparameter in our system. The recorded tracks are converted in a compatible tensor format and passed through our model. The resulting predictions are then converted into a Robot Operating System (ROS) message format and broadcast to subscribers within our system. While we did not implement this yet, they could be transmitted to vehicles in the testbed in the same way we transmit the digital twin or even be included in the same message.

We only make predictions if our system records the complete motion history of an agent for the specified time window. This is not the case anymore when a vehicle leaves our testbed and stops being captured in the digital twin. In the case a vehicle goes missing only for a short time, our data fusion will keep its track and predict missing detections forward with a Constant Velocity Model (CVM). That a vehicle goes undetected for several consecutive time steps such that its track gets lost, for example, due to occlusions, is very rare, as our evaluation in Sec. 5.5.3 confirmed.

6.3 Evaluation

In this section, we first evaluate the prediction performance and then the runtime of our model on realistic Providentia data. These quantities are crucial for use in a real and safety-critical system like Providentia.

6.3.1 Prediction Accuracy

To assess how well our prediction model from Ch. 4 is suited for use in the Providentia system, we evaluate how accurately it can predict the motion of vehicles in the testbed. We perform this evaluation with the real-world data that we recorded for evaluating the performance of Providentia's digital twin in Sec. 5.5.

However, in contrast to the experiments in Sec. 5.5 we use the digital twin not only from those areas where we also recorded ground truth but from the entire testbed between our two measurement points. Furthermore, we use data from an extended recording period that lasted 13 min. We assume that over the whole recording time and across the entire testbed, the positioning accuracy of the digital twin is comparable with that of our positioning evaluation.

While the latest version of our system is significantly faster at day (see Sec. 5.4), the data we used for evaluation in Sec. 5.5 and the experiments in this section were recorded with an average frequency of 9 Hz at day and 14 Hz at night. We define 2 s as our observation window and 3 s as our desired prediction horizon. These windows correspond to 18 time steps observation and 27 time steps prediction during the day, and 28 time steps observation and 42 time steps prediction at night. To obtain our datasets for training, validation, and testing, we slice each agent's trajectory with a step-size of one into sequences of length 45 and 70 for day and night, respectively. To quantify our model's prediction performance, we use the Minimum Average Displacement Error (minADE) and Minimum Final Displacement Error (minFDE) metrics, as we did in Ch. 3.3 and Ch. 4.5. In total, our datasets consist of 2177 unique agents and 133452 sliced training samples at day, and 2413 unique agents and 187455 training samples at night. The average distance our models must predict up to the defined three seconds prediction horizon is 79.94 m, which means that on average, the vehicles in the testbed moved with a speed of 95.93 kmh.

Models and Training

As a simple baseline, we include the CVM-S in our evaluation because we used it as our main prediction model in Providentia before developing and integrating FloMo. Furthermore, the highway we make predictions for is almost straight, so predicting with the linear CVM-S is a reasonable choice for a baseline. As standard deviation for the CVM-S's cone-shaped distribution we choose $\sigma = 5°$, because we expect less angular variation compared to predicting for pedestrians (see Sec. 3.3.3).

Because we compared our model FloMo already in Sec. 4.5 to other state-of-the-art models, in this experiment, we only include different versions of our model. In particular, we use **FloMo**, **FloMo-S** that uses our proposed scaling transformation, and **FloMo-I** that is interaction-aware. Because the highway at which we constructed Providentia is almost perfectly straight, we do not use additional environmental information. Furthermore, due to our results for FloMo-S, we do not use our scaling transformation in FloMo-I.

From our complete dataset, we use 70 % for training, 5 % for validation and the remaining 25 % for evaluating test errors. We only train our model on full trajectories and also evaluate always over the full prediction horizon, because we are interested in its performance at 3 s in the future.

As our optimizer, we use Adam [17] with a learning rate of 0.001, batch size 64, and train for 15 epochs. In terms of our model parameters, we define the support for each spline flow as B=60 and use 8 knot points in this experiment. As in previous experiments, we again stack 10 flow layers. For our noise injection we set set $\alpha = 1$, $\beta = 0.01$ and $\gamma = 0.005$ and in our random scaling augmentation we use $\mu = 1$, $\sigma = 0.4$, $s_{\text{min}} = = 0.7$, and $s_{\text{max}} = = 1.3$. We determined all these hyper-parameters empirically through testing.

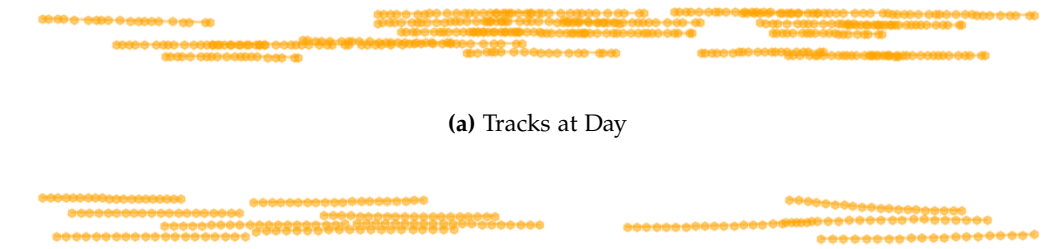
Results

The results of our experiments with the Providentia dataset are shown in Tab. 6.1. Compared to all versions of our novel model FloMo, the CVM-S's errors are significantly higher. This effect is especially strong for predictions during the day, at which it performs much worse than at night. This can be explained by differences in the tracks our system captures at day with cameras compared to those at night with radars. In Fig. 6.1 we show two scenes that contain such tracks. The tracks in Fig. 6.1a were recorded with our system's cameras and are compared to those in Fig. 6.1b much more irregular. This is most likely due to fluctuations in the processing times of our camera perception pipeline. The tracks are complete, but the time difference between some of the frames deviates from the average time difference. Because the CVM-S is making linear predictions based on the last velocity vector, such irregularities can cause drastic over or underestimation of the next predicted position. Furthermore, these errors accumulate, especially over a long prediction horizon. In contrast to that, the training of neural networks is based on statistical principles, and hence they automatically learn to filter irregularities in their inputs statistically.

Model	Day	Night
CVM-S	19.10 / 36.42	2.61 / 5.19
FloMo	3.13 / 2.48	1.22 / 1.89
FloMo-S	3.12 / 2.55	1.23 / 1.94
FloMo-I	**2.96 / 2.06**	**1.06 / 1.51**

Table 6.1: Prediction results of our model for Providentia, at day and night as minADE / minFDE. The interaction-aware version of our model performed the best.

Our basic FloMo achieved significantly lower errors than the CVM-S baseline. The performance gains are especially strong during the day, with a minADE of 3.13 m and a minFDE of 2.48 m. This shows that our model can account for the unequal time deltas



(a) Tracks at Day

(b) Tracks at Night

Figure 6.1: Image (a) shows tracks that our system captured with its cameras and image (b) tracks that were captured with our radars. Our system's tracks at day are more irregular than those at night. This is most likely the result of fluctuations in the processing times of our camera perception pipeline.

in the recorded track positions. However, augmenting our training data with random scaling for FloMo-S did not – unlike for pedestrian datasets in Sec. 4.5.2 and Sec. 4.5.3 – contribute to a better prediction performance. In contrast to that, including information about the target vehicle's neighbors did improve the prediction performance of our model. This is consistent with our hypothesis from Sec. 3.4.4 and our experiments with the Argoverse dataset in Sec. 4.5.4. In highly structured environments like roads, a braking maneuvers of a vehicle driving in front of another directly influences the other's future trajectory. Likewise, vehicles driving beside each other impede lane changes.

For the predictions during the day, except for the CVM-S, the models' predicted minADEs are worse than their minFDEs. This result is counter-intuitive, but we hypothesize that the irregularity of our day tracks can explain it. Our models learn to predict trajectories distributed just as in the training data. For a concrete prediction, it places position irregularities with the correct frequency but not where they actually occur, as they are distributed randomly. Then, on our dataset's relatively straight trajectories, summing the distances between predicted positions leads approximately to the correct endpoint, but many positions within the trajectory are slightly misplaced, leading to a higher minADE.

Fig. 6.2 shows a qualitative example for a prediction of the best version of our model FloMo-I at day. Most of the sampled trajectory predictions accurately fit the true movement of the target agent. However, our model also accounts for the possibility that the agent is performing an upward lane change or starts to accelerate. As it can be seen in Fig. 6.3, our model is also making accurate predictions based on our night measurements. In this example, it predicts a lane change, and as can be seen in the bottom image, this lane change was initiated in the observed motion history. It also accounts for the possibility that the agent is changing two lanes, crossing the middle one in the process.

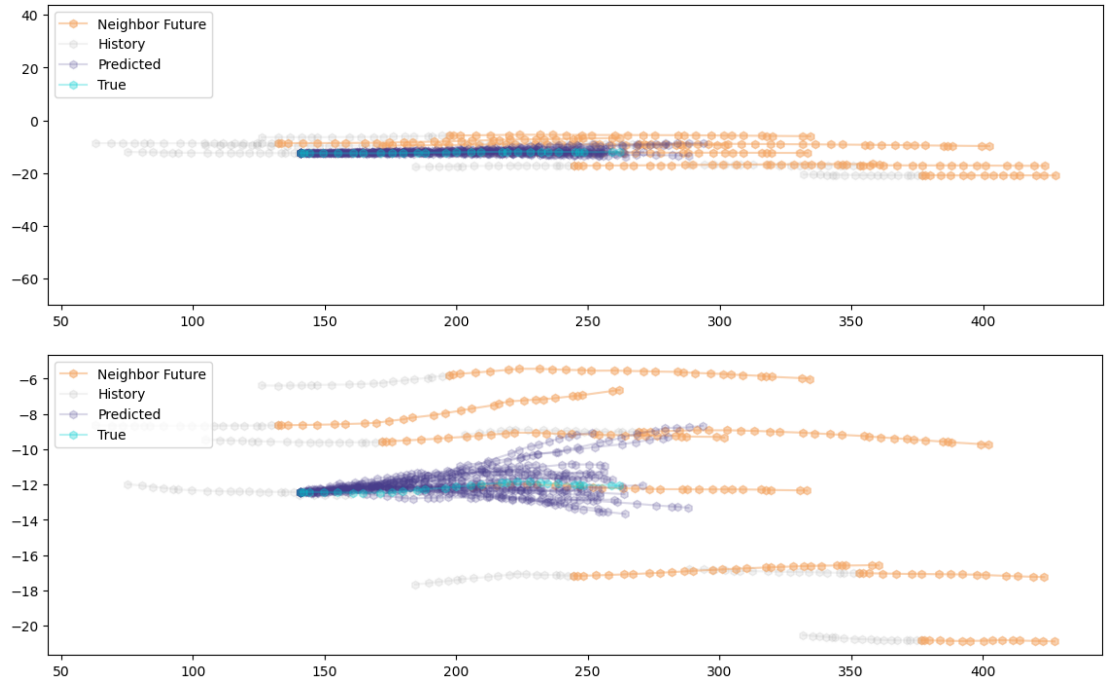


Figure 6.2: Prediction of FloMo-I in a day scene. Both images show the same prediction, but the coordinate system in the lower image is scaled differently for better visibility. Of the 20 predicted samples, the majority quite accurately fit the true movement.

In summary, our model can make very accurate predictions day and night. A prediction error of less than three meters over 3 s is most likely sufficient to help prevent collisions and enable safer high-level planning. Our results suggest that its prediction errors at day could be further reduced by stabilizing the sampling rate of our camera perception pipeline. This stabilization would result in more regular tracks that are easier to predict, similar to those our system generates from radar measurements at night.

Moreover, we must emphasize that the predictions we evaluated in this section are based on our system's digital twin and not on ground truth. In real-world applications, the digital twin is the only source of information on the traffic that we have. However, the true prediction error is a combination of the prediction error for the digital twin and the system's perception error (see Sec. 5.5.3). These errors can theoretically cancel out, but in the worst case, they sum up. Training our prediction model to predict ground truth directly is not necessary because all well correlated and systematic biases are already handled by our position refinement module (see Sec. 5.3.5). Nevertheless, improving our system's overall perception accuracy itself will improve the prediction of ground truth positions directly.

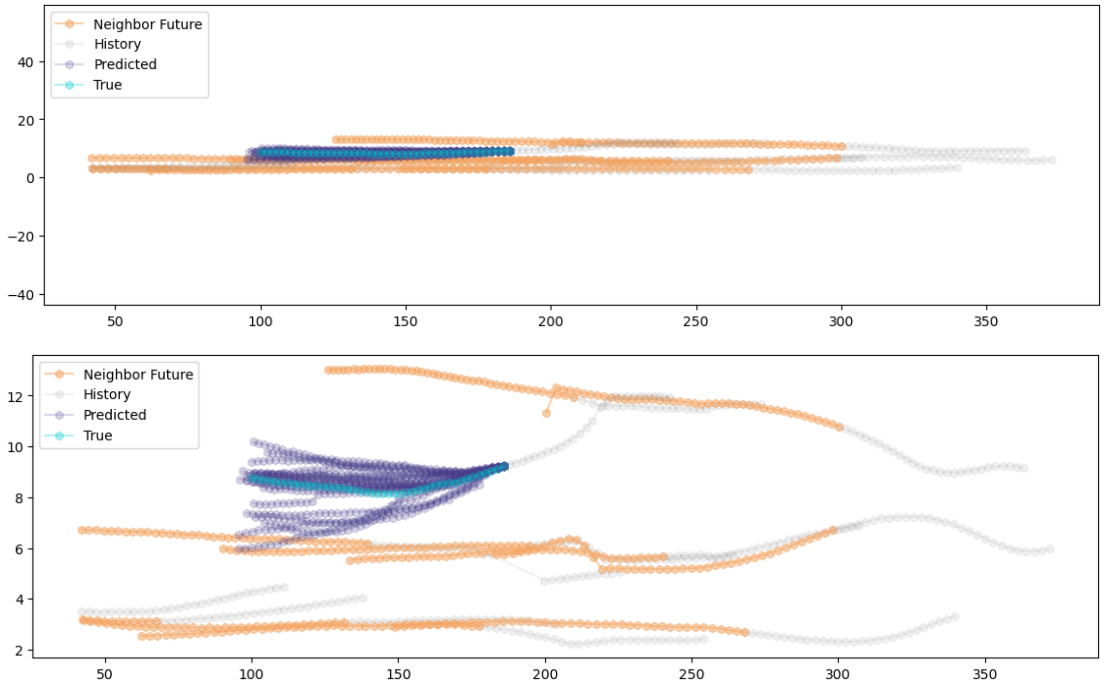


Figure 6.3: Prediction of FloMo-I in a night scene. As in Fig. 6.2, both images show the same prediction. Our model FloMo-I accurately predicts a lane change that was initiated in the observed motion history.

6.3.2 Runtime Analysis

In this section, we measure the runtime of our motion prediction model FloMo. As we explained earlier, a low execution time is essential for prediction models, such that they do not become a bottleneck in the overall system. Note that we only measure the execution time of our model itself and do not include boilerplate code that prepares tracks to enable predictions. Furthermore, along with runtime, we evaluate the performance of the different versions of our model. This analysis is important not only to optimize our model for speed, but also to find a fast variant that retains accuracy. We performed all experiments of this section on the Data Fusion Units (DFUs) that we described earlier in Sec. 5.3.1 and use as processing units in the Providentia system.

Models and Training

The models we compare in this experiment are the basic **FloMo** and the interaction-aware **FloMo-I** from the previous section. Furthermore, we evaluate **FloMo-I-F5**, in which we use only five instead of 10 flow modules. This modification cuts the model's size in half to increase speed but also reduces the flow's expressive power. Because our experiments in Sec. 3.4.2 showed that long motion histories do not seem to benefit the prediction performance of neural networks, we include **FloMo-I-H5** in our evaluation. This model only considers the five last observed time steps of the target agent's motion

Model	RT Day	RT Night	Error Day	Error Night
FloMo	18.9 Hz	11.61 Hz	3.13 / 2.48	1.22 / 1.89
FloMo-I	9.36 Hz	7.88 Hz	**2.96** / **2.06**	**1.06** / **1.51**
FloMo-I-F5	10.85 Hz	9.98 Hz	3.02 / 2.20	2.78 / 4.93
FloMo-I-H5	10.48 Hz	9.73 Hz	3.28 / 2.60	1.07 / 1.60
FloMo-I-I5	9.36 Hz	7.02 Hz	3.08 / 2.31	1.14 / 1.70
FloMo-P5	**23.94 Hz**	**17.83 Hz**	4.62 / 5.90	1.59 / 2.87

Table 6.2: Runtimes (RTs) of different versions of our model to predict one batch of 30 agents. We report errors as minADE / minFDE, as in previous chapters. The fastest model is FloMo-P5 which does not use interactions and only outputs five trajectory samples. FloMo-I-F5 is the fastest of the interaction-aware models but suffers in terms of prediction accuracy.

history. Assuming that not all neighbors in the vicinity of the target are relevant for making predictions, in **FloMo-I-I5** we limit the observed neighbors to the five closest ones. Lastly, with **FloMo-P5** we do not use interactions and only predict five future trajectories, instead of 20 as in all other models.

As the frame rate at day and night is different in our data, the time window our models take into account at day is slightly longer than at night, except for FloMo-I-H5. We predict a batch of 30 agents with each model, which would be a realistic setting in our system and report average runtimes over all test set batches. Nevertheless, our results are also relevant for larger batch sizes, as we observed in our experiments that doubling the batch size only increases runtime by approximately 15 %. For each evaluated model, we report the frequency with which a batch can be processed at day and night and its positioning errors.

Results

We show the results of our experiment in Tab. 6.2. Of our unmodified models, FloMo is almost double as fast as the interaction-aware FloMo-I. However, as we already showed in the previous section, interaction awareness improves prediction performance. Of the different interaction-aware versions, FloMo-I-F5 is the fastest but suffers in its prediction performance, especially in our night evaluation.

Reducing the history size, i.e., the size of our model's input vector, sightly improved our model's runtime but decreased the performance of FloMo-I-H5 during the day. The reason for this could be the irregular sampling of day tracks that we explained in the previous section. From the perspective of the prediction model, these irregularities behave like noise in the input trajectory's positions. In this case, an extended motion history provides more information for statistical filtering and leads to more robust predictions. This effect also explains why cutting the motion history did not impact the

prediction performance at night, where its performance is approximately equal to that of FloMo-I.

Reducing the number of observed neighbors to five slightly decreased prediction performance but unexpectedly increased the model's runtime. This runtime increase is because the selection of the agent's closest neighbors itself consumes time, which is not compensated by the faster inference of the prediction model. The fastest model in our evaluation is FloMo-P5, which uses the same architecture as FloMo, but only predicts five future trajectories per agent. Its errors are significantly higher because it is impossible to cover the same diversity with five predictions as with 20 predictions.

Overall, selecting the correct model configuration is a trade-off between runtime and prediction accuracy. Based on our results, the most favorable model is the unmodified FloMo, because it balances runtime and accuracy best. The model that matches our system's sampling rate more accurately is FloMo-P5. However, note that also models with a slower sampling frequency than the system itself can be used in Providentia when applied correctly. We outline some techniques to enable this in our future work in Sec. 7.2.

6.4 Conclusion

In this chapter, we thoroughly analyzed the performance of our developed prediction model FloMo in our IIS Providentia. In particular, we conducted experiments to determine its prediction accuracy and analyzed multiple versions of our model in terms of their runtime.

Our results show that the interaction-aware version of our model, which uses neighbor information, achieves the best performance in our system. Its average prediction errors are less than three meters, day and night. Considering that the average prediction horizon we evaluated was three seconds and approximately 80 m, the predictions of our model are highly accurate. They can be used to enable services like collision warnings and improve the safety of high-level planning in vehicles that pass through our system. As we outlined, we also believe that these results can be further improved, by improving the overall accuracy of the digital twin and the regularity of the generated tracks, especially during the day.

Besides prediction errors, we also evaluated the runtime of our model. In this evaluation, we included different versions of FloMo and tested multiple parameter settings. Our results show that the right choice for a system like Providentia is complex. It depends on the use case and is a trade-off between runtime and prediction accuracy. Intuitively reasonable changes, like reducing the number of observed neighbors, do not necessarily lead to the expected changes in performance and must be examined with caution before being used in the actual system.

Chapter 7

Conclusion

In this section, we first summarize the contributions of this thesis and discuss their implications for the domain of motion prediction, Intelligent Infrastructure Systems (IISs), and autonomous driving in general. Then we identify further open research questions and outline possible directions to address them.

7.1 Summary

This thesis addressed several problems related to neural motion prediction and environment perception. Both are essential capabilities of autonomous vehicles, and their reliability is critical for safe autonomous driving. A vehicle must know where other traffic participants are at any time, and by predicting their movements, it can anticipate where they will be in the future to avoid collisions and dangerous situations. In our work, we argued that prediction and perception are inseparable and must be addressed jointly. Only a complete and accurate environment perception can lead to precise predictions.

At first, we aimed to understand neural motion prediction from a fundamental point of view. Due to their computational power, when not trained and applied correctly, neural motion prediction models are more susceptible to severe failures than simple models like the Constant Velocity Model (CVM). Like other machine learning models, they are prone to overfitting. We demonstrated this with an experiment that showed that the CVM outperforms even state-of-the-art neural prediction models on popular prediction benchmarks. Motivated by this observation, we analyzed why neural networks could not utilize their complexity and information advantage to make superior predictions. Our analysis showed that when trained without caution, neural networks can learn environmental priors that are implicitly contained in the training data and negatively influence their performance and that the longer motion histories of agents contain mostly redundant information. Furthermore, in the case of pedestrians, interactions are too complex to predict and, on average, have only little influence on future trajectories. To address these issues, we showed that the use of subtle data augmentation can reduce the influence of environmental priors, and by focusing on relevant inputs, prediction models are less likely to overfit information that merely acts as noise in the training data. Furthermore, we argued that for vehicles explicitly provided environment and

interaction information can indeed help to make better predictions and later also confirmed this hypothesis in Sec. 4.5.4.

The results we obtained from our experiments with the generative CVM also indicated that many modern generative prediction models fail to learn the actual motion distribution they are trained for. If they would learn this distribution correctly, they should have outperformed the CVM that predicts a generic cone-shaped distribution of linear trajectories. Inspired by this observation, we developed a novel prediction model named FloMo, which is based on normalizing flows. Because we model prediction as a normalizing flow, our model provably learns a probability distribution, and we can train it with Maximum Likelihood Estimation (MLE). The flow transformations we use in our model are monotonic splines and demonstrated excellent performance in other density estimation tasks [91]. We extended this model with an interaction module based on fine-grained attention and showed that we could use the same module to encode environment information. Our model achieves state-of-the-art performance on several benchmarks, and the improvements obtained from our interaction and environment encoding on vehicle motion prediction support our argument that for vehicle motion prediction this information is valuable. Furthermore, we proposed a novel data augmentation that randomly scales trajectories during training and a noise injection method to stabilize training.

To get a reliable state estimate of the traffic, which is a precondition for precise motion prediction, we proposed the IIS Providentia. Providentia is a large-scale distributed sensor system that captures the traffic on a highway and generates a digital twin. Due to its sensors' favorable perspectives, multimodality, and redundant road coverage, it can accurately observe the highway day and night. This work describes our system in detail, including the selected sensor types and their positioning, hardware and software architecture, implemented algorithms, and challenges we faced during its build-up. A detailed description of a system like ours is novel in literature and can be used as a blueprint to build similar systems in the future. We demonstrated that it is possible to transmit the digital twin our system creates to vehicles passing through it and that it can be used fill in gaps in their perception, for example, caused by detection failures or due to occlusions. This completed perception enables safer motion planning in autonomous vehicles and helps to avoid collisions. Because the performance of a safety-critical system like Providentia is crucial, we evaluated it with an extensive experiment. In a system like ours, it is not sufficient to evaluate the detection performance of each sensor independently, but instead, a holistic evaluation setup that includes the whole perception pipeline is necessary. This complete evaluation includes calibration, object detection, sensor fusion, and tracking. To generate ground truth for our experiment, we recorded aerial images with a helicopter, labeled captured vehicles semi-automatically, and computed each detection's position in a global coordinate system. This evaluation shows that our system is very reliable at detecting vehicles and estimating their positions on the road. Therefore, it is suited to enhance the perception of autonomous vehicles, and it serves as an ideal data source for our motion prediction.

Lastly, we described how we prototypically implemented our prediction model FloMo in the Providentia system to complement its functionality. Our model predicts the future motion of vehicles that pass through our system based on the digital twin. To ensure these predictions are accurate enough for use in Providentia, we conducted experiments that evaluate our model's accuracy with real-world data. In particular, we evaluated different ablated versions of our model with day and night measurements. Our results show that our model can make accurate predictions, and incorporating interactions improved its performance. Furthermore, in a system like Providentia that should ideally operate in real-time, every component must have a fast execution time to not become a bottleneck. Therefore, we evaluated the runtime of FloMo and different versions of it. Our results showed that selecting the correct model is a trade-off between runtime and accuracy and helped identify versions suitable for our system.

7.2 Future Work

During our work for this thesis, we were able to identify possible future research directions and problems that are still unsolved. In the following, we will describe these for both neural motion prediction and IISs.

Neural Motion Prediction

The environment of an agent strongly influences its trajectory. To use this to our benefit, in Sec. 4.5.4 we encoded the road's centerlines to provide our prediction model information about the local road layout. Because the vehicle most likely will move along these lines, it enables our model to make better predictions. The centerlines we encoded are semantic information describing how the road progresses physically. In the future, prediction models could be extended with more semantic information, for example, traffic rules and signs [146], traffic lights, or the indicator signals of other vehicles. While this information does not determine the exact trajectory a traffic participant will take, it is predictive for maneuvers types and helps avoid significant prediction errors. Furthermore, taking into account static objects in the environment that physically constrain movement, for example, buildings, roadblocks, or parked vehicles, would further limit the number of possible future trajectories. In this case, we could enforce that the model does not predict colliding trajectories, or at least predictions of such trajectories could be penalized in training. All of these measures would require a tighter integration of the prediction model and the perception of the vehicle or IIS it is used in. In the case of pedestrian motion prediction, the development of new datasets with more diverse environments and more semantic information could be one way to train models that generalize better and manage to exploit environment information reliably.

The motion prediction model FloMo that we proposed learns a distribution over the future trajectories of a single target agent. This distribution is conditioned on the agent's motion history, neighboring agents' motion histories, and additional environment information. From a purely probabilistic point of view, this is not ideal. When one

agent takes a particular trajectory, this limits the trajectories that other agents in the scene can take. Ideally, our model should learn a joint motion distribution over the future trajectories of all agents in the scene. This would enable us to sample possible scenarios for how the whole scene evolves in the future and avoid sampled trajectory combinations for different agents that are incompatible. Small steps in this research direction have been made [147], but it could be worth exploring this in more depth. Significant difficulties with this approach are the variable number of agents we must predict. One way to approach this could be to use methods that predict (trajectory) sets from vector encodings [148]. Alternatively, by the chain rule of probability, the complex joint distribution could be factorized into a product of distributions for each separate agent and then trained autoregressively. Moreover, each factor distribution could be represented with a normalizing flow, as we did in this work.

Another research direction worth addressing in motion prediction is the handling of outlier behavior [149]. Outliers occur in every domain that deals with the natural world and can be defined as exceedingly rare data points. For machine learning models trained on the whole data distribution, such data points are difficult to handle. However, in motion prediction, the correct handling of such outliers is essential, for example, when the driving behavior of agents becomes irregular after an accident happens or in the case of a wrong-way driver. Applying techniques that detect these outliers and training the model such that it becomes robust towards them, for example, by oversampling those samples where the prediction error is high [150], could be beneficial in the future.

Because prediction is directly tied to perception, it would be interesting to see if smoothing the motion history, e.g., by fitting a lower degree polynomial and then projecting measurements on it, would improve prediction results. Pre-processing the input trajectories like this would essentially reduce measurement noise. Furthermore, besides noise in its position estimates, a perception system can also suffer from noise in the time dimension. Ideally, given a target frequency, the sampling rate of a perception system should be constant. This is important for components like a motion prediction model that uses information about how the states of observed objects change over time. However, variation in the number of observed objects, network delays, or irregular sensor measurements can cause fluctuations in the system's sampling rate. To restore regularity in the observed trajectories, we could temporally interpolate the detected positions in a track. A different approach could be to explicitly provide the predicting network with information about the time deltas between the trajectory's measurements.

There are also open research questions concerning the application of motion prediction models in IISs specifically. Once an IIS becomes more extensive and potentially stretches over multiple kilometers, the question arises whether it is better to train a single general prediction model that is operating across the whole system or if it is beneficial to train multiple specialized models. Such a specialized model could be trained for a specific area it is observing within the system and most likely make more accurate predictions in that area than a general model. However, the general model would be better suited to make correct predictions for outliers because it is trained with more diverse data.

One more problem that does also apply to autonomous vehicles, but is more severe in IISs, is that its perception performance is not static but changes over time. As we explained in Ch. 5.6, in a distributed system like Providentia that observes vehicles over large distances, small miscalibrations can have a significant impact on its precision. Such miscalibrations naturally occur due to weather and temperature changes but can be corrected with occasional recalibration. However, until recalibration, these changes also affect the prediction model. In the future, a solution implementing online training algorithms that constantly fit the prediction model to the system's current status could be a solution for this issue. Alternatively, the system could track prediction errors and trigger a training update of the model once the metrics exceed a defined threshold.

Another obstacle in applying state-of-the-art prediction models in systems like Providentia and autonomous vehicles is runtime. Competitive prediction models that use a variety of context information may not perform at a speed that matches the system they are applied in. Besides ablating and pruning the model, as we did in Sec. 6.3.2, the parallelization of encoding modules and model quantization could be applied. Other ways to boost prediction frequency could be to predict a longer time horizon than necessary and use slices of this prediction at each time step or to interpolate between prediction points and extrapolate at endpoints. The latter approach can be interpreted as applying a short-term prediction to adapt a long-term prediction to the desired frequency.

Intelligent Infrastructure Systems

In this work, we showed that our IIS Providentia can reliably detect vehicles on the highway day and night. We also observed our system's function under adverse weather conditions, and demonstrated promising performance. However, a thorough quantitative evaluation in such scenarios would be valuable. Furthermore, the traffic we recorded in our evaluation was relatively regular with a medium amount of vehicles. It would be interesting to test the performance of our system during traffic jams with severe occlusions in the future.

One way to improve the system's performance under sub-optimal conditions could be to deploy more and different types of sensors. In Providentia, we selected cameras and radars as the sensor types to use. These sensor types complement each other well because cameras can detect stationary vehicles, and radars perform better at night, as they are not blinded by vehicle headlights. However, in the future, it would be interesting to test how well other sensor types fit into an IIS like Providentia. For example, lidars would enable an IIS to accurately measure the dimensions of each vehicle passing through the testbed. Exploring the use of event cameras, which have a high dynamic range and do not suffer from motion blur [151], would be interesting too. Moreover, a system like Providentia that communicates with the vehicles passing through it could treat these vehicles as mobile sensors and receive information from their local perception and location. Then it could integrate the received information with its own perception in a global estimate of the highway state.

What becomes more difficult with using many sensors is the system's calibration. Each new sensor increases the system's degrees of freedom. Its pose in the overall system and relative to each other sensor must be determined. Furthermore, even an ideal initial calibration slowly deteriorates over time due to temperature changes, oscillations of measurement points due to wind and weather, and vibrations. To guarantee that the system continually delivers the same positioning performance without tight human control and intervention, automatic calibration methods are necessary. In a system with multimodal sensors like Providentia, this is incredibly challenging. Most likely, a combination of regression models for a coarse initial calibration [107], combined with classic optimization for the last steps, would be successful. Using an HD map to identify localized reference points for calibration could be helpful too. While such a global auto-calibration method could be operated offline, a sensor-wise online recalibration could be used to compensate instantaneous vibrations.

To determine our system's performance, we evaluated it holistically by recording geo-referenced aerial images to obtain ground truth. This extensive evaluation was necessary to answer our research questions and develop our system to achieve high positioning accuracy and detection rates. In the future, it would be beneficial to monitor system performance during long-term operation frequently. Knowing its accuracy at all times is imperative to ensure its function in safety-critical applications. However, an evaluation like ours would be economically infeasible on a regular basis. As an alternative, the system's detection and positioning performance could in the future be measured by utilizing a self-localizing test vehicle and driving with it through the testbed. By identifying the test vehicle in the system's perception and matching it to its representation in the digital twin, we could compute our system's positioning accuracy and detection rate. One way to account for various looks and vehicle types could be to replace the vehicles in the camera images with time-coherent style transfer methods [152], [153]. As a cheaper alternative for acquiring aerial images, the use of drones could be explored.

To deploy a system like Providentia nationwide, reducing hardware costs and simplifying deployment would be necessary. One way to achieve this goal could be to integrate the computing units that process sensor measurements directly into the sensor and to use modern wireless communication protocols like 5G to avoid cabling installation. Furthermore, studies to identify areas where such systems have the most significant impact would be helpful. Especially in areas with poor visibility and frequent traffic accidents, the cost of building such an IIS would be justified, and the system would amortize quickly. It is unknown what density of measurement points for a system like ours is ideal for the support of autonomous vehicles. However, it most likely should be increased in more dangerous areas, and fewer measurement points are needed where traffic is lighter and safer. Concerning the system's use cases, services like traffic jam detection, ghost driver detection, collision warnings, and traffic flow management should be explored.

In conclusion, we believe that there are many exciting directions to go forward with research on motion prediction and IISs. Advances in these research areas will help enable autonomous vehicles and increase the safety on our roads.

List of Figures

List of Tables

Acronyms

ADAS	Advanced Driver Assistance Systems. 1
ADE	Average Displacement Error. 21–23, 26, 53
ANN	Artificial Neural Network. 1, 8
BPTT	Backpropagation through time. 10
CAM	Constant Acceleration Model. 23
CNN	Convolutional Neural Network. 19, 37
CVM	Constant Velocity Model. 17, 19–24, 26, 29, 30, 32, 33, 35, 38, 67, 84, 93, 94
DFU	Data Fusion Unit. 63, 64, 89
ELU	Exponential Linear Unit. 41, 52
FDE	Final Displacement Error. 21–23, 26, 53
FNN	Feed-Forward Neural Network. 8–11, 22, 24, 25, 27, 29, 30, 40–42, 52, 67
FoV	Field of View. 60, 63, 65
GAN	Generative Adversarial Neural Network. 12–14, 19, 31, 35, 37, 44, 46, 49
GM-PHD	Gaussian Mixture Probability Hypothesis Density. 66, 67
GMM	Gaussian Mixture Model. 37
GPS	Global Positioning System. 65, 66, 69, 71

GRU Gated Recurrent Unit. 11

IIS Intelligent Infrastructure System. 2–4, 7, 17, 18, 35, 59–62, 66, 79, 81, 83, 91, 93–98

KLD Kullback-Leibler divergence. 12, 44

lidar Light Detection and Ranging. 61, 69, 70, 72, 97

LSTM Long Short-term Memory. 10, 11, 13, 18, 19, 22, 23, 26, 37

MAF Masked Autoregressive Flow. 15

minADE Minimum Average Displacement Error. 22, 23, 45, 47, 50, 51, 85–87, 90

minFDE Minimum Final Displacement Error. 22, 23, 45, 47, 50, 51, 85–87, 90

MLE Maximum Likelihood Estimation. 3, 8, 36, 37, 44, 94

MLP Multilayer Perceptron. 8

MR Miss Rate. 46

MSE Mean Squared Error. 21, 31

OR Oracle Top 10%. 46, 49

radar Radio Detection and Ranging. 60, 62–67, 69, 74–80, 97

ReLU Rectified Linear Unit. 9, 22

RFS Random Finite Set. 66

RHS Right-hand side (of equation). 12

RMSE Root-mean-square Error. 73, 74, 76, 78

RNN Recurrent Neural Network. 10, 11

ROS Robot Operating System. 64, 65, 80, 81, 84

RT runtime. 90

TCP Transmission Control Protocol. 81

UTM Universal Transverse Mercator. 72

VAE Variational Autoencoder. 12, 14, 19, 31, 36, 37, 44, 49

Bibliography

[1] E. Dickmanns and A. Zapp, "Guiding land vehicles along roadways by computer vision," in *Congres Automatique*, 1985.

[2] H.-H. Braess and G. Reichart, "Prometheus: A vision of the intelligent car on intelligent roads? an attempted critical appraisal. I," in *Automobiltechnische Zeitschrift (ATZ)*, 1995.

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge," in *Journal of Field Robotics*, 2006.

[4] Statistisches Bundesamt (Destatis), "Verkehrsunfälle 2020," in *Tech. Rep.*, 2021.

[5] European Commission. (2021). "Road safety: 4,000 fewer people lost their lives on eu roads in 2020 as death rate falls to all-time low." visited 2022-01-20, [Online]. Available: `https://ec.europa.eu/commission/presscorner/detail/en/IP_21_1767`.

[6] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," in *IEEE Internet of Things Journal (IoT-J)*, 2020.

[7] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," in *The International Journal of Robotics Research (IJRR)*, 2020.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] A. Graves, "Generating sequences with recurrent neural networks," in *arXiv preprint arXiv:1308.0850*, 2013.

[11] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *arXiv preprint arXiv:1609.03499*, 2016.

[12] C. Michaelis, B. Mitzkus, R. Geirhos, E. Rusak, O. Bringmann, A. S. Ecker, M. Bethge, and W. Brendel, "Benchmarking robustness in object detection: Autonomous driving when winter is coming," in *arXiv:1907.07484*, 2019.

[13] E. Rusak, L. Schott, R. S. Zimmermann, J. Bitterwolf, O. Bringmann, M. Bethge, and W. Brendel, "A simple way to make neural networks robust against diverse image corruptions," in *European Conference on Computer Vision (ECCV)*, 2020.

[14] K. N. Qureshi and A. H. Abdullah, "A survey on intelligent transportation systems," in *Middle-East Journal of Scientific Research (MEJSR)*, 2013.

[15] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press Cambridge, 2016.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference for Learning Representations (ICLR)*, 2015.

[18] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," in *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS)*, 1998.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2013.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural Computation*, 1997.

[21] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *arXiv preprint arXiv:1406.1078*, 2014.

[22] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[23] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "Stargan v2: Diverse image synthesis for multiple domains," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2016.

[25] D. W. Shu, S. W. Park, and J. Kwon, "3d point cloud generative adversarial network based on tree structured graph convolutions," in *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[26] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[27] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *European Conference on Computer Vision (ECCV)*, 2020.

[29] D. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

[31] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," in *arXiv preprint arXiv:1912.02762*, 2019.

[32] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[33] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, "What the constant velocity model can teach us about pedestrian motion prediction," in *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[34] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng, "Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[35] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, and S. Savarese, "Sophie: An attentive gan for predicting paths compliant to social and physical constraints," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[38] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[39] S. Pellegrini, A. Ess, and L. Van Gool, "Predicting pedestrian trajectories," in *Visual Analysis of Humans*, 2011.

[40] R. Baxter, M. Leach, S. Mukherjee, and N. Robertson, "An adaptive motion model for person tracking with instantaneous head-pose features," in *IEEE Signal Processing Letters*, 2015.

[41] J. F. Kooij, F. Flohr, E. A. Pool, and D. M. Gavrila, "Context-based path prediction for targets with switching dynamics," in *International Journal of Computer Vision (IJCV)*, 2019.

[42] A. Ess, K. Schindler, B. Leibe, and L. Van Gool, "Object detection and tracking for autonomous navigation in dynamic environments," in *The International Journal of Robotics Research (IJRR)*, 2010.

[43] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang, "Person tracking and following with 2d laser scanners," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[44] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn, "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker," in *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011.

[45] K. Yamaguchi, A. C. Berg, L. E. Ortiz, and T. L. Berg, "Who are you with and where are you going?" In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[46] S. Becker, R. Hug, W. Hubner, and M. Arens, "Red: A simple but effective baseline predictor for the trajnet benchmark," in *European Conference on Computer Vision Workshops (ECCV Workshops)*, 2018.

[47] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," in *Physical Review E*, 1995.

[48] M. Luber, J. Stork, G. Tipaldi, and K. Arras, "People tracking with human motion predictions from social forces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[49] Y. Xu, Z. Piao, and S. Gao, "Encoding crowd interaction with deep neural network for pedestrian trajectory prediction," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[50] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "Trafficpredict: Trajectory prediction for heterogeneous traffic-agents," in *AAAI Conference on Artificial Intelligence*, 2019.

[51] J. Amirian, J.-B. Hayet, and J. Pettré, "Social ways: Learning multi-modal distributions of pedestrian trajectories with gans," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, 2019.

[52] L. Ballan, F. Castaldo, A. Alahi, F. Palmieri, and S. Savarese, "Knowledge transfer for scene-specific motion prediction," in *European Conference on Computer Vision (ECCV)*, 2016.

[53] N. Jaipuria, G. Habibi, and J. P. How, "A transferable pedestrian motion prediction model for intersections with different geometries," in *arXiv preprint arXiv:1806.09444*, 2018.

[54] F. Bartoli, G. Lisanti, L. Ballan, and A. Del Bimbo, "Context-aware trajectory prediction," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2018.

[55] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[56] M. Koschi, C. Pek, M. Beikirch, and M. Althoff, "Set-based prediction of pedestrians in urban environments considering formalized traffic rules," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[57] G. Habibi, N. Jaipuria, and J. P. How, "Context-aware pedestrian motion prediction in urban intersections," in *arXiv preprint arXiv:1806.09453*, 2018.

[58] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, A. Bagnell, M. Hebert, A. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[59] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," in *Computer Graphics Forum*, 2007.

[60] R. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," in *Neural Computation*, 1989.

[61] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, "What the constant velocity model can teach us about pedestrian motion prediction," in *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[62] Z. Lipton and J. Steinhardt, "Troubling trends in machine learning scholarship," in *International Conference on Machine Learning Debates (ICML)*, 2018.

[63] W. Brendel and M. Bethge, "Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet," in *International Conference on Learning Representations (ICLR)*, 2019.

[64] J. Devlin, S. Gupta, R. Girshick, M. Mitchell, and C. L. Zitnick, "Exploring nearest neighbor approaches for image captioning," in *arXiv*, 2015.

[65] C. Schöller and A. Knoll, "Flomo: Tractable motion prediction with normalizing flows," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[66] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[67] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, "Mode regularized generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[68] S. Agarwal, H. Sikchi, C. Gulino, and E. Wilkinson, "Imitative planning using conditional normalizing flow," in *arXiv preprint arXiv:2007.16162*, 2020.

[69] A. Bhattacharyya, M. Hanselmann, M. Fritz, B. Schiele, and C.-N. Straehle, "Conditional flow variational autoencoders for structured sequence prediction," in *arXiv preprint arXiv:1908.09008*, 2019.

[70] H. Xue, D. Q. Huynh, and M. Reynolds, "Scene gated social graph: Pedestrian trajectory prediction based on dynamic social graphs and scene constraints," in *arXiv preprint arXiv:2010.05507*, 2020.

[71] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "Spatially-aware graph neural networks for relational behavior forecasting from sensor data," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[72] C. Luo, L. Sun, D. Dabiri, and A. Yuille, "Probabilistic multi-modal trajectory prediction with lane attention for autonomous vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[73] F. Diehl, T. Brunner, M. T. Le, and A. Knoll, "Graph neural networks for modelling traffic participant interaction," in *Intelligent Vehicles Symposium (IV)*, 2019.

[74] M. Bahari, V. Zehtab, S. Khorasani, S. Ayromlou, S. Saadatnejad, and A. Alahi, "Svg-net: An svg-based trajectory prediction model," in *arXiv preprint arXiv:2110.03706*, 2021.

[75] N. Nikhil and B. Tran Morris, "Convolutional neural network for trajectory prediction," in *European Conference on Computer Vision (ECCV)*, 2018.

[76] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, *et al.*, "Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction," in *arXiv preprint arXiv:2111.14973*, 2021.

[77] S. Srikanth, J. A. Ansari, R. K. Ram, S. Sharma, J. K. Murthy, and K. M. Krishna, "Infer: Intermediate representations for future prediction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[78] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer networks for trajectory forecasting," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2020.

[79] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, "Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[80] K. Mangalam, H. Girase, S. Agarwal, K.-H. Lee, E. Adeli, J. Malik, and A. Gaidon, "It is not the journey but the destination: Endpoint conditioned trajectory prediction," in *European Conference on Computer Vision (ECCV)*, 2020.

[81] E. Pajouheshgar and C. H. Lampert, "Back to square one: Probabilistic trajectory forecasting without bells and whistles," in *Conference on Neural Information Processing Systems Workshops (NeurIPS Workshops)*, 2018.

[82] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," in *Communications on Pure and Applied Mathematics*, 2013.

[83] S. Kim, S.-G. Lee, J. Song, J. Kim, and S. Yoon, "Flowavenet: A generative flow for raw audio," in *International Conference on Machine Learning (ICML)*, 2018.

[84] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[85] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International Conference on Machine Learning*, 2019.

[86] K. Madhawa, K. Ishiguro, K. Nakago, and M. Abe, "Graphnvp: An invertible flow model for generating molecular graphs," in *arXiv preprint arXiv:1905.11600*, 2019.

[87] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-vae: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations (ICLR)*, 2017.

[88] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, "Neural autoregressive flows," in *International Conference on Machine Learning (ICML)*, 2018.

[89] A. Bhattacharyya, C.-N. Straehle, M. Fritz, and B. Schiele, "Haar wavelet based block autoregressive flows for trajectories," in *DAGM German Conference on Pattern Recognition (GCPR)*, 2020.

[90] Y. J. Ma, J. P. Inala, D. Jayaraman, and O. Bastani, "Diverse sampling for normalizing flow based trajectory forecasting," in *arXiv preprint arXiv:2011.15084*, 2020.

[91] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[92] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," in *International Conference on Learning Representations Workshops (ICLR Workshops)*, 2015.

[93] J. Gregory and R. Delbourgo, "Piecewise rational quadratic interpolation to monotonic data," in *IMA Journal of Numerical Analysis (IMAJNA)*, Oxford University Press, 1982.

[94] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[95] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *International Conference on Learning Representations (ICLR)*, 2016.

[96] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, "Deep sets," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[97] H. Choi, K. Cho, and Y. Bengio, "Fine-grained attention mechanism for neural machine translation," in *Neurocomputing*, 2018.

[98] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[99] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[100] H. Kim, H. Lee, W. H. Kang, J. Y. Lee, and N. S. Kim, "Softflow: Probabilistic framework for normalizing flow on manifolds," in *arXiv preprint arXiv:2006.04604*, 2020.

[101] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning social etiquette: Human trajectory understanding in crowded scenes," in *European Conference on Computer Vision (ECCV)*, 2016.

[102] X. Huang, S. G. McGill, J. A. DeCastro, L. Fletcher, J. J. Leonard, B. C. Williams, and G. Rosman, "Diversitygan: Diversity-aware vehicle motion prediction via latent semantic sampling," in *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[103] A. Krämmer, C. Schöller, D. Gulati, and A. Knoll, "Providentia–a large scale sensing system for the assistance of autonomous vehicles," in *Robotics Science and Systems Workshops (RSS Workshops)*, 2019.

[104] A. Krämmer, C. Schöller, F. Kurz, D. Rosenbaum, and A. Knoll, "Vorausschauende Wahrnehmung für sicheres automatisiertes Fahren. Validierung intelligenter Infrastruktursysteme am Beispiel von Providentia," in *Internationales Verkehrswesen*, 2020.

[105] A. Krämmer, C. Schöller, D. Gulati, V. Lakshminarasimhan, F. Kurz, D. Rosenbaum, and A. Knoll, "Providentia – a large-scale sensor system for the assistance of autonomous vehicles and its evaluation," in *Journal of Field Robotics*, 2021.

[106] H. Menouar, I. Guvenc, K. Akkaya, A. S. Uluagac, A. Kadri, and A. Tuncer, "Uav-enabled intelligent transportation systems for the smart city: Applications and challenges," in *IEEE Communications Magazine*, 2017.

[107] C. Schöller, M. Schnettler, A. Krämmer, G. Hinz, M. Bakovic, M. Güzet, and A. Knoll, "Targetless rotational auto-calibration of radar and camera for intelligent transportation systems," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[108] G. Hinz, M. Büchel, F. Diehl, G. Chen, A. Krämmer, J. Kuhn, V. Lakshminarasimhan, M. Schellmann, U. Baumgarten, and A. Knoll, "Designing a far-reaching view for highway traffic scenarios with 5g-based intelligent infrastructure," in *8. Tagung Fahrerassistenz*, 2017.

[109] G. Hinz, J. Eichinger, M. Büchel, and A. Knoll, "Proactive video-based use of telecommunications technologies in innovative motorway scenarios," in *GI/ITG KuVS Fachgespräch Inter-Vehicle Communication*, 2017.

[110] S. E. Shladover, "The california path program of ivhs research and its approach to vehicle-highway automation," in *IEEE Intelligent Vehicles Symposium (IV)*, 1992.

[111] DIGINETPS. (2017). "Diginetps - the digitally connected protocol track." visited 2020-06-08, [Online]. Available: `https://diginet-ps.de/en/home`.

[112] Veronika. (2017). "Veronika." visited 2020-06-08, [Online]. Available: `https://www.bmvi.de/SharedDocs/DE/Artikel/DG/AVF-projekte/veronika.html`.

[113] University of Antwerp. (2018). "Antwerp smart highway." visited 2021-04-04, [Online]. Available: `https://www.uantwerpen.be/en/research-groups/idlab/infrastructure/smart-highway`.

[114] NYC Connected. (2015). "Nyc connected vehicle project." visited 2021-04-04, [Online]. Available: `https://cvp.nyc`.

[115] M-City. (2015). "M-City project." visited 2021-01-21, [Online]. Available: `https://mcity.umich.edu/our-work/mcity-test-facility`.

[116] T. Fleck, K. Daaboul, M. Weber, P. Schörner, M. Wehmer, J. Doll, S. Orf, N. Sußmann, C. Hubschneider, M. Zofka, F. Kuhnt, R. Kohlhaas, I. Baumgart, R. Zöllner, and J. Zöllner, "Towards large scale urban traffic reference data: Smart infrastructure in the Test Area Autonomous Driving Baden-Württemberg," in *International Conference on Intelligent Autonomous Systems (ICoIAS)*, 2018.

[117] M. Gabb, H. Digel, T. Müller, and R.-W. Henn, "Infrastructure-supported perception and track-level fusion using edge computing," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[118] S. Seebacher, B. Datler, J. Erhart, G. Greiner, M. Harrer, P. Hrassnig, A. Präsent, C. Schwarzl, and M. Ullrich, "Infrastructure data fusion for validation and future enhancements of autonomous vehicles' perception on austrian motorways," in *IEEE International Conference on Connecected Vehicles and Expo (ICCVE)*, 2019.

[119] J. Miller, "Vehicle-to-vehicle-to-infrastructure (v2v2i) intelligent transportation system architecture," in *IEEE Intelligent Vehicles Symposium (IV)*, 2008.

[120] I. Kabashkin, "Reliability of bidirectional v2x communications in the intelligent transport systems," in *Advances in Wireless and Optical Communications (RTUWO)*, 2015.

[121] KoRA9. (2017). "Kooperative Radarsensoren für das digitale Testfeld A9 - KoRA9." visited 2020-07-10, [Online]. Available: `https://www.bmvi.de/SharedDocs/DE/Artikel/DG/AVF-projekte/KoRA9.html`.

[122] F. Geissler and R. Gräfe, "Optimized sensor placement for dependable roadside infrastructures," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[123] S. Zhang, G. Wu, J. P. Costeira, and J. M. F. Moura, "Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[124] ——, "Understanding traffic density from large-scale web camera data," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[125] Y. Shen, T. Xiao, H. Li, S. Yi, and X. Wang, "Learning deep neural networks for vehicle re-id with visual-spatio-temporal path proposals," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[126] Y. Zhou and L. Shao, "Viewpoint-aware attentive multi-view inference for vehicle re-identification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[127] L. Yu, D. Zhang, X. Chen, and A. Hauptmann, "Traffic danger recognition with surveillance cameras without training data," in *IEEE International Conference on Advanced Video and Signal-based Surveillance (AVSS)*, 2018.

[128] F. Diehl, T. Brunner, M. Truong Le, and A. Knoll, "Graph neural networks for modelling traffic participant interaction," in *arXiv preprint arXiv:1903.01254v2*, 2019.

[129] J. Liu, Y. Luo, H. Xiong, T. Wang, H. Huang, and Z. Zhong, "An integrated approach to probabilistic vehicle trajectory prediction via driver characteristic and intention estimation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[130] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: An open-source robot operating system," in *IEEE International Conference on Robotics and Automation Workshops (ICRA Workshops)*, 2009.

[131] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," in *arXiv preprint arXiv:2004.10934*, 2020.

[132] N. K. Kanhere and S. T. Birchfield, "A taxonomy and analysis of camera calibration methods for traffic monitoring applications," in *IEEE Transactions on Intelligent Transportation Systems*, 2010.

[133] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," in *IEEE Aerospace and Electronic Systems Magazine*, 2004.

[134] R. P. S. Mahler, *Statistical multisource-multitarget information fusion*. Artech House, 2007.

[135] ——, *Advances in statistical multisource-multitarget information fusion*. Artech House, 2014.

[136] B.-N. Vo and W.-K. Ma, "The gaussian mixture probability hypothesis density filter," in *IEEE Transactions on Signal Processing*, 2006.

[137] K. Panta, D. E. Clark, and B.-N. Vo, "Data association and track management for the gaussian mixture probability hypothesis density filter," in *IEEE Transactions on Aerospace and Electronic Systems*, 2009.

[138] M. Vasic and A. Martinoli, "A collaborative sensor fusion algorithm for multi-object tracking using a gaussian mixture probability hypothesis density filter," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2015.

[139] R. P. S. Mahler, "Optimal/robust distributed data fusion: A unified approach," in *Signal Processing, Sensor Fusion, and Target Recognition IX*, 2000.

[140] T. Kessler, J. Bernhard, M. Büchel, K. Esterle, P. Hart, D. Malovetz, M. Truong Le, F. Diehl, T. Brunner, and A. Knoll, "Bridging the gap between open source software and vehicle hardware for autonomous driving," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[141] S. M. Azimi, R. Bahmanyar, C. Henry, and F. Kurz, "Eagle: Large-scale vehicle detection dataset in real-world scenarios using aerial imagery," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2021.

[142] S. M. Azimi, C. Henry, L. Sommer, A. Schumann, and E. Vig, "Skyscapes – fine-grained semantic understanding of aerial scenes," in *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[143] F. Kurz, T. Krauß, H. Runge, D. Rosenbaum, and P. Angelo, "Precise aerial image orientation using sar ground control points for mapping of urban landmarks," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS Archives)*, 2019.

[144] J. Munkres, "Algorithms for the assignment and transportation problems," in *Journal of the Society for Industrial and Applied Mathematics*, 1957.

[145] F. Geissler, A. Unnervik, and M. Paulitsch, "A plausibility-based fault detection method for high-level fusion perception systems," in *IEEE Open Journal of Intelligent Transportation Systems*, 2020.

[146] K. Esterle, L. Gressenbuch, and A. Knoll, "Formalizing traffic rules for machine interpretability," in *IEEE Connected and Automated Vehicles Symposium (CAVS)*, 2020.

[147] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun, "Implicit latent variable model for scene-consistent motion forecasting," in *arXiv preprint arXiv:2007.12036*, 2020.

[148] A. R. Kosiorek, H. Kim, and D. J. Rezende, "Conditional set generation with transformers," in *arXiv preprint arXiv:2006.16841*, 2020.

[149] L. Gressenbuch and M. Althoff, "Predictive monitoring of traffic rules," in *IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021.

[150] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," in *Journal of Artificial Intelligence Research (JAIR)*, 2002.

[151] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2019.

[152] C. Gao, D. Gu, F. Zhang, and Y. Yu, "Reconet: Real-time coherent video style transfer network," in *Asian Conference on Computer Vision (ACCV)*, 2018.

[153] X. Xia, T. Xue, W.-s. Lai, Z. Sun, A. Chang, B. Kulis, and J. Chen, "Real-time localized photorealistic video style transfer," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021.