Probabilistic Goal-Directed Pedestrian Prediction by Means of Artificial Neural Networks

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Maschinenbau
des Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von

DIPL.-ING. EIKE REHDER

aus Hamburg

Tag der mündlichen Prüfung:	11.02.2022
Hauptreferent:	Prof. Dr.-Ing. Christoph Stiller
Korreferent:	Prof. Dr.-Ing. Klaus Dietmayer

Vorwort

Die hier vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Mess- und Regelungstechnik am Karlsruher Institut für Technologie (KIT), teilweise in Zusammenarbeit mit der BMW AG im Kontext des UR:BAN-Projekts. Zunächst möchte ich mich bei Herrn Prof. Christoph Stiller für die Betreuung dieser Arbeit, die wissenschaftlichen Anregungen und die Freiheiten in der Forschung bedanken. Herrn Prof. Klaus Dietmayer danke ich für die Übernahme des Korreferats.

Bei meinen Kolleginnen und Kollegen des MRT möchte ich mich für die Zusammenarbeit, Anregungen in Diskussionen und Sommerseminaren, aber auch darüber hinaus für gemeinsame Unternehmungen, Social Tuesdays, Skifahrten und vieles mehr bedanken. Dr. Martin Lauer danke ich für die geduldige Unterstützung selbst bei den kleinsten Details. Besonderer Dank gilt Marilena Kammerer für gestalterische Unterstützung. Weiterhin danke ich den Heerscharen von Lektoren, die zum Gelingen beigetragen haben (in Reihenfolge der Kapitel), Nick Schneider, Janina Rehder, Maximilian Naumann, Nicolas Jourdan, Lukas Schneider, Martin Lauer, Jan Bernlöhr, Niels Ole Salscheider und Niklas Hanselmann.

Horst Klöden und Nina Brouwer von BMW danke ich für die fruchtbare Zusammenarbeit zu Anfang meiner Doktorandenzeit.

Dem Sekretariat des MRT danke ich für die organisatorische Unterstützung sowie den löwenhaften Einsatz für die Doktoranden im Angesicht von Verwaltungschaos. Außerdem danke ich den Werkstätten und Werner Paal für die technische Unterstützung. Der Karlsruhe School of Optics and Photonics (KSOP) danke ich für finanzielle Förderung und den spannenden Blick über den Tellerrand.

Außerdem danke ich dem gesamten Team von CV1, ohne deren Angebot diese Arbeit wahrscheinlich vier Jahre früher abgegeben worden aber ohne die sie danach auch niemals fertig geworden wäre.

Mein besonderer Dank gilt meinen Eltern für ihre gesamte Unterstützung. Ganz besonders danke ich meiner Frau Janina, für ihr Verständnis und ihre stoische Ruhe, wenn neben einer vollen Arbeitswoche noch eine Dissertation geschrieben wird.

Böblingen, im Juni 2021 *Eike Rehder*

Abstract

Perception and prediction of other traffic participants is crucial for safe and efficient navigation of automated vehicles. In this context, pedestrians require special attention due to their great vulnerability and complex motion patterns.

This thesis presents an end-to-end goal-directed prediction network for pedestrian prediction. For this, a probabilistic formulation is proposed that utilizes path planning techniques for prediction. The required goal is treated as a latent variable in form of a short-term destination. Two different realizations of goal-directed prediction are derived from this, one based on Markov Decision Processes and one inspired by the Forward-Backward-Algorithm. Using a grid representation, an efficient solution to the planning task using recursive convolutions is shown.

The proposed system consists of two parts, the inference of destinations and the planning towards them. Both components are implemented as neural networks. From time series of pedestrian observations, a recurrent mixture density network predicts distributions of destinations. These serve as the latent goals for the subsequent planning networks. Based on semantic grid maps of the environment, these networks estimate the trajectory towards the destinations in a probabilistic fashion. As all individual building blocks are fully differentiable, they can be fused and trained as a single, monolithic neural network from observed pedestrian behavior.

For training and evaluation, a naturalistic dataset consisting of video sequences is proposed. In these, pedestrians are annotated manually and enriched with trajectory information and local semantic grid maps. To assess performance, metrics are selected that reflect both, precision and safety of the prediction. With this at hand, a detailed analysis of the method's design parameters is provided, also in comparison to established approaches from literature. It is shown that the method performs best in all evaluated domains.

Kurzfassung

Die Erkennung und Vorhersage anderer Verkehrsteilnehmer ist eine Grundvoraussetzung für den erfolgreichen Betrieb automatischer Fahrzeuge. Wegen ihrer komplexen Dynamik und Verletzungsgefahr muss hierbei besonderes Augenmerk auf Fußgänger gelegt werden.

In dieser Arbeit wird ein System zur Vorhersage von Fußgängerbewegung auf Basis zielgerichteter Planung vorgestellt. Dieses integiert alle Bestandteile in einem einzigen probabilistischen Modell. Die hierfür zunächst notwendigen Ziele werden als latente Variable behandelt. Gegeben dieser Ziele werden zwei Lösungsansätze für die Pfadplanung präsentiert, basierend zum einen auf Markov-Entscheidungsprossen, zum anderen auf dem Vorwärts-Rückwärts-Algorithmus. Eine Repräsentation durch Rasterkarten erlaubt hierbei eine effiziente Berechnung mittels diskreter Faltungen.

Das vorgestellte System setzt sich aus zwei Einzelteilen zusammen, der Inferenz möglicher Ziele und der Trajektorienplanung dorthin. Beide werden jeweils als künstliche neuronale Netze formuliert. Die Vorhersage von Zielen erfolgt mithilfe eines rekurrenten Netzes, welches die Parameter einer Mischverteilung aus Fußgängerbeobachtungen inferiert. Diese dienen dann als Zielzustand für die jeweiligen Planungsnetzwerke. Auf Basis von Merkmalskarten der Umgebung erzeugen sie Wahrscheinlichkeitsverteilungen für die Zwischenzustände auf dem Weg zum Ziel. Da sowohl die Zielprädiktion als auch Planung differenzierbar sind, können beide Einzelteile als ein ganzheitliches neuronales Netz betrachtet und daher auch als solches trainiert werden.

Zur Evaluation wird ein Datensatz bestehend aus realen Verkehrsszenen vorgestellt. Hierfür werden zunächst Fußgänger in Videosequenzen manuell annotiert. Mittels Stereo-Bildverarbeitung werden diese zusätzlich mit Trajektorieninformation und semantischen Umfeldkarten angereichert. Um Prädiktion quantifizieren zu können, werden Metriken vorgeschlagen, die sowohl die Genauigkeit als auch die Sicherheit der Vorhersage widerspiegeln. Auf dieser Basis werden zunächst die modellspezifischen Parameter im Detail analysiert. Zuletzt werden etablierte Methoden aus der Literatur zum Vergleich herangezogen. Das vorgestellten Verfahren überbietet diese sowohl in Genauigkeit und Sicherheit.

Contents

Notation and Symbols

Acronyms

ADAM	Adaptive Momentum
AF	Activity Forecasting
ANN	Artificial Neural Network
AuPR	Area under Precision-Recall-Curve
AuROC	Area under Receiver Operator Characteristic
BEV	Bird's Eye View
Bwd	Backward
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DOGMa	Dynamic Occupancy Grid Map
ELU	Exponential Linear Unit
FC	Fully Connected (Layer or Network)
FCN	Fully Convolutional Network
Fwd	Forward
Fwd-Bwd	Forward-Backward
GPU	Graphics Processing Unit
IL	Imitation Learning
IMM	Interacting Multiple Model (Filter)
IR	Infra-Red
IRL	Inverse Reinforcement Learning
KF	Kalman Filter
Laser	Light Amplification by Stimulated Emission of Radiation
LRN	Local Response Normalization
LSTM	Long Short-Term Memory
MDN	Mixture Density Network
MDP	Markov Decision Process
mNLP	Mean Negative Log Probability
mPP	Mean Predicted Probability
PDF	Probability Density Function

PR	Precision-Recall-(Curve)
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMDN	Recurrent Mixture Density Network
RNN	Recurrent Neural Network
ROC	Receiver Operator Characteristic
SGD	Stochastic Gradient Descent
VGG	Visual Geometry Group
VI	Value Iteration
VIN	Value Iteration Network
VRU	Vulnerable Road User

General Notation

Scalars	Regular (Greek) lower case	a, b, σ, λ
Vectors	Bold (Greek) lower case	$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{\sigma}, \boldsymbol{\lambda}$
Matrices	Bold (Greek) upper case	$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Sigma}, \boldsymbol{\Lambda}$
Sets	Calligraphic upper case	$\mathcal{A}, \mathcal{B}$

Generic Mathematical Symbols

ϵ	A small constant
$\circ$	Arbitrary argument (of function or operator)
$\hat{\circ}$	An estimate
$\mathbb{1}$	Selector, conditionally evaluates to 0 or 1
$\otimes$	Convolution operator
$\langle \boldsymbol{A}, \boldsymbol{B} \rangle_F$	Frobenius Product of $\boldsymbol{A}$ and $\boldsymbol{B}$, $\sum_i \sum_j a_{ij} b_{ij}$
$J(\circ)$	Cost functional

Trajectories Representations

$s, \boldsymbol{s}$	An agent's state
a	An action
$\mathcal{A}$	Set of all possible actions
$u, \boldsymbol{u}$	Motion
ζ	Trajectory
N_ζ	Number or states in trajectory
t	A discrete time step
T	The last time step of a trajectory segment
$s_0, \boldsymbol{s}_0$	Current starting point of trajectory segment
$s_T, \boldsymbol{s}_T$	Current destination of trajectory segment

Neural Networks

$x, \boldsymbol{x}, \boldsymbol{X}$	Input data
$\hat{y}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{Y}}$	A network's output
$y, \boldsymbol{y}$	Target values (labels)
$\hat{w}, \hat{\boldsymbol{w}}, \hat{\boldsymbol{W}}$	Trainable parameters
$\boldsymbol{c}$	Cell state of an RNN
L	Loss function
η	Learning rate
λ	Weighting factor
$\mathcal{D}$	Dataset
$\mathcal{B}$	Batch of training examples

Representation of Predictions

Θ	Feature map of the environment
Φ	Grid map of predicted probability distribution
$R(\circ)$	Reward function in Value Iteration
$V(\circ)$	Value function in Value Iteration
$\pi(\circ)$	Policy in an MDP

Dataset Generation

$\boldsymbol{T}$	Pose in $SE(3)$
$\boldsymbol{R}$	Rotation matrix
$\boldsymbol{K}$	Camera projection matrix
u, v	Image coordinates
f	Camera's focal length
c_u, c_v	Camera's projection center in image
b	Stereo camera baseline
d	Disparity
$\pi(\circ)$	Projection function
$\boldsymbol{x}$	3D point in the world
$\boldsymbol{p}$	2D point in image
$\hat{e}, \hat{\boldsymbol{e}}$	Residual
$\mathcal{L}$	Set of landmarks
$\mathcal{T}$	Set of poses
l, r, ψ	Arc, radius and angle of curve
k	Height of a person
w, h	Dimensions of bounding box
o	Binary label of occlusion
g	Binary label of validity of disparity

Color Schemes

Throughout this work, consistent color schemes are used to visualize information. They can be grouped into three categories, namely color schemes to distinguish entities, color representation of semantic classes, and finally sequential colormaps for qualitative display of values.

Semantic classes are displayed according to a modified CityScapes color mapping [37]. The classes and their respective colors are displayed below.

Road	Sidewalk	Building	Pole	Traffic Light	Traffic Sign
Vegetation	Terrain	Sky	Person	Rider	Car
Motorcycle	Bike	Obstacle	Curb		

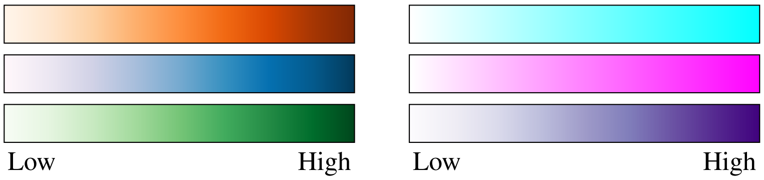
Colormap for semantic classes built upon CityScapes color mapping [37].

This work makes extensive use of artificial neural networks. These networks are constructed of recurring building blocks. Each of these blocks is used for distinct characteristics in processing. For description of the underlying concepts, refer to Appendix A. Due to the complexity of some of the models used in this work, the following color scheme is employed for simple visualization of building blocks.

Input	Convolution	Pooling	Matrix Multiply
Concatenation	Deconvolution	Elem.-wise Op.	Other

Color map to visualize network components. Color palette taken from [59].

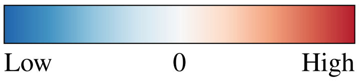
Real-valued results are visualized using sequential colormaps. The actual range of the mapped values varies depending on context. However, they are only meant for qualitative depictions of results.



Sequential colormaps for qualitative visualization. Color palette taken from [59].

Note that some of these colormaps are consistently but not exclusively attributed to visualize classes of methods throughout this thesis. Oranges are used for reference experiments, blues for RMDN and MDP, and finally, greens denote Fwd-Bwd results.

In some cases, it is necessary to identify zeros in visualization, e.g. for cost maps. For this, the following diverging colormap is used. For better visualization, positive and negative values may be scaled differently to make use of the whole color spectrum.



Diverging colormap for qualitative visualization with distinct zero color value. Color palette taken from [59].

Chapter 1

Introduction

Automated driving is considered one of the key technologies of future mobility. If it were possible to transfer the task of driving from the human to the vehicle itself, many challenges of future mobility could be addressed.

Automation of driving tasks can increase the quality of life in many aspects. It will help to reduce and mitigate traffic accidents. It will provide mobility for those that cannot or dare not safely operate a vehicle, such as the elderly or impaired. Furthermore, automation can ease tedious driving tasks, such as daily commute or congested traffic, providing more quality time for the driver. Lastly, it can help to optimize traffic flow for more efficient and eco friendly transport [7, 106].

1.1. Towards Automated Driving

The vision of automated driving is almost as old as the automobile itself. First ideas of driverless cars date back as early as the 1920s [115, 116]. With technological progress, automation quickly evolved from teleoperation towards in-vehicle sensing, processing and control [40, 81]. The two great challenges hosted by the Defense Advanced Research Program Agency (DARPA), the Grand Challenge and the Urban Challenge, introduced the field to a larger community [27, 28]. Since then, almost all vehicle manufacturers as well as soft- and hardware suppliers have established their own research programs.

As a first realization, advanced driver assistance systems have been introduced to production line vehicles successfully. Systems like adaptive cruise control or active lane assist support the driver in structured traffic.

(a) MERCEDES-BENZ S500 INTELLIGENT DRIVE [184]. Source: Mercedes-Benz

(b) WAYMO Driverless Car [118]. Source: Waymo

Figure 1.1.: Examples of successful automated vehicles in urban environments.

Meanwhile, the task of urban transport remains challenging. An automated vehicle that is to navigate through all urban scenarios has to fulfill the same tasks as a human driver. That is, it has to monitor the entire environment, obey to all traffic rules, detect other traffic participants and negotiate for interactions.

Recently, some works have started to approach automation in urban traffic, some examples shown in Fig. 1.1. However, due to the unstructured nature of the problem, these systems were only applied in limited scope [118, 184]. This is mainly due to the diversity of road scenes as well as the sometimes chaotic interactions between a great variety of traffic participants. These pose a severe challenge since the road must be shared not only with other automobiles but also with motorbikes, cyclists, pedestrians, wheelchairs, animals and many more.

1.2. Pedestrian Safety: A Challenge For Automated Driving

Among all the different traffic participants in cities, the safety of *vulnerable road users* (VRUs, i.e. cyclists, pedestrians and the like) should be of special concern as collisions may have severe, even fatal, consequences.

This becomes apparent from the accident statistics. In Fig. 1.2, we see the evolution of all fatalities in traffic accidents in Germany as well as that of pedestrians. In the past decades, the number of fatalities in traffic has steadily declined. This is a consequence of both, safety systems as well as stricter law enforcement.

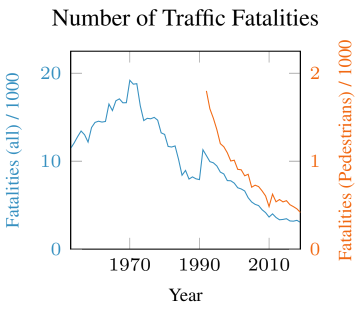


Figure 1.2.: Numbers of fatalities p.a. in traffic over time.

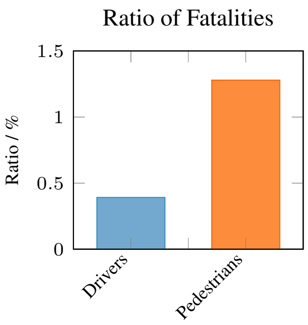


Figure 1.3.: Ratio of fatalities per accident and kind of traffic participation (2019).

If we analyze these accidents, the ratios of fatalities in Fig. 1.3 give a better insight. Of all pedestrians involved in traffic accidents, consequences were fatal for almost 1.3%. This number is three times as high as that for drivers. In total, even though pedestrians make up for only one tenth of accident participants, they account for one sixth of the fatalities. This, again, justifies the term of *vulnerable road users* since accidents often result in serious injuries or even death. All in all, it shows the demand for measures that directly address pedestrian safety.

Increased safety can be achieved through active systems. For this, vehicles are equipped with sensors that perceive the environment. These allow to detect VRUs and track them over time. A prediction of the VRU's behavior then serves as an input for vehicle motion planning, e.g. braking or evasive maneuvers. While detection and tracking has received considerable attention in research, prediction still lags behind. In order to fill this gap, this work focuses on the prediction with special focus on pedestrians.

Current prediction systems can be separated into three categories: those that reason based on object dynamics, those that employ planning methods and, finally, pattern recognition models. All come in great variety of forms and features, e.g. incorporation of additional knowledge such as body pose or previously observed scenarios. However, the same way that prediction approaches can be assigned to one of the three classes, they also exhibit one of three shortcomings. Either, they only apply for short time horizons of less than a second, they require previously observed, static scenes, or they a not physically founded.

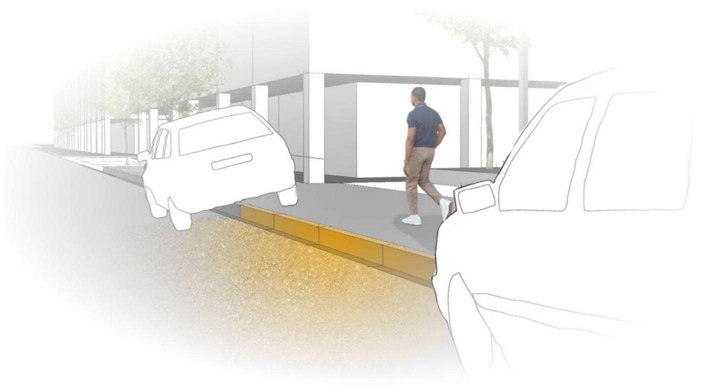


Figure 1.4.: Pedestrian at the curb: A multitude of factors influences prediction.

In the context of automated driving however, none of those drawbacks is acceptable. In order to navigate successfully a vehicle has to predict other traffic participants' actions for the range of seconds. Also, the constraint of static or previously observed scenes cannot be satisfied as vehicles move through dynamically changing environments. Finally, only dynamically feasible trajectories can be considered valid predictions.

1.3. Human Motion Is Decision Making

For time horizons of multiple seconds, pedestrian motion is no longer purely reactive, but tactical instead. Humans *plan* their motion in order to reach a specific goal. Mostly, this goal is to reach a certain destination. At the same time, the human is subject to extrinsic and intrinsic constraints. Extrinsic constraints cover all factors that influence motion from the outside, such as obstacles along the way, traffic regulations or the risk of collision. Intrinsic constraints stem from the human's limited dynamics, e.g. limits on pace or acceleration.

Figure 1.4 visualizes these three factors. The pedestrian at the curb faces a common urban scene with cars parked along a sidewalk. If we aim to predict his future motion, we first have to determine his intention. If he was headed along the sidewalk, his actions would be less likely to interfere with a vehicle on the road. If, however, he aimed to cross the road, a passing vehicle would have to take special care. For both cases, however, the pedestrian is subject to the constraints of the scene geometry. He cannot pass through the parked cars and will prefer to stay on the sidewalk.

Even though we cannot reason about his dynamics from a single image, we may assume from his posture that he is facing the curb stone, indicating a future trajectory closer to, or even on, the road. This simple example demonstrates the reasoning about pedestrian behavior that human drivers do and, consequently, an automated vehicle should be capable of as well.

Accordingly, automated prediction should reason about intentions and the consequent motion to fulfill them. As the intention may only be inferred from a VRU's behavior, a system should incorporate VRU-specific information such as past motion or visual cues. Especially for long-term prediction, the most likely future motion can be deduced from this intention. Naturally, this is also impacted by the layout of the scene. Consequently, a prediction system should incorporate all these features into its reasoning. Unfortunately, it is infeasible to model all relationships between features and a VRU's behavior explicitly. Instead, a prediction method should be able to infer these from past observations. As all predictions inherently are subject to uncertainty they need to be approached in a probabilistic fashion.

1.4. Thesis Outline

This work proposes to model prediction by means of probabilistic goal-directed planning. As seen in the previous example, the intention is of particular importance. Therefore, it is inferred from observations in form of short-term destinations. With such destinations given, planning techniques are employed to reason about future trajectories. Here, also knowledge about the environment is incorporated. The entire method is formulated as an artificial neural network which allows to learn the prediction from observed trajectories. For both, learning and evaluation, a naturalistic pedestrian dataset is created, using a stereo camera as only sensor. With this real world data, the impact of the method's design parameters is analyzed and finally, it is compared against established methods from literature.

In the following, the method for prediction is explained in detail. After this introduction, a brief review relevant fundamentals and related literature follows in Chapter 2 and Chapter 3, respectively. The informed reader may like to skip the two and continue with the generic statement of the goal-directed prediction framework in Chapter 4. The application to pedestrian trajectory prediction for automated vehicles is presented in Chapter 5. The system is evaluated on real world data captured by an experimental vehicle, described in Chapter 6. The results of this evaluation are explained in Chapter 7. The work is concluded in Chapter 8.

Chapter 2

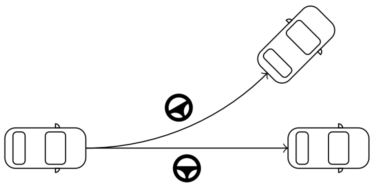
Fundamentals

Forecasting of future actions is inherently subject to uncertainties. Consequently, a probabilistic prediction model should be constructed. To make the most from observations it should incorporate features for both, a pedestrian and the environment. If we try to build such a model directly from expert knowledge, we quickly encounter difficulties in parameterizations of all relevant factors. With an increasing number of features and interdependencies it becomes infeasible to explicitly program a suitable predictor. In this case, machine learning is an attractive alternative. With powerful tools of machine intelligence at hand, the probabilistic model can be obtained by learning. For this, a machine should infer a prediction model from previously observed trajectories.

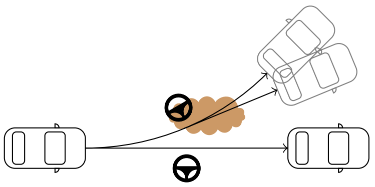
This chapter briefly discusses the fundamentals of probabilistic planning and machine learning used in this work. For probabilistic modeling of motion, Markov Decision Processes (MDPs) are explained briefly. As the machine learning backbone for the prediction model, artificial neural networks are employed. Since only a basic understanding is required, only the most general concepts are introduced. The interested reader is referred to Goodfellow *et al.* [56] and the list of network components in Appendix A.

2.1. Markov Decision Processes

An agent's navigation can be considered as selection of actions from a set of choices. This process can be modeled as a graph in which every state is represented as a node and every action is an edge in the graph.



(a) Vehicle maneuvering as deterministic decision graph.



(b) Due to uncertainty, e.g. unkown road friction, steering becomes a Markov Decision Process.

Figure 2.1.: Vehicle steering as a decision process, either deterministic or stochastic.

Let us first consider a simple example of a moving vehicle. We may model selected discrete positions as possible states. The action alternatives now represent the transitions between the states, e.g *driving straight*, *taking a left turn*, and so on, as displayed in Fig. 2.1a. In this formulation, the vehicle's motion can be seen as a simple decision graph. If a certain state should be reached, the actions leading to that state can be selected and executed deterministically.

This very simple model becomes more complicated when the outcome is not deterministic. This might be the case if the vehicle moves over terrain with varying friction, e.g. on gravel roads. Now, the wheel slip and thus the action outcome may be subject to the random nature of the ground, as illustrated in Fig. 2.1b. This has to be accounted for in transition selection.

For that, the selection of an action depends on the distribution of possible outcomes. The problem of decision graphs with uncertain transitions has been studied as Markov Decision Processes (MDPs). An MDP consists of a set of states $\mathcal{S}$ and a set of actions $\mathcal{A}$. To every state $s{\in}\mathcal{S}$, a transition probability distribution is associated to end up in a new state $s'{\in}\mathcal{S}$ when taking an action $a{\in}\mathcal{A}$ as

$$P_a(s, s') = P(s_{k+1} = s' | s_k = s, a_k = a). \tag{2.1}$$

Now, a reward $R_a(s, s')$ is defined for taking action a in s that ends up in state s'. The goal then is to find a policy per state $\pi(s)$ in order to maximize the expected reward.

One solution to find a policy for an MDP is *Value Iteration* (VI). A value function $V(s)$ is defined as the current expected reward per state when following the current optimal policy. In an iterative fashion, the value of every state $V(s)$ is updated with the maximum expected reward

$$V_{k+1}(s) \;:=\; \max_a \sum_{s'} P_a(s,s')\left(R_a(s,s') + \gamma V_k(s')\right), \qquad (2.2)$$

where γ is a discounting factor to favor early rewards. Here, again, every action a is associated with a state transition probability distribution as defined in Eq. (2.1).

In the expanded product of Eq. (2.2), the term $P_a(s,s')R_a(s,s')$ reflects the expected reward for a single action. The term $P_a(s,s')V_k(s')$ includes the previous result $V_k(s')$ and with that, resembles rewards that can later be obtained from future states s'. The final policy function is the argmax of the value function at convergence $V(s)$

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P_a(s,s')\left(R_a(s,s') + \gamma V(s')\right). \qquad (2.3)$$

Since these rewards reinforce a certain behavior policy that is learned iteratively, this task is called *reinforcement learning* (RL) [160].

In many cases the reward function $R_a(s,s')$ is defined by an external observer. Unfortunately, this reward function is not always known. If, however, policies or action outcomes of others solving the same problem can be observed, we may try to infer the underlying reward function. This concept is studied as Imitation Learning (IL) or Inverse Reinforcement Learning (IRL). For IL, the aim is to imitate demonstrated actions [132]. In IRL, we observe a series of action outcomes and from this, infer the reward function [99].

2.2. Artificial Neural Networks

The field of machine learning deals with the task of artificially gaining knowledge from experience. Especially in computer science, this task is of great relevance as it lets a computer perform tasks without being explicitly programmed to do so [21].

In most current implementations of machine learning, this is done by modeling a function that maps an input $\boldsymbol{x}$ to a desired output $\hat{\boldsymbol{y}}$ using a set of parameters $\hat{\boldsymbol{w}}$ [21]

$$f\left(\boldsymbol{x}, \hat{\boldsymbol{w}}\right) \mapsto \hat{\boldsymbol{y}}. \tag{2.4}$$

In this context, the function $f(\circ)$ and the parameters $\hat{\boldsymbol{w}}$ represent a machine's knowledge, whereas the input $\boldsymbol{x}$ is the data to be reasoned upon.

Analogous to human learning, machine knowledge is built up from experience, i.e. some kind of feedback. In human learning this might be rewards or penalties given by some authority like parents or teachers, but also the experience of success or failure. In machine learning, it has to be modeled by a feedback function

$$J\left(f\left(\boldsymbol{x}, \hat{\boldsymbol{w}}\right), \boldsymbol{y}\right) \tag{2.5}$$

that compares the machine's output $\hat{\boldsymbol{y}}$ to a desired value $\boldsymbol{y}$. These so-called target values may be fed to the machine by a human user in *supervised learning*. This is analogous to a human being taught. If the target values can be computed by the machine itself, it learns in an *unsupervised* fashion, just like a human does from experience.

One way or the other, the function $J(\hat{\boldsymbol{y}}, \boldsymbol{y})$ will compute a feedback value that is used to update the parameters $\hat{\boldsymbol{w}}$ to improve the prediction for the given data. This process, called *training*, requires a pair of input vector and desired output $(\boldsymbol{x}, \boldsymbol{y})$. The input is fed to the function $f(\boldsymbol{x}, \hat{\boldsymbol{w}})$ and produce an output $\hat{\boldsymbol{y}}$. With the desired output $\boldsymbol{y}$, the function $J(\hat{\boldsymbol{y}}, \boldsymbol{y})$ can be evaluated and finally the parameters $\hat{\boldsymbol{w}}$ are changed to improve the outcome.

The output of function $J(\circ)$ can be seen as either *reward* or *loss* for the machine under training. For loss functions, small values are desirable. The machine learning problem is defined as a minimization task w.r.t. the parameters $\hat{\boldsymbol{w}}$, so that

$$\hat{\boldsymbol{w}}^{*} = \underset{\hat{\boldsymbol{w}}}{\operatorname{argmin}}\, J\left(f\left(\boldsymbol{x}, \hat{\boldsymbol{w}}\right), \boldsymbol{y}\right). \tag{2.6}$$

The use of a reward function only differs in equation Eq. (2.6) such that $J(\circ)$ is maximized instead.

2.2.1. Artificial Neurons

Artificial Neural Networks (ANN) are a special class of learning algorithms that were designed to mimic biological information processing [5]. In animate beings, information is processed and transmitted via neurons.

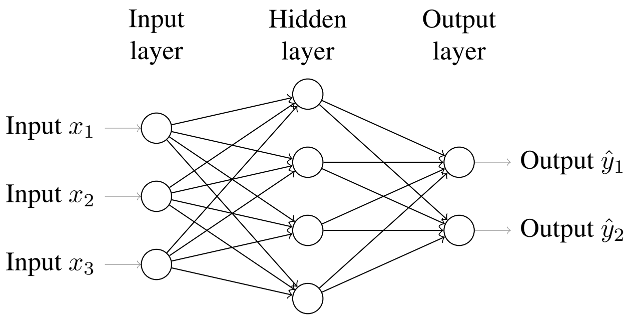


Figure 2.2.: An artificial neural network with three input nodes, one hidden layer with four neurons and two outputs.

These neurons are electrically excitable cells that connect to other cells via synapses [2]. In artificial neurons, this prototype is emulated by multivariate non-linear functions. This work makes use of the multilayer perceptron and variants thereof. The perceptron computes a weighted sum of a multitude of inputs to model the stimuli of a biological neuron. A non-linear *activation function* is applied to the result of this weighted sum of inputs to compute the excitation of the artificial neuron

$$f(\hat{\boldsymbol{w}}^\top \boldsymbol{x}) = \hat{y}, \tag{2.7}$$

where $\hat{\boldsymbol{w}}$ is a weighting vector for the input $\boldsymbol{x}$ and $f(\circ)$ is some non-linear function. The output $\hat{y}$ is called *activation* of a neuron.

To construct a network from artificial neurons, they are interconnected both in series as well as in parallel. For parallel processing, one simply exchanges the weight vector $\hat{\boldsymbol{w}}^\top$ in Eq. (2.7) by a weighting matrix

$$f(\hat{\boldsymbol{W}}\boldsymbol{x}) = \hat{\boldsymbol{y}}, \tag{2.8}$$

where $\hat{\boldsymbol{W}} \in \mathbb{R}^{\mathrm{M}\times\mathrm{N}}$, with N is the number of input vector entries and M the number of output neurons. This concept is called a *layer*. Multiple layers can be stacked by using the output of one layer as the input of another. The final output layer of an ANN is the predicted value of an ANN given the input data $\boldsymbol{x}$. The first layer that represents the input data points is called *input layer*, the last layer that outputs the predicted values *output layer*, all remaining layers are *hidden layers* as their activations are only used within the network's processing and remain hidden from the user. An exemplary network is shown in Fig. 2.2.

$$\begin{tikzcd}
\boldsymbol{x} \arrow[r, "\boldsymbol{x}"] & f \arrow[r, shift left, "f(\boldsymbol{x})"] \arrow[d, shift left, "\nabla_{\hat{\boldsymbol{w}}_1} J"] & g \arrow[l, shift left, "\nabla_f J"] \arrow[r, shift left, "g(f(\boldsymbol{x}))"] \arrow[d, shift left, "\nabla_{\hat{\boldsymbol{w}}_2} J"] & J \arrow[l, shift left, "\nabla_g J"] \\
& \hat{\boldsymbol{w}}_1 \arrow[u, shift left, "\hat{\boldsymbol{w}}_1"] & \hat{\boldsymbol{w}}_2 \arrow[u, shift left, "\hat{\boldsymbol{w}}_2"] &
\end{tikzcd}$$

Figure 2.3.: Data flow in a artificial neural network training. In the forward pass, input data $\boldsymbol{x}$ is processed by all layers to produce output g and its loss J. For parameter update, the loss gradient is propagated back to layers and their weights.

Such models are so-called fully connected (FC) networks. The name stems from the fact that for a given layer, there exist weights that couple every output neuron from the previous layer to every neuron in the given layer, thus fully connecting all neurons. This of course is only true if matrix $\hat{\boldsymbol{W}}$ is densely populated.

The number of layers in ANNs is called its *depth*. Early ANNs mostly were shallow networks with only two or three layers. Today, networks with hundreds of layers have successfully been trained [62].

2.2.2. Network Training

Given a neural network architecture, only its parameters Eq. (2.4) govern the mapping from in- to output. These parameters are found using minimization of a loss function according to Eq. (2.6). For this training process, non-linear optimization is used to find the parameters $\hat{\boldsymbol{w}}$ that best fit the desired output in terms of loss.

The most common optimization technique for neural networks is gradient descent. In gradient descent, parameters are iteratively updated to decrease the loss function's value. Given the current parameters $\hat{\boldsymbol{w}}_k$, we compute the gradient of the loss function J w.r.t. the parameters $\hat{\boldsymbol{w}}$, $\nabla_{\hat{\boldsymbol{w}}} J$. The parameters are then updated in opposing direction of the gradient,

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k - \eta \nabla_{\hat{\boldsymbol{w}}} J|_{\hat{\boldsymbol{w}}_k}, \tag{2.9}$$

where the parameter η, called *learning rate*, controls the scale of parameters updates per iteration.

For training, the gradient $\nabla_{\hat{\boldsymbol{w}}} J$ needs to be computed for all pairs of input data and desired output, $(\boldsymbol{x}, \boldsymbol{y})$. This is done efficiently when treating the neural network as a computation graph. An example is shown in Fig. 2.3. Every layer in the network is represented as a node in the graph.

In training, data is passed forward along the edges and processed in the nodes. The final node evaluates the loss of the network. Then, the gradient of the loss w.r.t. the network's output is fed back to the network. In every layer, it is split into the gradient w.r.t. the layer's parameters and its input. The gradient w.r.t. the parameters is used as update in Eq. (2.9) while the one w.r.t. the inputs is propagated back along the graph. Therefore, this algorithm is called back propagation [145]. It requires all operations to be differentiable w.r.t. their inputs and parameters.

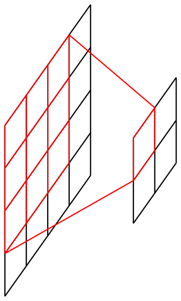
Every parameter update according to Eq. (2.9) requires processing of all examples in the dataset $\mathcal{D}$. Since a single training might require millions of iterations and datasets can consist of millions of individual data points, this may become impractical. Instead, the gradient can be evaluated on small subsets $\mathcal{B}$ of the full dataset $\mathcal{D}$. For every iteration, a subset $\mathcal{B}$, called batch, is drawn from $\mathcal{D}$ and the update is computed as

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k - \eta \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{B}} \nabla_{\hat{\boldsymbol{w}}} J(\boldsymbol{x}, \boldsymbol{y})|_{\hat{\boldsymbol{w}}_k}. \tag{2.10}$$

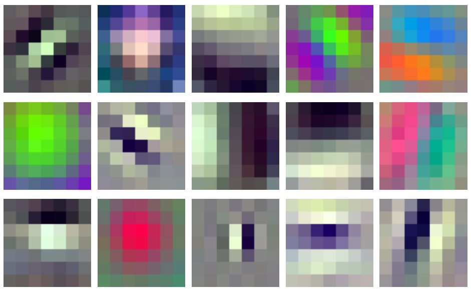
Due to the random composition of $\mathcal{B}$, Eq. (2.10) is called Stochastic Gradient Descent (SGD). The number of elements $|\mathcal{B}|$, the batch size, becomes a training hyper-parameter that reflects a trade-off between computational demand and convergence speed per iteration [24]. To further improve robustness and convergence, variants of SGD have been proposed. For example, parameter updates can be done with a weighted sum of the current gradient and the previous update. This so-called momentum alters the update towards the predominant direction of recent gradients [145]. The adaptive momentum (ADAM) adjusts the weighting of the sum depending on first and second moments of the gradient [85].

2.2.3. Convolutional Neural Networks and Deep Learning

Especially in the context of image processing where the input data possibly consists of millions of pixels, fully connected networks become infeasible. This is due to the fact that for each layer with input size N and output size M, a set of $N \times M$ parameters is required. For large values of M and N or very deep architectures, the complexity quickly explodes and thus, training becomes increasingly difficult. Moreover, FC networks use the naive assumption that every neuron should draw information from every preceding one individually.



(a) A convolutional neuron draws its activation from a local neighborhood.



(b) Examplary filters from first convolutional layer of the Inception network for image classification [157, 161].

Figure 2.4.: Convolutional neurons to detect local patterns. For images, early filters reflect edges or color blobs. Later stages combine them to abstract concepts, e.g. eyes or wheels, that eventually let the network identify objects.

In structured data such as images, however, local patterns are often highly correlated and translated arbitrarily. Thus, identification of such patterns together with fully connected networks can greatly improve network performance.

This train of thought gave rise to the concept of Convolutional Neural Networks (CNNs) [90, 94]. In these networks, instead of fully connecting one layer to the next, the response of a layer is computed from convolutions of an image $\boldsymbol{X}$ with trainable filter masks $\hat{\boldsymbol{W}}_c$

$$\hat{\boldsymbol{Y}}_c = f(\hat{\boldsymbol{W}}_c \otimes \boldsymbol{X}), \tag{2.11}$$

where the subscript $c \in 1, \ldots, C$ denotes the *output channel* of the convolution. The result is a new image $\hat{\boldsymbol{Y}}$ with C channels.

In convolutional layers, every artificial neuron draws information only from a limited area. An example is shown in Fig. 2.4a. Every neuron computes its activation using the same weights but a different neighborhood. The area of influence that is used to compute a neuron's activation is called *receptive field*, in analogy to biological sensing [73]. Since every filter responds to a particular local pattern, the channels represent different such patterns. Some example filters from the first layer of the Inception network for image classification are shown in Fig. 2.4b [157, 161]. During training, they developed detectors for edges, corners or color blobs.

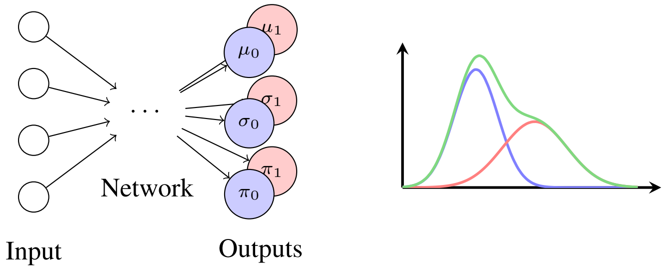


Figure 2.5.: Mixture Density Network: groups of three output nodes represent the two parameters of a Gaussian Distribution and the corresponding mixing coefficient (blue and red). Together, they can be interpreted as a Mixture of Gaussians (green).

2.2.4. Network Concepts

The decrease of variable parameters as well as the availability of sufficient training data and computational capacities now facilitate the use of very deep architectures. While early ANNs featured only few layers, today, networks with hundreds of layers have been trained successfully. This development, now called *Deep Learning*, resembles a set of functions, architectures and tools that enable training and deployment of deep neural networks in a multitude of applications.

In this work, numerous deep learning techniques have been applied. While the list of all would exceed the scope of this chapter, the most important ones will be explained here briefly. For a complete overview over the methods used in this work, see Appendix A. Due to the modular structure of ANNs, most techniques can be combined in a single network.

Mixture Density Networks ANNs can be used to estimate parameters of conditional probability density functions (PDFs) [20, 78]. Through appropriate activation functions applied to the output nodes individually, constraints imposed by PDFs can be met. For a univariate Normal Distribution, for example, two output neurons can predict the two parameters, namely mean and variance. Since there is no restriction on the mean value, a linear activation function can be used. The constraint of a strictly positive variance can be met by an exponential function as activation of the second neuron. With multiple of such outputs, mixture density functions can be modeled, as shown in Fig. 2.5. Apart from all parameters of the base distributions, the mixing coefficients have to be predicted by the network, which is achieved with the softmax (cf. Section A.2.4) as activation function [175].

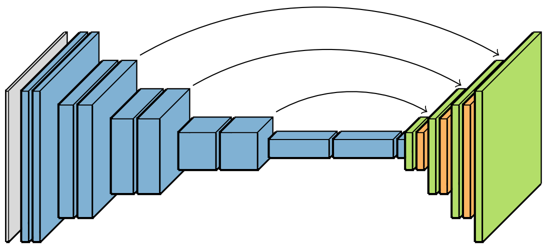


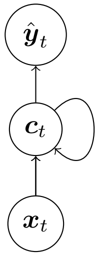
Figure 2.6.: Fully convolutional architecture for pixel-wise prediction: an input image (gray) is fed through an encoder consisting of multiple convolution and pooling stages (blue). For the output image, low scale predictions are successively upsampled (green) and combined with predictions from higher scale feature maps (arrows) by a simple sum (orange).

Pretraining For many tasks, well established architectures exist. Thus, in application of CNNs, researchers may want to use a given architecture for a new problem. Since CNN training is demanding in both, computational effort and data, optimization of all parameters from scratch is rarely an option. However, since ANNs in general are extremely modular, the stump of a network can be repurposed. For that, the architecture and weights of an existing network are copied into a new one. Then, only the top layers have to be exchanged for the new purpose. The newly augmented network is then trained with the relevant data and small learning rates. Depending on the perspective, this is called *pretraining* or *finetuning* [42].

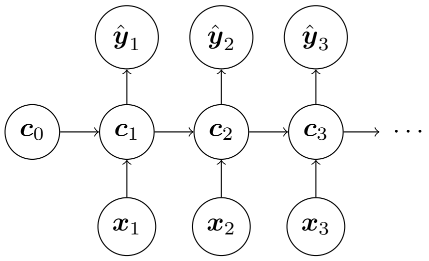
Fully Convolutional Networks CNNs are commonly employed for classification or regression of few output variables. However, sometimes it may be desired to reason about every pixel within an image. *Fully Convolutional Networks* (FCNs) have been applied to solve this task [101]. For this, predictions are computed as convolutions rather than matrix multiplications.

Since most architectures feature sub-sampling, fully convolutional networks have to invert their effect in order to predict at full input resolution. For this, either trainable transposed convolutions can be employed or feature layers are upscaled bi-linearly. Higher resolution features can be added for final predictions by bypassing the sub-sampling.

Fig. 2.6 shows an example architecture as proposed by Long *et al.* [101]. The network encodes features from the image, including multiple subsampling stages (blue). The final predictions are upsampled in steps (green).



(a) Neural network with recurrent element,



(b) Recurrent element unrolled in time,

Figure 2.7.: Recurrent neural networks for time series processing

For every step, the upsampled results are combined with predictions based on the same resolution encoder layer (arrows) as an addition (red). Networks like this have been applied to tasks like sliding window object detection, semantic segmentation, optical flow computation, and many more [44, 100, 101].

Recurrent Neural Networks Classical neural networks feature a static shape and, thus, can only deal with fixed input dimensions. Due to this limitations, they are not suitable for application to data of variable length. Unfortunately, for sequential data, variable length is a key aspect, e.g. in speech recognition or stock market prediction. To overcome the limitations of static neural networks, an internal memory can be introduced.

An ANN that can remember past inputs is inherently able to deal with variable sequence lengths. Since for this, the memory of a network couples back to its own output and memory, these networks are called *recurrent neural networks* (RNNs) [67, 70].

Unfortunately, RNNs are significantly harder to construct for gradient propagation. This is due to the recoupling of the network into its own state as shown by cyclic data flow in Fig. 2.7a. To cope with this challenge, *back propagation through time* or *time unrollment* has been proposed. In training, the recurrent stage of an RNN is replicated as many times as there are entries in the input sequences. Fig. 2.7b shows the unrolled network that now only exhibits non-cyclic connections. The neuron weights are shared between all replicas. This way, an RNN can be trained in standard backpropagation but still incorporate knowledge gained over time [67].

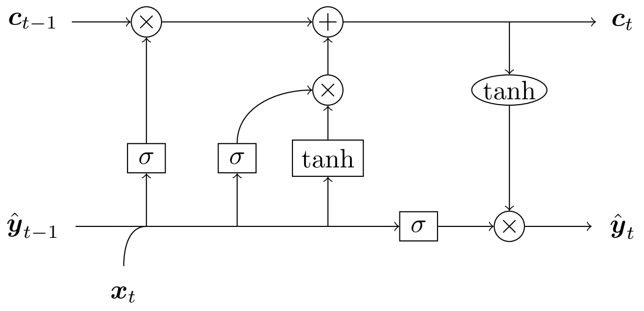


Figure 2.8.: Long Short-Term Memory cell [67]. Four layers process past output $\hat{\boldsymbol{y}}_{t-1}$ and current input $\boldsymbol{x}_t$ to update the cell state from $\boldsymbol{c}_{t-1}$ to $\boldsymbol{c}_t$ and produce output $\hat{\boldsymbol{y}}_t$. Layers are shown rectangular, element-wise operations rounded, σ denotes sigmoidal activations. See text for more information.

Long Short Term Memory The central component of an RNN is its memory cell. One special case of such memory cells is the Long Short Term Memory (LSTM) [67]. Its central design concept is gating of information from past state, current update and the output.

Fig. 2.8 shows the structure of an LSTM cell. A new data input at time t, $\boldsymbol{x}_t$, is concatenated with the past output $\hat{\boldsymbol{y}}_{t-1}$ to update the cell's state $\boldsymbol{c}_{t-1}$. First, a layer with sigmoidal activation serves as the so-called forget gate. Its output is multiplied with the last state $\boldsymbol{c}_{t-1}$ to only retain relevant information. Next, the cell's state update is computed from a layer with $\tanh$ activation. It is added to the current state, again gated by a sigmoidal layer, the input gate. Finally, the cell's output is the $\tanh$ of the new cell state $\boldsymbol{c}_t$ with an output gate applied to it. Variants of the LSTM exist, e.g. *peepholes* in which the gates have access to the cell state [53, 149] or even merging gates and states [34].

Regularization Neural networks with large numbers of parameters are prone to memorizing training data rather than generalizing to the underlying concept. This is especially true for very deep architectures with millions of parameters. This phenomenon is known as overfit in machine learning.

Overfitting can be explained with the simple example of polynomial fitting. Let us consider a set of N x-coordinates $x_i, i{=}1, \dots, N$, each associated with a y-coordinate.

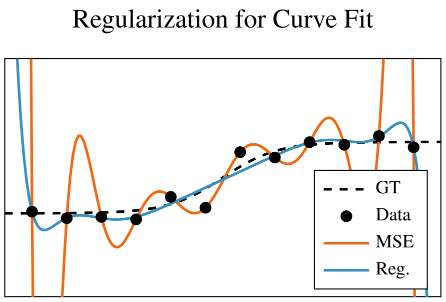


Figure 2.9.: Effect of regularization on polynomial fits. A sigmoidal function (dashed black line) was used to generate noisy data (black dots). Fitting a polynomial of order 11 (orange line) leads to perfect prediction of points but poor approximation of the underlying function. Regularization can mitigate this (blue line) to yield better generalization.

Figure 2.9 illustrates this problem. For this, data was generated from a ground truth function (black dashed line) with additive random noise (black dots). With these as input, a polynomial of order M

$$\hat{y} = \sum_{j=0}^{M} \hat{w}_j x^j \tag{2.12}$$

is fitted to the data to predict y-values from x-values by minimizing the mean squared error between estimate $\hat{y}$ and ground truth y,

$$\hat{\boldsymbol{w}}^* = \operatorname*{argmin}_{\hat{\boldsymbol{w}}} \frac{1}{N} \sum_{i=1}^{N} \|\hat{y}_i - y_i\|_2^2 \,, \tag{2.13}$$

where $\hat{\boldsymbol{w}}$ represents the parameter vector $(\hat{w}_0, \hat{w}_1, \ldots)^\top$. Every data point imposes one constraint on the fit. As long as M is smaller than $N{-}1$, we can never satisfy the constraints of all data points. However, as soon as M equals $N{-}1$, the polynomial will perfectly match every data point. In the presence of noisy training data, this leads to poor prediction results. This can be seen from the orange line in the example in Fig. 2.9 that oscillates around the desired result.

A countermeasure to overfitting is to reduce the parameter count. In very deep neural networks, however, this may not be an option. Instead, a trait present in most overfits can be used: to perfectly match all data points, parameters will have to take large values.

Large parameter values can be inhibited by adding their squared sum as a *regularization* term to the optimization function

$$\hat{\boldsymbol{w}}^* = \underset{\hat{\boldsymbol{w}}}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=1}^{N} \|\hat{y}_i - y_i\|_2^2 + \lambda \hat{\boldsymbol{w}}^\top \hat{\boldsymbol{w}} \right] , \tag{2.14}$$

where $\lambda > 0$ is a weighting factor [50]. This way, large parameter sets can be used while still training a model that generalizes well. The same function that produced the orange line in Fig. 2.9 can now approximate the black line much smoother (blue). In deep neural networks, those trained without regularization may not even converge at all [90].

Regularization with the squared sum of parameters (L_2-Regularization) is not the only way to control the optimization process. Many other regularization schemes exists, mostly designed for specific optimization problems, e.g. activation regularization [107] or random dropping of neurons [65].

Chapter 3

Related Work

Prediction of pedestrians has been addressed from various perspectives in both, applications and algorithms. Following the taxonomy of Rudenko *et al.* [143], categorization can be done from two different points of view. Once w.r.t. the prediction model and once w.r.t. the type of input features those models use. This allows to separate the actual prediction algorithm from the information available to it. This may seem counter-intuitive since algorithms clearly depend on their input. However, many methods can be applied to different sources of information and vice versa.

The overview of the relevant literature will begin with the most commonly used input features. Then, the different paradigms of how predictions are generated are presented. At this point, no assessment of their benefits or shortcomings is provided. Instead, representative methods are selected that will be reviewed in detail. They will later serve as a baseline for the experiments in Chapter 7. For a deeper insight into the state of the art, the reader is referred to existing surveys [26, 33, 66, 91, 93, 96, 110, 112, 136, 143].

3.1. Features

Prediction of the future is always based on current observations. Schmidt *et al.* showed that humans, for their own forecasts, rely on person-specific observations such as head orientation, pose or gate, but also on environment information like traffic situation and road layout [150, 151]. Accordingly, computational predictions rely on similar input features that cover both, observations of the agent and the situation.

3.1.1. Pedestrian-Specific Features

In order to predict a person's motion, past behavior of that person has to be observed. As forecasting of motion is the goal, position information is the most obvious feature. In processing of time series, this can manifest in sequential observations of just positions [86, 129, 130, 133, 137, 182] or positions together with velocities [23, 88, 89, 152].

Beyond pure motion, many prediction works utilize estimates of a person's head orientation [12, 60, 88, 89, 129, 130, 131, 140, 154, 167]. Furthermore, body posture provides valuable information. Thus, body orientation [23, 137] or even full joint configuration are used [108, 125]. Not only posture, but also semantic attributes can aid prediction, e.g. age or gender [14, 104]. Of course, the most information can be drawn from raw sensor measurements [87, 103, 133].

3.1.2. Environment Features

Motion of an agent is closely tied to its surrounding. Therefore, not only the agent but also the environment itself plays a fundamental role in these interactions. As such, it can be integrated as a source of information in prediction methods. Many works have utilized maps of free space [64, 69, 74, 122, 123, 139, 141, 169, 182], semantic properties of the environment [38, 82, 86, 129, 130, 133, 134, 135, 156, 162, 168] and relations to scene attributes, e.g. distance to curbs [23, 86, 89, 182]. Not only static surrounding contributes to behavior, but also other dynamic agents, including interactions between them [6, 47, 63, 69, 89, 102, 121, 165, 170]. Finally, points of interest can indicate predictable motion patterns, e.g. cross-walks [17, 82, 86, 123, 141, 182].

3.2. Methods

Regardless of the contextual information at hand, a variety of actual prediction algorithms has been proposed. These can roughly be clustered into three groups, namely approaches that try to model the physical processes of motion, pattern recognition methods that employ statistical learning, and planning-based techniques that aim to reason about the underlying decisions of navigation [143].

3.2.1. Physical Model

Since motion is bound to the laws of physics, it is justified to model prediction by studying these physical processes. In most approaches, motion is understood as a dynamical model where the model's parameters are estimated from observations. For prediction, the dynamics are then extrapolated into the future.

Probably the largest group of such methods are Recursive Bayesian Filters. Amongst these, the Kalman Filter (KF) and Extended Kalman Filter are the most commonly used. Their application varies in the motion model, e.g. constant velocity or constant turn, as well as observations [10, 11, 16, 19, 48, 83, 109, 152]. Also, multiple filters have been used in interaction to cope with rapidly changing behavior [83, 89, 92, 154]. The selection of filters can also be based on other observations beyond pure motion [54, 89, 88]. Apart from KFs, particle filters are another common prediction concept [18, 29, 57].

3.2.2. Pattern Recognition

In contrast to physical models, pattern recognition methods aim to solve the prediction task by means of statistical learning. Different machine learning algorithms have been used to regress trajectories from observations, e.g. clustering [13, 32, 178], Gaussian Process Dynamical Models [108, 125], or neural networks [6, 41, 69, 133, 137].

In some cases, not trajectories but discrete behavior patterns may be of interest, e.g. *will someone cross the street?* This gives rise to classification-based prediction [23, 83, 87].

3.2.3. Prediction by Planning

Especially for long term forecasting, the goal-directed nature of human motion can be incorporated in prediction models. Prediction then no longer is solved by extrapolation of observations but as path planning comparable to that in robot navigation.

Path planning is typically solved as an optimization task in which a path of minimum cost is to be found. To apply this to prediction, a person's optimization function needs to be inferred. Some methods use static cost functions for all pedestrians [130, 169, 179] while others infer them from observations of the scene and human behavior [71, 86, 104, 133, 182]. Also, motion of other agents in the scene can be incorporated [142, 144, 168].

3.3. Reference Methods

The previously presented body of literature provides the context of this work. As representative reference algorithms, a set of methods is selected to study the individual paradigms and their implications in more detail.

Embodiments for all of the three concepts are studied in detail, i.e. physical models, pattern recognition, and prediction by planning. To put the more elaborate algorithms into perspective, the Kalman Filter (KF) serves as a naïve baseline [16, 19]. The interacting multiple model filter (IMM) is reported to improve upon pure KF prediction [152]. Thus, it serves as the representative physical prediction model. With the recent success of Convolutional Neural Networks (CNNs), it makes sense to employ them to prediction. We choose direct learning of future occupancy grids as exemplary architecture [69]. Finally, inverse reinforcement learning in the form of Activity Forecasting can be seen as the basis of all planning-based prediction techniques and, hence, is included in the study [86].

In the following, we will dive into each of these algorithms in more detail. Their basic principles of operation are explained as well as implementation considerations. Finally, an experimental evaluation and comparison of the four methods is provided.

3.3.1. Naïve Baseline: Kalman Filter

One of the most widely used prediction schemes is recursive Bayesian filtering. The Kalman Filter (KF) is one specialization of such systems that is well-established for state estimation [22, 79]. The KF serves as naïve baseline. A constant velocity model is employed, i.e. with state $\boldsymbol{s} = (x, y, \dot{x}, \dot{y})^\top$, random noise $\boldsymbol{v}_t$ and the state transition as

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} + \Delta t \begin{pmatrix} \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{pmatrix} + \boldsymbol{v}_t, \quad \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \end{pmatrix} = \begin{pmatrix} \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{pmatrix}. \tag{3.1}$$

The ground truth positions of pedestrians are used as observations in the filter's update step.

The variable parameters of the KF are the covariance matrices of initial state, process noise, and measurement noise. To achieve best prediction, they are optimized for extrapolation results. Given ground truth pedestrian tracks, the KF is run as tracking. At every time instance, it is used to predict the state into the future for a fixed time horizon. Then, the negative log-likelihood of the ground truth future position w.r.t. the filter's parameters is minimized.

3.3.2. Physical Model: Interacting Multiple Models

The Kalman Filter can only represent unimodal distributions. To pay respect to multimodalities, a set of KFs can be applied simultaneously for different motion hypotheses. Mixing coefficients reflect in which motion state the agent is believed to be in. Following [152], an interacting multiple model filter (IMM) is employed with two separate motion models. Driven by the Kalman Filter experiments, the constant velocity (cv) model as introduced in Chapter 3.3.1 is selected as the first, constant position (cp) as the second motion model, i.e. standing, with $x_t{=}x_{t-1}, y_t{=}y_{t-1}, \dot{x}_t{=}\dot{y}_t{=}0$.

The variable parameters include those of two filters, their initial covariance matrices and their process noise. Only a single measurement covariance matrix is needed as observations are independent of the dynamical model. Finally, an initial estimate of the two motion state mixing variables π_{cv}, π_{cp} is required, where $0 \leq \pi_{cv}, \pi_{cp}$ and $\pi_{cv} + \pi_{cp} = 1$ and the model transition function

$$\begin{pmatrix} \pi_{cv,t} \\ \pi_{cp,t} \end{pmatrix} = \begin{bmatrix} p_{cv\to cv} & p_{cp\to cv} \\ p_{cv\to cp} & p_{cp\to cp} \end{bmatrix} \begin{pmatrix} \pi_{cv,t-1} \\ \pi_{cp,t-1} \end{pmatrix}, \tag{3.2}$$

where $p_{\{cv,cp\}\to\{cv,cp\}}$ are the transition probabilities that a person changes from one motion state to the other. The prediction step includes mixing of the individual filter states. Parameter tuning is done the same way as for the Kalman Filter to optimize for best possible prediction by minimizing negative log-likelihood.

3.3.3. Pattern Recognition: Dynamic Occupancy Grid Prediction

Pattern recognition systems rely on learning prediction from observed data. The work of Hörmann *et al.* realizes this as a CNN that predicts future occupancy grid states from a dynamic occupancy grid map (DOGMa) [69]. The map used in this work contains four features per grid cell, namely occupancy with static and dynamic objects together with north- and eastbound velocities [120]. Static occupancy provides information about accessible areas and obstructions while dynamic cells can be used to infer motion and even interactions between agents.

For evaluation, the DOGMa from [69] is replicated. The static occupancy grid maps are generated from stereo imaging. For details, the reader is referred to Chapter 6.1.5. The dynamic part of the map is simulated from ground truth: at every ground truth pedestrian's position, dynamic occupancy and velocities are inserted, covering a circular area of 0.15m^2 [174].

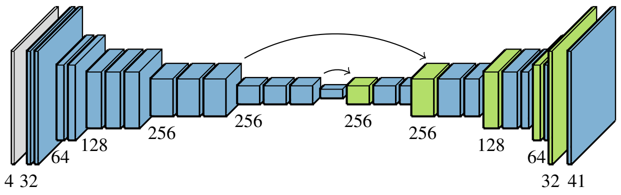


Figure 3.1.: Network architecture for learned grid prediction with a CNN [69]. The decoder uses learned deconvolutions (green) in the upsampling path [119]. The configuration of channel count and skip connections was found through hyperparameter search.

The CNN architecture is an encoder-decoder scheme [119] with adjustments as stated in [69]. It uses a mirrored architecture where every downsampling stage in the encoder network has a same-sized counterpart in the upsampling part. As in [69], bypass connections are added that directly connect same-resolution layers from down- to upsampling. The bypasses are added to the upsampled features according to [97]. The complete architecture is depicted in Fig. 3.1.

As output, the network produces one layer of dynamic occupancy for every predicted time step together with one layer for static occupancy. Thanks to sigmoidal activation, each output can be interpreted as a cell-wise occupancy probability. In training, the mean squared error between ground truth and predicted maps is used as loss. Weightings are applied to compensate the class imbalance between static and dynamic predictions. Since the original publication is not specific about the exact architecture and training details, hyperparameter search is run for all relevant parameters. This also includes variations of the network architecture to achieve the best possible performance for fair comparison.

3.3.4. Prediction by Planning: Activity Forecasting

Any moving person can be seen as an intelligent decision maker planning its actions to reach a goal. Thus, prediction can be interpreted not only as a problem of extrapolation into the future but as a planning task instead. Activity Forecasting (AF) of Kitani *et al.* [86], building up on Ziebart *et al.* [182], learns to infer behavior policies from scene observations. Prediction is then done by simulating these policies.

The prediction framework operates on top views of static scenes that are discretized into a regular grid of small cells. Using semantic segmentation [111], a vector of features $\boldsymbol{f}(s)$ is generated per grid cell s.

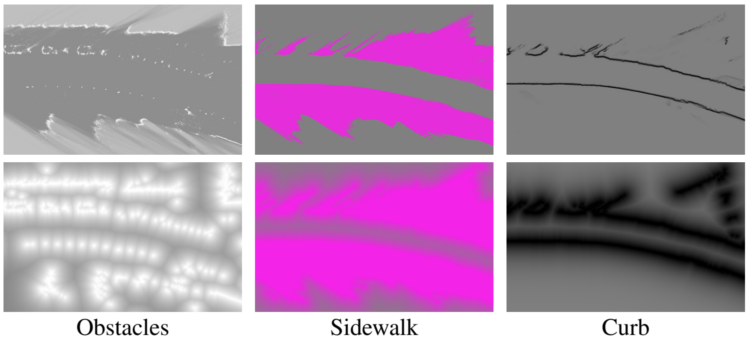


Figure 3.2.: Feature maps and an exemplary exponential distance map, as used in [86]. For details on feature mapping, see Section 6.1.5.

These features are combined linearly to predict a reward R as a weighted sum $R(s,\hat{\boldsymbol{w}})=\hat{\boldsymbol{w}}^\top \boldsymbol{f}(s)$ with weight vector $\hat{\boldsymbol{w}}$. For every pedestrian, a Value Iteration-like algorithm is executed on these reward maps to generate policy maps for a Markov Decision Process (MDP). This policy map encodes probabilities of transition between neighboring cells.

The outcome of the planning algorithm is governed by the weighting vector $\hat{\boldsymbol{w}}$. It is found through maximum entropy inverse reinforcement learning [183]. Given a set of observed trajectories $\mathcal{D}$, the statistics of features $\bar{\boldsymbol{f}}$ of all cells s visited by trajectories is estimated. Next, given an initial guess for $\hat{\boldsymbol{w}}$, the MDP is simulated for the combinations of start and destination for all trajectories in $\mathcal{D}$. From the resulting policies, again, statistics $\hat{\boldsymbol{f}}_{\hat{\boldsymbol{w}}}$ for the features of all visited cells are computed. Then, the parameters are updated to match observed statistics $\bar{\boldsymbol{f}}$ and those generated from simulations, $\hat{\boldsymbol{f}}_{\hat{\boldsymbol{w}}}$. The intuition behind this is to not learn from trajectories directly, but rather try to replicate location-dependent behavior patterns.

The use of semantic feature maps allows to incorporate arbitrary environment traits. In this work, a subset of the features of the original work is used, namely road, sidewalk, curb and obstacles. Since distances to features also play a major role, exponential distance maps were computed for different decay parameters. Exemplary feature maps are depicted in Fig. 3.2.

Since planning-based prediction requires some goal to plan for, distributions of goal locations are extracted from the training data. At inference time, planning is executed to all of these goals. As time proceeds, it is possible to recursively estimate the probability of every goal to be correct. This is done by comparing the observed actions to the simulated policies.

Intuitively, if the observed actions consecutively align with the simulations for a specific goal, this goal is more likely to be correct.

For evaluation, the weight vector $\hat{w}$ is first trained given start and goal of ground truth trajectories as stated in [86]. Evaluation is then run using inferred goal locations. The required distribution of possible goals is estimated using a static mixture of Gaussian Distributions (cf. Section 5.1.1) for which here, only the mixing coefficients are updated.

3.3.5. Comparison and Discussion

The four selected reference methods were optimized and evaluated on the same datasets. They are assessed with respect to three use cases, namely prediction of full trajectories in time and space, time-independent path forecasting, and inference of destinations. For detailed explanation of data and evaluation, see Section 6.

The same way the use cases vary, different properties of prediction are assessed with different metrics. The average exactness of an algorithm is reflected by the mean of predicted probabilities evaluated in ground truth positions, mPP. This, however, under-represents cases where prediction is completely off, i.e. the prediction's safety. The mean negative log probability, mNLP, pays more respect to erroneous results, i.e. where predicted probabilities approach zero, the NLP goes to infinity. Prediction in grid maps can also be interpreted as decision whether a cell is occupied or not. For such classification tasks, the area under the precision-recall curve (AuPR) is commonly used. The metrics and their selection are discussed in detail in Section 6.4.

To ensure comparability, the parameters of all experiments are set identical where applicable. The evaluated prediction time horizon for all experiments is four seconds in steps of 0.1s. The prediction is evaluated starting at all individual observations in a trajectory. The Kalman Filter and IMM incorporate all observations in the state. Thus, full trajectories up to the current time instance are fed to the filters. For the CNN, the tracking state is integrated in the DOGMa. Therefore, prediction can be run on individual grids. For Activity Forecasting, the observations impact the goal identification only. Experiments show that after 0.5s of observations, they may become non-informative if the observed behavior does not match any of the predicted trajectories. Consequently, only the last 0.5s of the recorded track are used for goal inference. Furthermore, to decouple evaluation of planning from goal prediction, both, the pure maximum entropy reinforcement learning with known goals as well as the full inference as presented in [86] are assessed.

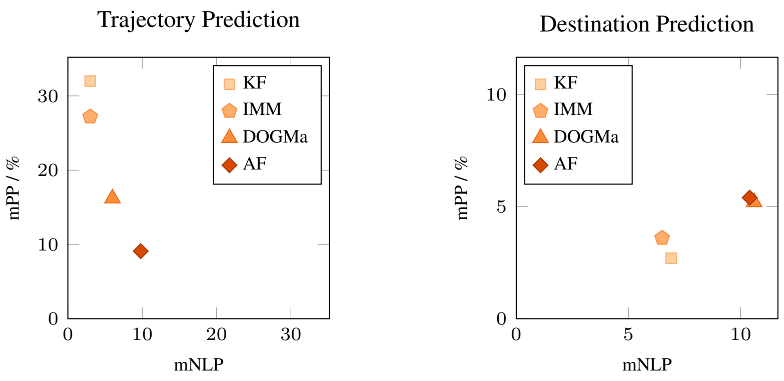
Method	Trajectory		Path	Destination	
	mPP/%	mNLP	AuPR	mPP/%	mNLP
Kalman Filter [19]	**32.0**	**3.0**	43.8	2.7	6.9
IMM [152]	27.2	**3.0**	**45.0**	3.6	**6.5**
DOGMa [69]	16.2	6.0	28.6	5.2	10.6
AF+known dest. [86]	16.7	14.8	27.2	-	-
AF [86]	9.1	9.8	12.0	**5.4**	10.4

Table 3.1.: Evaluation of reference algorithms for prediction of trajectory, path and destination. For mean predicted probabilities (mPP) and area under precision recall (AuPR), high values are better, for mean negative log probability (mNLP), low values are better.

The evaluation results of all methods are listed in Table 3.1 and also visualized in Fig. 3.3. They show that, on average, the Kalman Filter prediction performs impressively well even compared to more sophisticated solutions. It achieves best results in trajectory prediction, only closely followed by the interacting multiple model filter. The latter manages well to predict a path as it can represent categories of motion patterns in individual models and does not have to account for them in process uncertainty. Unfortunately, switching between these models introduces some latency that leads to degraded trajectory prediction.

For destination prediction, the IMM and Kalman Filter are good in *safety*, as reflected by mNLP. Since their uncertainty ellipses grows over time, the true destination will most likely be included in it. Sharper localization, however, is achieved by CNN and Activity Forecasting that identify motion patterns rather than mere physical feasibility.

Surprisingly, the planning-based solution does not live up to expectations in trajectory prediction. Even with destinations given, it does not perform better than other models. Adding the uncertainty of destinations improves the algorithm's prediction in terms of mNLP but, of course, deteriorates the sharpness of prediction. The reason for these shortcomings lie in both, the design of the algorithm and its input data. Firstly, maximum reinforcement learning is known to be overconfident in its predictions [75]. This explains poor values in mNLP. Furthermore, with overconfident trajectory predictions, the inference of goals fails easily, leading to poor mNLP in destination prediction.



(a) Trajectory prediction. (b) Destination prediction.

Figure 3.3.: Result maps of prediction tasks. The plots show the mean negative log probability (mNLP), representing safety, vs. the mean predicted probability (mPP) of the ground truth, i.e. exactness. The further to the top left, the better the prediction.

Last, regression of the reward as a linear feature combination per cell cannot deal with unobservable regions and context. It can, thus, only be applied in fully observed environments. This, however, is impossible to realize from a moving vehicle in traffic.

Several conclusions can be drawn from this experimental comparison. Most obviously, the physical model of the Kalman Filter delivers impressive performance, a fact that has also been recognized in literature [153]. However, for long-term prediction, its indefinitely growing uncertainty cannot compete with recognition of behavior patterns in terms of precise localization. While the planning-based algorithm promises the most natural explanation for motion, its current realization cannot fulfill this promise from on-board vehicle sensing. To overcome the drawbacks of the individual approaches, a solution is desired that can combine physical motion models with the explanatory capabilities of planning within the boundaries of statistical plausibility.

Chapter 4

Probabilistic Goal-Directed Prediction

As we have seen before, prediction of motion is always subject to uncertainties. These stem from two different sources: firstly, the underlying intentions that drive all motion cannot be known from the outside. Secondly, even if the intention was known, its actual execution can still vary and is therefore subject to process noise.

In the proposed prediction framework, respect should be paid to both of these sources of uncertainty. Both can be addressed jointly by probabilistic goal-directed prediction. In this, uncertainty about goals is used to reflect unknown intentions. Given these goals, probabilistic prediction towards them is used to incorporate the uncertainty about motion. In this chapter, we introduce the foundations for the model as well as two possible takes to goal-directed motion prediction, also published in [129, 130, 133].

4.1. Model Formulation

In general, a trajectory describes the evolution of an agent's state variables over time. In the case of object motion, it is customary that the state represents the agent's position with respect to some reference coordinate frame. Due to their continuous nature of both time and space, a real world trajectory is always defined in continuous domain.

Intelligent systems usually deal with time-discrete sampling, often due to fixed sensor measurement rates. Consequently, in this work, trajectories are represented as a sequence of discrete-time states

$$\zeta = \left(\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{N_\zeta}\right), \tag{4.1}$$

where the subscripts in the sequence of states denote the time instances. For the actual prediction task, only of trajectories of length $T{+}1$ are considered. Also, w.l.o.g., th start time of such a piece is referred to as $t{=}0$ and the end time as $t{=}T$.

For a predicted trajectory ζ, the states $\boldsymbol{s}_0$ and $\boldsymbol{s}_T$ are of special relevance: the one at time $t{=}0$ denotes the starting point of a trajectory. The other at time $t{=}T$ is the end point which may be considered as the short term *goal* of an agent's motion. This goal is substantial in trajectory planning. All actions in the time span $0{\leq}t{<}T$ will be executed with the purpose of reaching the destination $\boldsymbol{s}_T$.

In addition to the destination, the environment also shapes the layout of a trajectory. As introduced in Chapter 1.3, the surrounding of a moving agent greatly affects its motion. A comprehensible example might be an obstacle blocking the direct path towards the goal. Accordingly, a representation of the surrounding Θ is incorporated. It represents the spacial layout of different properties of the environment that might be relevant for decisions in motion planning.

In prediction, none of the above properties is known with absolute certainty. Instead, probabilities have to be incorporated into the sequence of states that form the trajectory. Thus, the prediction task can be understood as estimating the probability distribution of a trajectory ζ, given the observations $z_t, t{\leq}0$ up to the current time instance $t{=}0$

$$p(\zeta|z_0, z_{-1}, \ldots). \tag{4.2}$$

As a first assumption, the observations $z_0, z_{-1}, \ldots$ are split into those that stem from the environment and those that are specific for the agent in question. It is assumed that all previous observations of the environment can be incorporated into single map Θ_0, while those of the agent itself are condensed into x_0, $p(\zeta|\Theta_0, x_0)$.

As mentioned above, the state at the end time T, $\boldsymbol{s}_T$, plays a major role as this specific state represents the goal of the current trajectory. All intermediate states within the time horizon $0<t<T$ are traversed in order to reach this goal state. This means that all transition states can be expressed as dependent on environment, agent-specific observations and goal state

$$p(\zeta|\Theta_0, x_0, \boldsymbol{s}_T). \tag{4.3}$$

Since an agent's current goal cannot be known from the outside, this again is subject to uncertainty. Thus, the goal state may be treated as a latent variable. If a distribution of the goal states can be estimated from observations, $p(\boldsymbol{s}_T|\Theta_0, x_0)$, we may marginalize over all possible values to obtain the distribution of all transition states given only the observations

$$p(\zeta|\Theta_0, x_0) = \int p(\zeta|\Theta_0, x_0, \boldsymbol{s}'_T)p(\boldsymbol{s}'_T|\Theta_0, x_0)d\boldsymbol{s}'_T. \tag{4.4}$$

Equation (4.4) describes a generic model for probabilistic goal-directed prediction. In practice, however, the probability distribution of a specific trajectory $p(\zeta|\Theta_0, x_0)$ are of little interest. Instead, an autonomous systems needs to assess its risk of colliding with another agent at a certain position and time. This information is embedded in the marginal distributions of $p(\zeta|\Theta_0, x_0)$ for the individual time steps, $p(\boldsymbol{s}_t|\Theta_0, x_0), \forall t \in [0, T]$. Thus, instead of estimating Eq. (4.4), we limit ourself to the marginal distributions

$$p(\boldsymbol{s}_t|\Theta_0, x_0){=}\int p(\boldsymbol{s}_t|\Theta_0, x_0, \boldsymbol{s}'_T)p(\boldsymbol{s}'_T|\Theta_0, x_0)d\boldsymbol{s}'_T,\ \forall t \in [0, T]. \tag{4.5}$$

This formulation now allows for a time step-wise estimate of the distributions. Accordingly, in order to solve this problem, a parameterization of the distributions has to be found and the parameters estimated.

The prediction of uncertainties from the time instance t to another time $t{+}1$ requires a model to transform the distribution $p(\boldsymbol{s}_t)$ into $p(\boldsymbol{s}_{t+1})$. Independent of the specific motion model of an agent, we may consider it as a control input at time t, $\boldsymbol{u}_t$, that is added to the current state vector to generate the next state

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t + \boldsymbol{u}_t. \tag{4.6}$$

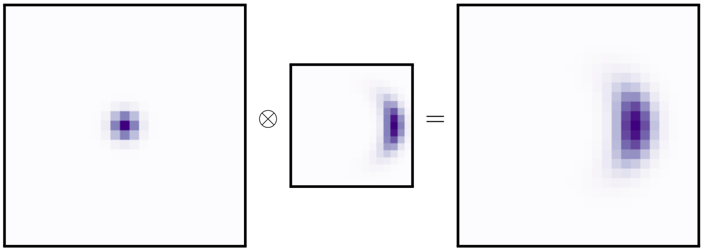


Figure 4.1.: Convolution of an initial distribution with a distribution of motion together and resulting distribution.

Both the state $\boldsymbol{s}_t$ and the motion input $\boldsymbol{u}_t$ in Eq. (4.6) are subject to uncertainty, i.e.

$$\begin{aligned} \boldsymbol{s}_t &\sim p_{\boldsymbol{s}_t}, &\quad (4.7)\\ \boldsymbol{u}_t &\sim p_{\boldsymbol{u}_t}. &\quad (4.8)\end{aligned}$$

If we wish to construct the distribution of $\boldsymbol{s}_{t+1}$, we can use the fact that we can solve Eq. (4.6) for $\boldsymbol{u}_t$ and thus set

$$p_{\boldsymbol{s}_{t+1}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t) = p_{\boldsymbol{u}_t}(\boldsymbol{s}_{t+1} - \boldsymbol{s}_t). \qquad (4.9)$$

Marginalization over all possible preceding states s_t yields

$$p_{\boldsymbol{s}_{t+1}}(\boldsymbol{s}_{t+1}) = \int p_{\boldsymbol{u}_t}(\boldsymbol{s}_{t+1} - \boldsymbol{s}_t) p_{\boldsymbol{s}_t}(\boldsymbol{s}_t) d\boldsymbol{s}_t, \qquad (4.10)$$

which is nothing else but the convolution of $p_{\boldsymbol{u}_t}$ with $p_{\boldsymbol{s}_t}$ [68]

$$p_{\boldsymbol{s}_{t+1}} = p_{\boldsymbol{u}_t} \otimes p_{\boldsymbol{s}_t}. \qquad (4.11)$$

The same also holds for the inverse problem, i.e. finding the distribution of a preceding state given that of the current state. This will be necessary to step back in time from an estimate of a goal state $\boldsymbol{s}_T$. For this, Eq. (4.6) simply is solved for the preceding state $\boldsymbol{s}_t$,

$$\boldsymbol{s}_t = \boldsymbol{s}_{t+1} - \boldsymbol{u}_t. \qquad (4.12)$$

For the stepping back, Eq. (4.11) still applies, with only a modification such that $p_{\boldsymbol{u}_t}$ is flipped in the convolution, thus resulting in a correlation with the original distribution,

$$p_{\boldsymbol{s}_t} = p_{\text{-}\boldsymbol{u}_t} \otimes p_{\boldsymbol{s}_{t+1}}. \tag{4.13}$$

Eq. (4.13) represents a backward-directed prediction.

Depending on the individual distributions, one may find more or less compact solutions to Eq. (4.11), e.g. the prediction step in the Kalman Filter [22]. In this work, however, the aim is to model arbitrary distributions with a great variety of shapes and modalities. Due to these requirements, all distributions are approximated by a grid discretization if not stated otherwise. Accordingly, all state vectors $\boldsymbol{s}$ turn into discrete states s, all integrals get replaced by sums and convolutions are implemented as discrete convolutions. Fig. 4.1 visualizes the computation of Eq. (4.11) as a discrete convolution. With the use of discretized distributions and thus computations executed in grids, all available optimizations designed for image processing and convolutional neural networks can be utilized, e.g. [1, 77].

4.2. Forward-Backward Prediction

The goal-directed prediction problem can be solved by evaluating the probability of taking every possible path from start to destination. This can be reformulated as the problem of computing the probability to traverse from start to destination via any given intermediate state. In Markov Chains, the Forward-Backward-Algorithm is used to compute these intermediate probabilities [21, 39].

In a first step, the goal-directed prediction is modeled given a known start and goal state. For the sake of legibility, the influence of the environment and the agent-specific observations are neglected . For now, let us examine only the term

$$p(s_t|s_0, s_T). \tag{4.14}$$

In the following, the conditional independence of a previous state $s_{\text{-}}$ and a future state s_+ given an intermediate state s_0 is used,

$$p(s_{\text{-}}, s_+|s_0) = p(s_{\text{-}}|s_0)p(s_+|s_0). \tag{4.15}$$

With this, it can be derived that

$$\begin{aligned} p(s_t|s_0, s_T) &= \frac{p(s_0, s_T|s_t)p(s_t)}{p(s_0, s_T)} \\ &= \frac{p(s_0|s_t)p(s_t)p(s_T|s_t)}{p(s_0, s_T)}. \end{aligned} \tag{4.16}$$

Equation (4.16) shows that the prediction can be separated into parts that reflect the impact of the start and of the destination, respectively. In the following, let us analyze them individually. For this, they are separated according to

$$p(s_t|s_0, s_T) = \frac{\overbrace{p(s_0|s_t)p(s_t)}^{\alpha(s_t)}\overbrace{p(s_T|s_t)}^{\beta(s_t)}}{\underbrace{p(s_0, s_T)}_{Z}}, \tag{4.17}$$

where the denominator Z can be understood as a normalization.

The first term under consideration is $\alpha(s_t)$, the one that incorporates the starting state s_0. Making use of the definition of the conditional probability and Eq. (4.15) yields

$$\begin{aligned} \alpha(s_t) &= p(s_0|s_t)p(s_t) = p(s_0, s_t) \\ &= \sum_{s_{t-1}} p(s_0, s_t|s_{t-1})p(s_{t-1}) \\ &= \sum_{s_{t-1}} p(s_t|s_{t-1})p(s_0|s_{t-1})p(s_{t-1}) \\ &= \sum_{s_{t-1}} p(s_t|s_{t-1})\alpha(s_{t-1}). \end{aligned} \tag{4.18}$$

By introduction of the previous state s_{t-1} as a latent variable, it can be seen that $\alpha(s_t)$ can actually be computed recursively from the previous result $\alpha(s_{t-1})$ and the state transition model $p(s_t|s_{t-1})$.

As we have seen in Eq. (4.11), the state transition can be computed as a convolution of an input distribution $\alpha(s_t)$ with a motion model p_{u_t}

$$\alpha(s_t) = \alpha(s_{t-1}) \otimes p_{u_{t-1}}. \tag{4.19}$$

The same procedure can be applied to the term depending on the goal state, $\beta(s_t)$, to obtain

$$\begin{aligned}\beta(s_t) &= p(s_T|s_t)\\ &= \sum_{s_{t+1}} p(s_T|s_{t+1}, s_t)p(s_{t+1}|s_t)\\ &= \sum_{s_{t+1}} p(s_T|s_{t+1})p(s_{t+1}|s_t)\\ &= \sum_{s_{t+1}} \beta(s_{t+1})p(s_{t+1}|s_t). \end{aligned} \tag{4.20}$$

Again, we see that $\beta(s_t)$ follows recursion, but this time, it is computed from the subsequent state, $\beta(s_{t+1})$.

In analogy to Eq. (4.13), the backward-directed prediction is

$$\beta(s_t) = \beta(s_{t+1}) \otimes p_{\text{-}u_t}. \tag{4.21}$$

Finally, the normalization term Z is computed from $p(s_0, s_T)$. As $p(s_t|s_0, s_T)$ has to sum to one, we can see that

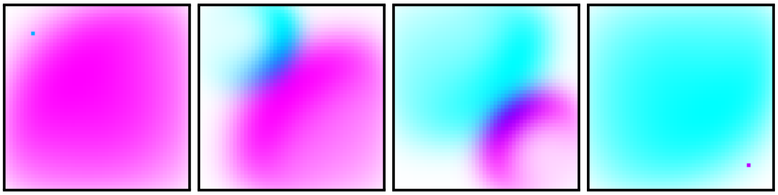
$$p(s_0, s_T) = Z = \sum_{s_t} \alpha(s_t)\beta(s_t). \tag{4.22}$$

The results from Eq. (4.19) to Eq. (4.22) are substituted into Eq. (4.17). The goal-directed prediction problem is therefore solved from two sets of recursive convolutions: the first starting at p_{s_0} up to p_{s_t} and the second starting at p_{s_T} and going backward to p_{s_t},

$$\begin{aligned} p(s_t) = &\frac{1}{Z}\left(p_{s_0} \otimes p_{u_0} \otimes \cdots \otimes p_{u_{t\text{-}1}}\right)\\ &\cdot \left(p_{s_T} \otimes p_{\text{-}u_{T\text{-}1}} \otimes \cdots \otimes p_{\text{-}u_t}\right). \end{aligned} \tag{4.23}$$

A toy example of this forward-backward prediction scheme is displayed in Fig. 4.2. In the top row, Fig. 4.2a, the forward and backward expansion of the probability distributions are shown in cyan and magenta, respectively. As time passes, the forward prediction expands away from the start in the top left while the backward prediction contracts towards the goal in the bottom right. Their joint distribution in Fig. 4.2b forms blob-shaped distributions for the transition steps.

A moving agent will select its dynamic behavior also based on the surrounding. A pedestrian, for example, will behave differently when crossing


(a) Forward (cyan) and backward (magenta) expansion of probability distributions over time in recursive prediction.

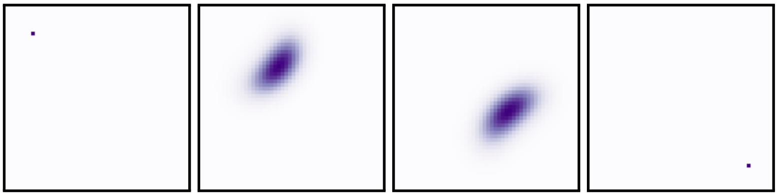

(b) Joint distribution that represents the state transition estimate over time as multiplication of fwd- and bwd-pass from above.

Figure 4.2.: Prediction as a multiplication of one forward and one backward oriented recursive convolution. The prediction starts at the top left corner and ends at the bottom right.

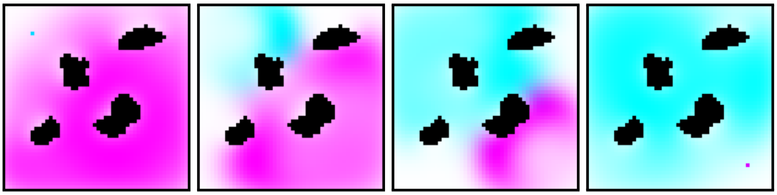
a road than when walking on the sidewalk. This impact of the environment on an agent's motion has to be accounted for in prediction. So far, our prediction system incorporates only dynamics and goal states.

To incorporate selective behavior, it is assumed that a set of different actions $\mathcal{A}$ exist. Every action $a \in \mathcal{A}$ defines a state transition probability

$$p(s_{t+1} | s_t, a). \tag{4.24}$$

The introduction of actions allows modeling location-specific behavior. For that, actions are selected dependent on the static layout of the environment, i.e. the currently observed map Θ, but not time t,

$$p(a | s_t, \Theta). \tag{4.25}$$


(a) Forward (cyan) and backward (magenta) expansion of probability distributions influenced by obstacles (black).

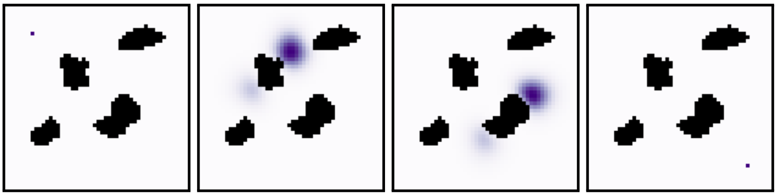

(b) Joint distribution results in a multimodal trajectory prediction that avoids the obstacles.

Figure 4.3.: Prediction with environment influence. As in Fig. 4.2, the trajectory starts at the top left and ends at the bottom right.

Under these assumptions, the environment-aware prediction is obtained by marginalization over all actions

$$p(s_{t+1}|s_t,\Theta) = \sum_{a\in\mathcal{A}} p(s_{t+1}|s_t,a)p(a|s_t,\Theta). \tag{4.26}$$

Note that this can be implemented as a convolution similar to Eq. (4.11). However, now, the input features one additional dimension, namely that for the selected action,

$$p(s_{t+1}|\Theta) = p(a|s_t,\Theta)p(s_t|\Theta) \otimes p_{a,u_t}. \tag{4.27}$$

Substituting this prediction scheme into Eq. (4.19) and Eq. (4.21) yields

$$\begin{aligned} p(s_t|\Theta) =& \frac{1}{Z}\left(p_{s_0}p(a|\Theta) \otimes p_{a,u_0}p(a|\Theta) \otimes \cdots \otimes p_{a,u_{t\text{-}1}})\right) \\ &\cdot \left(p_{s_T}p(a|\Theta) \otimes p_{a,\text{-}u_{T\text{-}1}}p(a|\Theta) \otimes \cdots \otimes p_{a,\text{-}u_t}\right). \end{aligned} \tag{4.28}$$

Fig. 4.3 shows the same toy example as above. But now, obstacles have been introduced to the prediction grid (shown in black). These are modeled as a binary distribution $p(a|s_t, \Theta)$ that is zero for actions that transition into an obstacle and uniform everywhere else. Accordingly, the forward and backward predictions will be blocked by the obstacles as shown in Fig. 4.3a. The joint distribution results in a bi-modal trajectory prediction, passing either above or below the obstructions (Fig. 4.3b).

4.3. Markov Decision Processes for Prediction

The concept of moving through a state space can also be considered execution of a chain of decision. Since this is subject to uncertainty, they can be modeled as a Markov Decision Process [86, 182].

Let us reconsider Eq. (2.2) for Value Iteration

$$V_{k+1}(s) \ := \ \max_a \sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V_k(s')).$$

This equation can be expanded into two parts

$$V_{k+1}(s) := \max_a \left[\underbrace{\sum_{s'} P_a(s,s')R_a(s,s')}_{\text{Expected reward of current action}} + \gamma \underbrace{\sum_{s'} P_a(s,s')V_k(s')}_{\text{Expected reward of later actions}} \right]. \tag{4.29}$$

The first part, $\sum_{s'} P_a(s,s')R_a(s,s')$, models the reward that an agent will gain from taking an action starting in s and ending up in s'. This reward is weighted with the probability of this event to happen.

The second part, $\sum_{s'} P_a(s,s')V_k(s')$, propagates expected rewards of later actions onto the current state s. It is weighted by a discount factor γ to favor early rewards.

The distribution $P_a(s,s')$ describes the probability of ending up in state s' when taking an action a in state s. In this context, different actions a may be considered as control inputs just as u in Eq. (4.6).

Consequently, the probability of traversing from s to s' is distributed according to $p_u(s'{-}s)$. Then, the transition probability $P_a(s,s')$ becomes

$$P_a(s,s') = p_u(s'-s) = p_{\text{-}u}(s-s'), \tag{4.30}$$

where the last equality uses the fact that $s{-}s'{=}\text{-}u$.

Substituting Eq. (4.30) into Eq. (4.29) yields

$$V_{k+1}(s) := \max_a \left[\sum_{s'} p_{\text{-}u}(s-s') R_a(s,s') + \gamma \sum_{s'} p_{\text{-}u}(s-s') V_k(s') \right]. \tag{4.31}$$

The second term in Eq. (4.31) denotes a discrete convolution

$$\sum_{s'} p_{\text{-}u}(s-s') V_k(s') = p_{\text{-}u}(s) \otimes V_k(s) \tag{4.32}$$

of the value function $V_k(s)$ with the flipped distribution of the motion vector, $p_{\text{-}u}$, so that in the following $p_{\text{-}u}(s){=}p_a(s)$. For the first part, we can see that there is a unique reward per possible action a that is a function of both start and end states, s and s'. For further simplification, rewards are demanded to only depend on their target state,

$$R_a(s,s') := R_a(s), \tag{4.33}$$

similar to the assumption made in Q-learning [173]. Substituting Eq. (4.32) and Eq. (4.33) into Eq. (4.31) yields

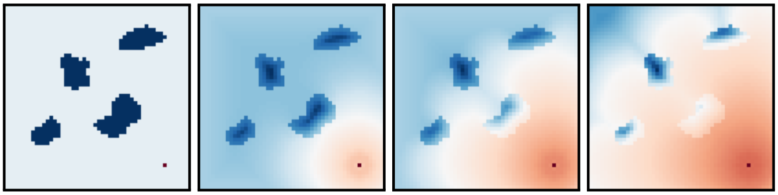
$$V_{k+1}(s) \;:=\; \max_a \left[\gamma(p_a(s) \otimes V_k(s)) + R_a(s) \right]. \tag{4.34}$$

This finding is fundamental as it allows to formulate the Value Iteration Algorithm as recursive convolutions of the value function $V_k(s)$ with transition PDFs $p_s(s)$ for all possible motion patterns [155, 163].

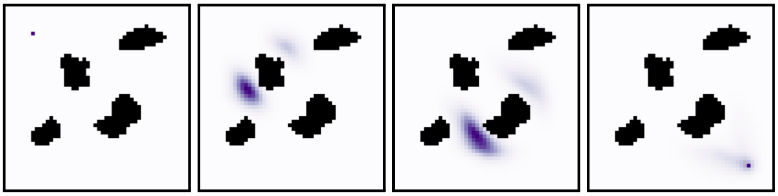
The Value Iteration Algorithm from Eq. (4.34) can be parameterized by two design choices, namely the transition PDFs $p_s(s,s')$ and the reward function $R_a(s)$. The transition probability distributions represent different motion patterns that can be selected by an agent on its way to the goal, including their uncertainty.

The reward function on the other hand determines the location dependent planning characteristics within the MDP. Thus, it has to incorporate the goal to be planned for as well as the influence of the environment on the motion behavior.

In that sense, while the transition function only depends on the agent under examination, the reward function $R_a(s)$ also depends on the goal state s_T and the environment Θ. Following the insights from Section 4.2, the reward function is modeled as $R_a(s, s_T, \Theta_0)$. The Value Iteration Algorithm then computes the maximum reward to be collected in each state.



(a) Expansion of the value function in Value Iteration. High rewards (red) expand away from the destination while obstacels introduce high penalties (blue) .



(b) Simulation of the policy derived from value function. High penalties for obstacle collision lead to multimodal trajectory prediction.

Figure 4.4.: Position prediction with Value Iteration. Just as in Fig. 4.3, the trajectory starts at the top left and ends at the bottom right.

Running the argmax instead of the maximum at convergence of $V(s)$ will output the optimal policy

$$\pi_a(s) \;:=\; \underset{a}{\operatorname{argmax}} \left[\gamma(p_a(s) \otimes V(s)) + R_a(s)\right]. \qquad (4.35)$$

For prediction, this policy then has to be tracked from the starting state until the goal state is reached.

Figure 4.4 depicts the trajectory prediction using an MDP. The same setup that was used to generate Fig. 4.3, is now used to simulate the MDP. For this, the reward function Eq. (4.33) is defined so that the goal state is rewarded while hitting obstacles is penalized. Then, Eq. (4.34) is run to generate Fig. 4.4a. Once Eq. (4.34) has converged, Eq. (4.35) is applied to generate the optimal policy that is then simulated for Fig. 4.4b. Again, the result is a bi-modal distribution that successfully avoids the obstacles.

Chapter 5

Pedestrian Position Prediction

While prediction of other moving agents in general is a crucial task in autonomous navigation, this becomes even more important if these agents are humans. Due to the risk of injury, an autonomous agent should avoid collision with a human in any case [9]. This is especially true for automated vehicles in real world traffic since collisions often have fatal consequences [117]. Thus, accurate prediction is a prerequisite for autonomous driving. Chapter 4 introduced a generic scheme for goal-directed prediction in arbitrary scenarios. This chapter looks at this from the perspective of automated driving where the moving agent in question is a pedestrian walking in the streets.

Pedestrian navigation in traffic can be considered as that of an intelligent agent planning a trajectory towards a predefined goal. With this assumption, all findings from Chapter 4 can be used for pedestrian prediction. Specifically, goal-directed prediction can be employed under the assumption of a latent destination as presented in Chapter 4.1. The two major challenges in this context are the estimation of the destination as well as the (possibly position-dependent) motion model. Since the system should be applied for automated driving, these need to be inferred from observations from a vehicle in traffic.

In order to apply the proposed goal-directed prediction Eq. (4.5), we need to find a model to infer the relevant parameters from observations. As a prerequisite, this begins with the estimation of possible destinations.

Once these are available, prediction can be run using a suitable parameterization of the Forward-Backward Algorithm Eq. (4.28) or Value Iteration Eq. (4.34). With the sheer quantity of traffic scenarios, however, it is infeasible to manually tune the parameters of the algorithms. Instead, the optimal parameterization needs to be *learned* from previously observed pedestrian behavior.

Of all the machine learning techniques, Convolutional Neural Networks (CNNs) and Deep Learning have proven the most versatile [56]. With its recent development, a backbone of toolboxes, methods and network architectures exists to facilitate model parameterization [1, 15, 36, 77]. Through the formulation of prediction as a recursive convolution in either Eq. (4.28) or Eq. (4.34) and appropriate constraints, the prediction problem can be interpreted as a recurrent CNN. Furthermore, all data-driven probability distributions can be inferred from sensor readings by artificial neural networks (ANNs), e.g. for destinations or transition probabilities.

In this chapter, the goal-directed prediction of pedestrians in traffic is implemented by means of ANNs, also partly published in [133]. For this, the estimation of possible goals is the prerequisite. Consequently, first, a network architecture is introduced to infer probability density functions of possible destinations from observations. Secondly, two different recurrent CNNs are proposed, namely one to solve the forward-backward prediction presented in Chapter 4.2 and one for the MDP from Chapter 4.3. Finally, goal and trajectory prediction are combined together in one monolithic network that is fully differentiable. Accordingly, it is shown how network can consistently be trained end to end.

5.1. Destination Prediction

Goal-directed planning for prediction can only be applied if knowledge of a pedestrian's goal is available. In this work, the goal of planning is for a pedestrian to reach a certain destination at a specific time. However, since one cannot know a pedestrian's destination from the outside, it has to be inferred from observations. Due to the probabilistic nature of this task, the aim is to estimate a PDF $p(s_T|x_0)$ of the pedestrian's current destination s_T at a future time T, given past observations x_0.

Human motion is usually driven by some intention, i.e. reaching some destination. Observed from the outside, a pedestrian's intention will not be obvious. Instead, a set of possible intentions has to be considered of which only one will turn out to be correct. For the example in Fig. 5.1, one may have to consider the options of the pedestrians *crossing the road* versus *stopping at the curb*.

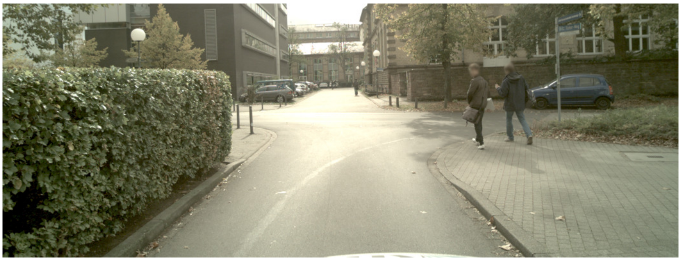


Figure 5.1.: Even from a single image, the pedestrians' intention to cross the road is still obvious to the human driver.

Now, one possible destination can be associated with both of these options, namely the sidewalk on the other side of the road or the curb on this side. However, even if these two destinations can be identified, they still are subject to uncertainty since their exact location cannot be predicted. From this, we can see that prediction of destinations consists of two distinct sources of uncertainty: first, the intention is obscure to us. It can be considered as a categorical distribution of different possible intentions, where a probability can be associated with any of them. Second, the outcome may vary when an intention is carried out.

From the perspective of autonomous driving, we are not necessarily interested in the semantic nature of an intention. It is not relevant to know whether a destination represents *crossing the road* or *stopping at the curb* as long as this enables correct prediction and, consequently, collision risk assessment. Thus, the destination is modeled as a mixture density function. Every mixture component can be interpreted as a categorical intention together with its intrinsic uncertainty. The mixture coefficients lastly estimate the probability of each intention to be true.

5.1.1. Goal Distribution Parameterization

As we have seen, the multitude of possible intentions of a person can be expressed as a mixture density function. The individual components of this function are designed to represent one possible destination each. Naturally, these components should express a probability in terms of the position of the goal (x_T, y_T). Also, for this use case, the pedestrian's orientation in the goal state, ψ_T, provides valuable information. This orientation helps to identify the direction of motion, even for the goal state. It may, for example, tell apart pedestrians *facing* the road or walking *along* the sidewalk. Consequently, the goal state is defined as $\boldsymbol{s}_T{=}(x_T, y_T, \psi_T)^\top$.

To incorporate both, position and orientation, into the destination PDF, it is factorized into two parts. One part is dependent solely on the position and the other on the orientation of the goal state,

$$p_i(\boldsymbol{s}_T) = p_i(x_T, y_T, \psi_T) = q_i(x_T, y_T) v_i(\psi_T), \tag{5.1}$$

where the subscript i denotes the ith mixture component.

The PDF $q_i(x_T, y_T)$ represents the uncertainty of the goal's position. It is parameterized as a general bivariate Normal Distribution with mean $(\bar{x}_i, \bar{y}_i)$ and covariance matrix $\boldsymbol{\Sigma}_i$ [68],

$$\begin{aligned} q_i(x_T, y_T) &= \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_i)}} \exp\left(\text{-}\frac{1}{2} \begin{pmatrix} x_T\text{-}\bar{x}_i \\ y_T\text{-}\bar{y}_i \end{pmatrix}^{\top} \boldsymbol{\Sigma}_i^{-1} \begin{pmatrix} x_T\text{-}\bar{x}_i \\ y_T\text{-}\bar{y}_i \end{pmatrix} \right) \\ &= \mathcal{N}(x_T, y_T | \bar{x}_i, \bar{y}_i, \boldsymbol{\Sigma}_i) \end{aligned} \tag{5.2}$$

The uncertainty of orientation is modeled by a von-Mises Distribution with mean angle $\bar{\psi}_i$ and concentration parameter κ_i [105],

$$\begin{aligned} v_i(\psi_T) &= \frac{1}{2\pi I_0(\kappa_i)} \exp(\kappa_i \cos(\psi - \bar{\psi}_i)), \\ &= \mathcal{M}(\psi_T | \bar{\psi}_i, \kappa_i), \end{aligned} \tag{5.3}$$

where $I_0(\kappa_i)$ is the modified Bessel Function of the first kind which is required as normalization of the PDF.

Every component according to Eq. (5.1) represents a single destination. Since multiple independent goals should be modeled, different goals are mixed together in one distribution. The final estimate of the multi-modal PDF for destinations is a weighted sum of N individual possible destinations, each of them weighted with a mixing coefficient π_i as

$$p(\boldsymbol{s}_T) = \sum_{i=1}^{N} \pi_i p_i(x_T, y_T, \psi_T | \bar{x}_i, \bar{y}_i, \bar{\psi}_i, \boldsymbol{\Sigma}_i, \kappa_i), \tag{5.4}$$

$$\text{where} \quad 0 \le \pi_i, \ \forall \pi_i \quad \text{and} \quad \sum_{i=1}^{N} \pi_i = 1.$$

The two factors Eq. (5.2) and Eq. (5.3), are conditioned on a set of parameters, namely the shape parameters of the individual distributions. Also, each mixing coefficient also introduces a parameter π_i.

For the sake of legibility, the set of parameters per mixture component is abbreviated as $\boldsymbol{v}_i = (\bar{x}_i, \bar{y}_i, \bar{\psi}_i, \Sigma_i, \kappa_i, \pi_i)$, where $i = 1, \ldots, N$ for N mixture components. With $\boldsymbol{v}^N = \boldsymbol{v}_1, \ldots, \boldsymbol{v}_N$, the final conditional is

$$p(x_T, y_T, \psi_T | \boldsymbol{v}^N). \tag{5.5}$$

The parameter set $\boldsymbol{v}^N$ fully describes the distribution of possible destinations. Thus, to predict destinations in form of such a distribution, said parameter set have to be estimated from observations $\boldsymbol{x}$. In order to do so, a function

$$\boldsymbol{v}^N = f(\boldsymbol{x}, \hat{\boldsymbol{w}}) \tag{5.6}$$

is employed where $\hat{\boldsymbol{w}}$ is the parameter set governing the function f.

The function f with given observations $\boldsymbol{x}$ and function parameters $\hat{\boldsymbol{w}}$ can readily be substituted into Eq. (5.5). As a consequence, the distribution is no longer conditioned on its parameters but instead, on the parameters $\hat{\boldsymbol{w}}$ and the observations $\boldsymbol{x}$. Generally, the parameters and observations are design choices of the system where prediction is to be applied. In practice, however, the realizations of observations are restricted by the sensor setup in use. In system design, a setup should be selected that optimally captures the information relevant for prediction. The choice of the function f should then be able to make the most use of this sensor data.

5.1.2. Parameter Estimation

Previous works have shown that pedestrians' intentions can be inferred from a multitude of features [23, 83]. Studies with human participants, however, showed that motion and visual information pose the strongest cues for prediction [150, 151]. A sensor that can inherently capture these is a video camera. While still images already contain valuable information such as posture and scene layout, subsequent frames also contain their evolution over time.

In video processing, Deep Neural Networks have, as already mentioned, achieved state of the art performance, even for time sequence understanding [43, 171, 180]. The prediction model introduced in Section 5.1.1 proposed to model the goal of a pedestrian as a mixture density function. As defined in Eq. (5.4), a function $f(\boldsymbol{x}, \hat{\boldsymbol{w}})$ is required to estimate the mixture parameters from observations. Exactly this property is reflected in Mixture Density Networks (MDN) [20, 21]. In these, outputs of a neural network are adjusted with suitable activation functions to satisfy the constraints of a given probability distribution.

For application, the parameters of Eq. (5.4) are estimated using an MDN. The mixture itself is parameterized by eight parameters per component. These originate from the two factors in Eq. (5.1) as well as one mixing coefficient per component. Consequently, a neural network is designed with a final output of the type $\{m_x, m_y, d_x, d_y, r, p, k, g\}_i$, where $i \in 1, \ldots, N$ for N mixture components.

The Gaussian part of the factorization, Eq. (5.2), can be reflected by the two-dimensional mean vector $(\bar{x}_i, \bar{y}_i)^\top$ as well as the covariance matrix

$$\mathbf{\Sigma}_i = \begin{bmatrix} \sigma_{x,i}^2 & \rho_i \sigma_{x,i} \sigma_{y,i} \\ \rho_i \sigma_{x,i} \sigma_{y,i} & \sigma_{y,i}^2 \end{bmatrix}, \tag{5.7}$$

with variances $\sigma_{x,i}^2, \sigma_{y,i}^2$ and Pearson Coefficient ρ_i.

For the mean vector, linear activation is applied to the neuron outputs,

$$\bar{x}_i = m_{x,i}, \tag{5.8}$$

$$\bar{y}_i = m_{y,i}, \tag{5.9}$$

since the position of the goal can take arbitrary real values in the world.

The variances, on the other hand, have to be strictly positive. This can be achieved with exponential activations

$$\sigma_{x,i} = \exp(d_{x,i}), \tag{5.10}$$

$$\sigma_{y,i} = \exp(d_{y,i}). \tag{5.11}$$

Since the Pearson Coefficient must satisfy $-1 \leq \rho_i \leq 1$, a hyperbolic tangent is used as activation,

$$\rho_i = \tanh(r_i). \tag{5.12}$$

With this, the general covariance matrix Eq. (5.7) can be constructed from the network outputs.

The second factor of Eq. (5.1), the von-Mises Distribution in Eq. (5.3) is parameterized by two parameters. These can also be estimated with fitting activations. The parameters, mean and concentration, of the distribution are computed as

$$\bar{\psi}_i = g_i, \quad \text{and} \tag{5.13}$$

$$\kappa_i = \exp(k_i). \tag{5.14}$$

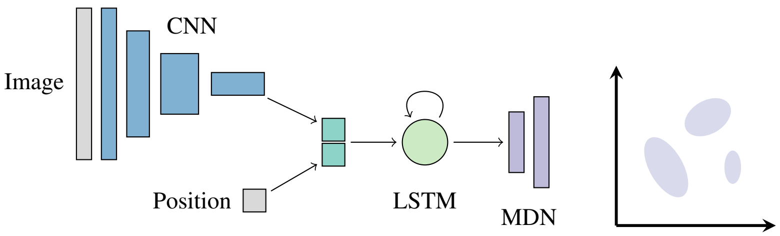


Figure 5.2.: Recurrent Mixture Density Network for destination prediction: a CNN embeds features of the image that are concatenated with the pedestrians position. This is fed to an LSTM cell to process sequential data. The LSTM's output serves as an input to an MDN that predicts the final location distribution.

Finally, to generate a valid mixture, the mixing coefficients predicted by the network need to satisfy

$$\pi_i \geq 0 \quad \forall i \in 1, \dots, N, \tag{5.15}$$

$$\sum_{i=1}^{N} \pi_i = 1. \tag{5.16}$$

A function that fulfills these requirements is the softmax over all corresponding network outputs

$$\pi_i = \frac{\exp(p_i)}{\sum_{j=1}^{N} \exp(p_j)}. \tag{5.17}$$

These activation functions enforce that at any time, the output of the neural network will satisfy the constraints imposed by the mixture's probability density function.

The MDN as defined by Eq. (5.4) can be built upon arbitrary underlying networks. The only prerequisite is the dimensionality of the output as eight parameters per mixture. However, with the informative features from video sequences, there is demand for a network that can make use of image features as well as time sequence information. To satisfy these requirements a network is constructed from a CNN part for image processing and a recurrent part for time series understanding. The full network is depicted in Fig. 5.2. In the first part, image crops of the pedestrian of interest are passed through a standard CNN architecture. The resulting feature embedding is then concatenated with the pedestrian's current position for joint processing in the subsequent network.

The concatenated feature vector is fed to a Long Short-Term Memory (LSTM) recurrent network cell [67, 149]. Finally, the LSTM's output is processed by a fully connected (FC) network to generate the output according to Eq. (5.4).

Once trained, the final network can perform mixture density destination prediction from time series of image and trajectory data. With this as an input, all following planning techniques can rely on destination predictions. Moreover, even multiple possible destinations can be tracked, including their uncertainty.

5.2. Trajectory Prediction

Chapter 4 argued that planning algorithms can be used as a predictor when knowledge about a destination is available. The previous section introduced a learning architecture to predict such destinations. With this, now, planning algorithms can be applied for probabilistic prediction.

Most planning algorithms are parameterized by two design elements, namely the motion model and a cost functional to be optimized for. However, modeling human planning behavior proves difficult: humans are usually unaware of their actual dynamical model and rarely consciously optimize a cost functional to plan their own motion. To navigate through the world, they make decisions based on *experience*. Consequently, it is plain infeasible to model the human planning explicitly. Instead, just like a human, in this work, a planning algorithm is built from experience. For this, the planning is reformulated as a machine learning task. From previously observed pedestrians, the model *learns* a planning algorithm to imitate human behavior. With the obvious benefits of deep learning in mind, this section explains how the planning schemes from Chapter 4 can be formulated by means of Artificial Neural Networks and how to train them.

5.2.1. Forward Backward Network

The first trajectory prediction under consideration is the Forward Backward Algorithm as introduced in Chapter 4.2. As we have already seen in Eq. (4.28), prediction can be modeled as a product of recursive convolutions. This formulation enables the use of CNN architectures for prediction learning.

As a first prerequisite, all distributions are represented as discrete state grids. For this, the state space is discretized into equidistant cells of a fixed size. Also, the prediction time is evaluated in steps of a fixed interval. For every time step up to the prediction horizon, one state grid is created.

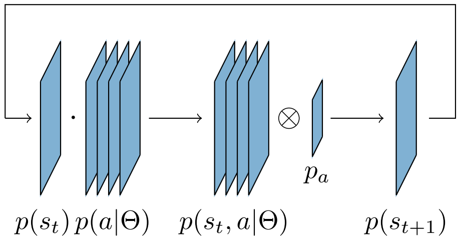


Figure 5.3.: Recurrent stage for forward prediction with map-dependent probability of actions.

The cells of these grids will later reflect the probability of finding a pedestrian in a certain *area* at a certain time. Similarly, the distributions of state transitions are discretized into grids of state differences. The pair of these two grids can model Eq. (4.11) as a convolutional layer in a CNN. The input of the layer is the distribution of possible positions of a pedestrian, the filter masks represent the state transition distributions. For this, they are required to be valid probability distributions. To achieve this, a suitable activation is applied to the filter weights according to

$$p(x, y, a) = \frac{\exp(\hat{w}(x, y, a))}{\sum_{\tilde{x}=1}^{W} \sum_{\tilde{y}=1}^{H} \exp(\hat{w}(\tilde{x}, \tilde{y}, a))}, \tag{5.18}$$

where W and H are the spatial extent of the filter and, again, a the action. Note that Eq. (5.18) is closely related to the softmax function that is used to generate categorical distributions. But instead, here, normalization is carried out over individual filters to create valid transition distributions.

As shown in Eq. (4.19), repeated convolutions can be used to propagate probability distributions into the future. The prediction for two time steps ahead can be obtained from the prediction for one time step ahead. Hence, Eq. (4.19) can be implemented as a recurrent convolutional network in which the output of a layer is directly fed back to its input as shown in Fig. 5.3. At every recursion of the network, a prediction for the next time step can be extracted. As initial state of the network, a distribution grid may be obtained from tracking information.

A convolutional layer alone will only perform the dynamics-based prediction but is unconscious of the environment. To introduce the impact of the scene layout on pedestrian motion, the factorization of Eq. (4.28) can be used. Again, this requires a discretized grid.

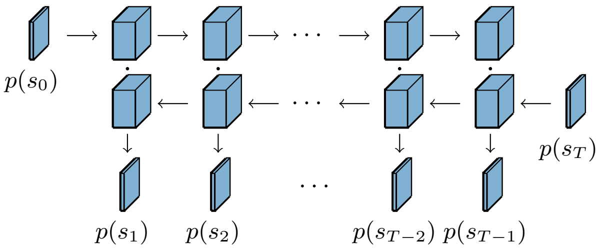


Figure 5.4.: Forward-Backward-Architecture. The forward pass models motion from start to every state in the grid, the backward pass models motion from every state towards the destination. Every block resembles the architecture from Fig. 5.3. The product of the two represents the PDF of the pedestrian's position at different time steps.

For the scene layout, the grid represents the probability of a pedestrian using a specific transition action in that grid cell. This is incorporated into the recurrent network by multiplication of the state grids with the grid for the probability of actions.

Equivalently to the forward prediction, the same can be done for the backward recursion as in Eq. (4.21). For the backward propagation, the initial grid should represent the probability of possible destinations. Furthermore, the filter masks have to be flipped in order to represent motion towards the destination rather than away from it.

Finally, the complete prediction of transition from start to destination has to be computed. For this, the results of both, forward and backward pass, are multiplied according to Eq. (4.28). For every time step within the prediction horizon, the corresponding grids are multiplied element-wise. The result are the unnormalized joint distributions of the prediction away from the start and towards the destination. Figure 5.4 visualizes this process: beginning with a start distribution grid for $p(s_0)$ and one for the destination $p(s_T)$, the forward and backward predictions can be computed recursively. When the recurrent layers are unrolled in time (Fig. 5.4), joint distributions per time step can be computed. Finally, each joint grid can be normalized to sum to one to yield a valid probability distribution.

The entire Forward-Backward Network is constructed exclusively of fully differentiable functions. Consequently, the entire network itself can be trained in a standard backpropagation framework. The set of variable parameters are the transition filter masks and what the definition of the map of $p(a|\Theta)$ requires. Hence, given start and destination, these two parameter sets fully define the prediction.

5.2.2. Markov Decision Process Network

As we have seen before, prediction can be modeled as Markov Decision Process (MDP). Furthermore, Value Iteration (VI) was introduced as a possible solution for MDPs. Section 4.3 showed that VI can be reformulated using recurrent convolutions. This property enables the use of CNNs for this task.

The application of MDPs for prediction consists of two parts, namely the VI part to generate the optimal policy and a simulation part that plays out the optimal policy to generate actual predicted trajectories. Note that, while the trajectory is a function in time, the policy is static as it only resembles what action to take per state. To model this MDP as a CNN, each of the two parts has to be modeled as a separate neural network.

The structure of the Value Iteration Network can directly be derived from Eq. (4.35) [155, 163]. An initial grid is generated that represents the value function $V_0(s)$. Next, a set of actions $\mathcal{A}$ is defined, where each action $a{\in}\mathcal{A}$ represents a characteristic motion pattern. Each of these actions can be associated with a transition probability distribution $p_a(s, s')$ of reaching state s' when taking action a in state s. When the value grid $V_0(s)$ is convolved with one of these filters, it computes the new expected reward per state when taking action a, $V_{1,a}(s)$. This process is repeated for every possible action in $\mathcal{A}$. This generates $|\mathcal{A}|$ grids, all with the same extent as $V_0(s)$. Each of these grids per action a holds the expected reward for every cell when taking the respective action.

For every action and state, a reward of taking that action in that specific state is associated, again reflected by $|\mathcal{A}|$ grids. These rewards $R_a(s)$ are added to the corresponding value maps $V_{1,a}(s)$. These two operations together form the argument of the max of Eq. (4.34). The full value function update Eq. (4.34) is computed as the cell-wise maximum of the sum of maps $V_{1,a}(s) + R_a(s)$, taken along the action dimension.

In the context of convolutional neural networks, the propagation of rewards for different actions can be modeled as a convolutional layer with a single input and $|\mathcal{A}|$ output channels and thus, filter masks. As before, Eq. (5.18) satisfies the PDF's constraints by exponentials together with layer-wise normalization. The reward per cell and action can be considered as an additive bias layer. Finally, the maximum expected reward per cell is obtained from maximum pooling in the channel direction as these reflect the different actions. The result of this pooling operation then is the updated value function $V_{k+1}(s)$. Figure 5.5 visualizes this architecture.

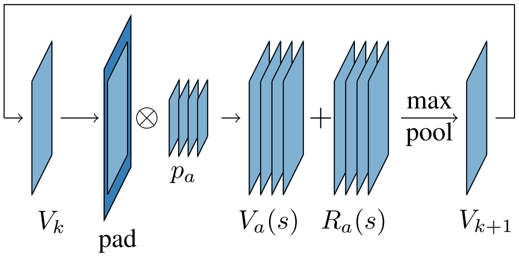


Figure 5.5.: Recurrent stage of Value Iteration Network. The current value grid is convolved with action filters to propagate the expected reward. Note that padding is explicit since implicit padding might falsely introduce rewards in border regions. Then, the newly obtained reward per state and action is added. The max-pooling extracts the maximally achievable reward per state.

The VI Network has to be run until convergence, i.e.

$$V(s) := V_{k+1}(s) \approx V_k(s). \tag{5.19}$$

In CNN accelerators, it is more efficient to run the VI network for a fixed amount of steps. These have to be sufficient so that convergence is approximated. This is in contrast to the forward backward network in which we have to run both forward and backward pass only for as many steps as we would like to predict time steps into the future.

The value iteration algorithm generates a policy for optimal action selection along the path. For this, the VI network is run one last time up to the maximum pooling stage. Instead of running pooling now, the action index of the maximum in channel direction is extracted. This yields the action per state that leads to the maximum expected reward in that state and, thus, the optimal policy $\pi(s)$.

Once the optimal policy is generated, it is simulated to obtain the actual prediction PDFs. In order to incorporate the optimal policy $\pi(s)$, it is represented as a grid in one-hot representation. For each cell in the grid, $|\mathcal{A}|$ channels are defined, where all entries in channel direction are filled with zeros except the one corresponding to the optimal action for that state. This entry is set to one. For brevity, this grid is referred to as $\pi_a(s)$.

Simulation of the policy is again done using a recurrent CNN, which is shown in Fig. 5.6. Initially, a grid $p(s_0)$ is created that represents the PDF for the current pedestrian's position. However, for simplicity let us assume that we perfectly know the exact cell of the grid that the pedestrian is currently located in.

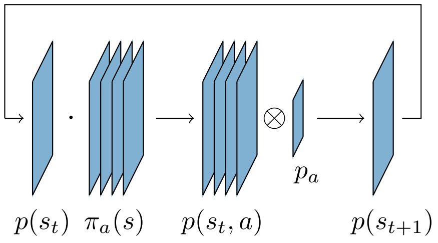


Figure 5.6.: Network to simulate the policy generated by the VI net. The policy grid is multiplied with the current state grid to select actions per state. These are convolved with the corresponding action filters. The result is the updated state grid.

In that case, we can initialize a grid that is all zero except in that one cell where it is set to one. This means we know what state the pedestrian is in and now have to select the optimal action for that state according to $\pi(s)$. The one-hot grid $\pi_a(s)$ represents exactly this selection. Thus, every channel of $\pi_a(s)$ is multiplied with the state grid $p(s_0)$. The outcome now has $|\mathcal{A}|$ channels, but only features a single one in the entire grid, namely the one selecting the optimal action for the current state s_0. To execute this action, this grid is convolved with the transition filter masks that have already been used for the VI network. However, in this case, not the sum over future rewards per action is of interest but the future states for the action taken. Thus, the transition filter masks have to be flipped in spatial directions and exchange the input and output dimensions. This way, the product of $p(s_0)$ and $\pi_a(s)$ gets reduced from channel count $|\mathcal{A}|$ back to a channel count of one. This result can readily be interpreted as $p(s_1)$. To obtain the next prediction step, this result is fed back to the network and propagate probabilities through the space according to $\pi(s)$.

5.2.3. Planning Network Training

The prediction scheme as a neural network can be trained from observed examples. For this, the backpropagation algorithm needs to be applied, which in turn requires gradient computation for all operations. While the two individual parts of the MDP network are fully differentiable, the complete network is not. This is due to the two operations of creating the policy map $\pi(s)$ and $\pi_a(s)$ as argmax and one-hot operations are not differentiable. Thus, to train the MDP network in backpropagation, these operations have to be circumvented. Taking the softmax over the channel direction is a simple and established solution to this problem [155, 163].

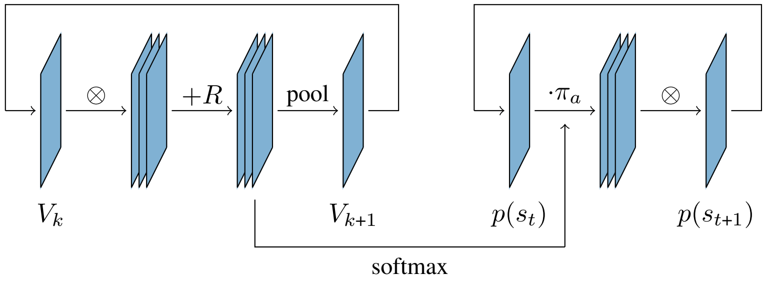


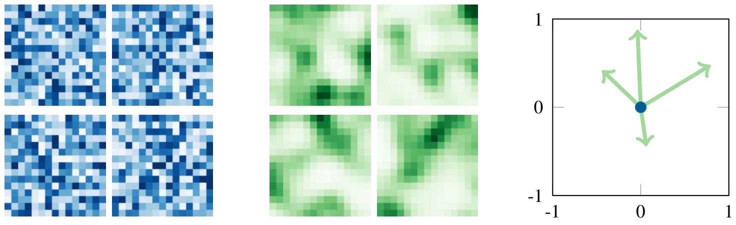
Figure 5.7.: MDP-Network. The left part performs Value Iteration to find the policy, the right part simulates it. The softmax function is used to connect the two in a differentiable fashion. This way, the VI network can be trained from loss computed for the simulation results. Details on the two parts can be found in Fig. 5.5 and Fig. 5.6.

By use of the softmax, the entire setup is fully differentiable. The interconnection between VIN and policy simulation in Fig. 5.7 now allows for unhindered gradient flow.

In training of the full MDP net, the policy changes depending on the shape of the transition PDFs which in turn are optimized depending on the selected policy. This is a classical chicken-and-egg problem since one would need to know the policy to train the transition filters but need transition filters to obtain the policy. Other works avoid this problem by defining transition PDFs by hand or manually annotating policies [155, 163]. In a setting where neither action PDFs nor policies can be observed, e.g. traffic, this is not an option. To overcome this problem, the optimization needs to be guided towards enforcing convergence of valid action PDFs and policies.

Guidance for optimization can be derived from introspection into the nature of the problem itself. As we have seen, the actions $\mathcal{A}$ should represent characteristic motion patterns such as *moving left* or *standing*. These interpretations are characteristic as their motion directions feature a predominant direction. For defining a transition PDF per action, this consideration can be taken into account at initialization. Still, the learning potential of CNNs should be exploited by training the filters. Therefore, they require random initialization.

Plenty of works on neural networks have shown the significance of a good initialization for convergence [55, 61]. According to the considerations from above, the aim is to initialize transition filter masks that define a predominant direction of motion but still allow for randomness.



(a) Xavier [55] (b) Smoothed (c) Expected shift (pixels)

Figure 5.8.: Initialization for filter masks: Xavier initialization [55] (blue) generates zero-mean masks (blue dot in (c)), smoothing (green) leads to directional characteristics (green arrows). Color maps only differ for association between filters and expected shift.

To achieve this, filter weights are initialized randomly from a widespread Normal Distribution to get great parameter variations within one single filter. Next, the weights are smoothed by convolution with a blur kernel. This way, weights become spatially correlated. When they are converted to a transition PDF according to Eq. (5.18), this results in locally focused centers of probability mass in the individual filter masks. Thus, this procedure generates random transition PDF filters with emphasized directional characteristics. As an example, Fig. 5.8 shows the difference between purely random initialization and that with smoothing. Filters generated according to [55] display a noise pattern where the center of mass lies in the filter's center. After smoothing, filter values have pronounced imbalances in space and, thus, feature strong directional properties. Consequently, policies are more meaningful even in the beginning of the training.

Not only the initialization but also regularization can promote directional filters. One particular property of directed filters is low spatial variance. For every filter p_u as defined in Eq. (5.18), the variance can be computed in x- and y-direction as $\sigma^2_{a,x}$ and $\sigma^2_{a,y}$, respectively. Then a regularization loss is introduced to favor smaller variances as

$$L_{\text{var}} = \sum_{a \in \mathcal{A}} \left(\sigma^2_{a,x} + \sigma^2_{a,y} \right) . \tag{5.20}$$

This way, the training is guided towards condensed transition filters.

5.3. Joint Destination and Planning Network

Goal-directed prediction is a joint task of goal estimation and planning towards that goal. Chapter 4 introduced a framework for general goal-directed prediction together with two different planning models that solve this problem. The previous Section 5.1 proposed a neural network architecture to predict pedestrian destinations. Also, it was shown how to implement planning methods as neural networks in Section 5.2. Now, the entire goal-directed prediction task for pedestrian trajectories can be augmented as one monolithic neural network. For this to work, the destination prediction network is used as input for either of the planning networks.

5.3.1. Destination Mixture Density Grids

The destination prediction network presented in Section 5.1 outputs a mixture density function for possible destinations. All planning networks, however, operate on bird's eye view grid maps. Thus, in a first step, the continuous, parameterized output of the MDN needs to be converted into a grid map. Here, we note that the grid maps for planning represent the same domain as the output of the MDN, namely the space (and possibly orientation) the pedestrian occupies. Every cell within the planning grids represents a discrete *space element* of which we can identify the extent in the continuous-valued domain of the MDN output. Therefore, the PDF output of the MDN can be mapped to grid cells.

The grid cells themselves are agnostic of their own location, they only map to metric space through an appropriate conversion function. This is done by a coordinate transform of the grid cells' row and column indices, including scaling. The resulting mapping from grid cell indices to spaces can be used to efficiently map the output of the MDP to the grid domain. For this, the center position and orientation (x, y, ψ) of all cells in space is computed. Next, the argument of the exponentials is expanded in Eq. (5.2) and Eq. (5.3) into parts depending on x, y and ψ, comparable to a radial basis function network [25],

$$\begin{aligned}\beta_i(x,y,\psi) = & - \frac{(x-\bar{x}_i)^2}{2\sigma_{x,i}^2(1-\rho_i^2)} - \frac{(y-\bar{y}_i)^2}{2\sigma_{y,i}^2(1-\rho_i^2)} \\ & + \frac{\rho_i(x-\bar{x}_i)(y-\bar{y}_i)}{\sigma_{x,i}\sigma_{y,i}(1-\rho_i^2)} \\ & + \kappa_i \cos(\psi - \bar{\psi}_i) \qquad (5.21)\end{aligned}$$

with the indices i for the ith mixture component.

The (x, y, ψ)-location of every single grid cell is now substituted as the arguments of $\beta_i(x, y, \psi)$. This way, a grid is filled cell-wise with the argument of the exponential function of Eq. (5.1). After taking the exponential of every grid cell, the grid is normalized so that all cells sum to one to obtain a valid PDF. The actual normalization of Eq. (5.1) is avoided due to the limited grid extent. For the mixture density grid map, one such grid is computed per mixture component and then summed, weighted with the corresponding mixing coefficients. This result can be substituted as the destination PDF map of any planning algorithm.

Note that the only newly introduced operations are the substitutions of the constant positions (x, y, ψ) to construct the grid. This way, the mapping from a continuous mixture density function to a discrete mixture density grid remains fully differentiable. Again, this allows backpropagation for parameter training.

5.3.2. Planning Map Topology

Planning is executed on bird's eye view grid maps that reflect properties of the environment. For that, a map is required that provides information on topological properties of the world that are relevant for motion. These properties may be the presence of obstacles, the location of the sidewalk, but also, even more obvious, the location of the destination. In the planning framework, maps of these given features need to be converted into a representation that is understood by the planning stage. In the case of the Forward-Backward Network, this is the location-dependent probability of actions $p(a|\Theta)$ in Eq. (4.28). For the MDP Network, location information is introduced by the reward term $R_a(s)$ needed in Eq. (4.34).

Regardless of the planning algorithm, however, these topological properties have to be estimated for prediction. Specifically, a concept is required to predict a predefined domain of values per grid cell given features of that cell. This is a classical use case of Fully Convolutional Networks (FCNs) where one prediction is made per input pixel of an image [101]. Thus, *topology maps* are computed, either $p(a|\Theta)$ or $R_a(s)$, using an FCN operating on feature grid maps of the environment. For the sake of completeness, the destination PDF grid from Section 5.3.1 is also included as an additional layer. Since relevant features are the same between Forward-Backward-Net and MDP net, the same FCN architecture can be used for both planning models. All that has to be exchanged is the final activation function. Since the Forward-Backward-Net expects probabilities of actions, the activation function is softmax. For the MDP, linear activation is applied to obtain rewards.

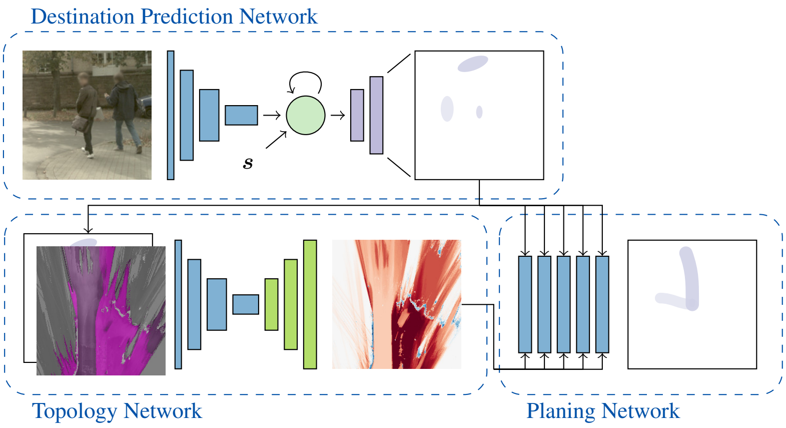


Figure 5.9.: The full network is augmented from the three individual architectures, one for the destination prediction, one that generates the topology map for planning and the final network that executes the prediction in this map given the destinations. The intermediate results are passed between the networks and all connections are differentiable.

5.3.3. Prediction Network Augmentation and Training

While the prediction algorithm can be separated into the tasks of destination inference, topology mapping and planning, they are all integral parts of one complete system. With that, they should also be considered as a joint task. The formulation of the three individual parts as neural networks allows for their integration into one single prediction setup. For this, the output of the destination inference is connected to the input of the topology network and in the end, feed the resulting topology to the planning stage. As the final output of the complete system, individual grid maps are obtained, each representing a PDF for the pedestrian's position at a future time step.

Figure 5.9 visualizes this complete structure. From observations of a pedestrian, a grid of possible destinations are computed. This is concatenated with features of the environment. An FCN infers a topology map for planning from this input. The FCN's output is used as the input for the planning network. Since all of these individual parts are fully differentiable, flow of information can be realized in both directions: forward for inference and backwards for network training. Consequently, when the entire prediction pipeline can be seen as one single network, it can also be trained as such.

Training of neural networks requires a loss function to be minimized w.r.t. the network's parameters as explained in Section 2.2. This loss function will punish the network for wrong predictions. Through minimization of this punishment, the parameters are updated to improve the final output. Previously recorded pedestrian trajectories serve as supervision. In that sense, the training is an imitation learning setup as the network aims to imitate observed behavior of humans.

From observed ground truth pedestrian trajectories, the desired output of the network can be constructed. In an ideal case, the prediction would perfectly predict a pedestrian's future position. In the grid representation, this corresponds to a state grid that is all zeros except for a single one in the cell the pedestrian occupies at a specific time step. For every time step, one such grid can be generated to reflect the ground truth. From the prediction network, comparable grids can be computed, however, now, they resemble the predicted PDF of the pedestrian's future states. With these, the ground truth grid can be compared with the predicted PDF grid from the network.

Let s_t denote the ground truth state of a pedestrian at time instance t. For the current prediction, all input data is provided to the neural network, namely an image of the pedestrian, the pedestrian's current position and a feature map of the environment, condensed in x. With the parameters of RMDN, topology and planning network, condensed in $\hat{\boldsymbol{w}}$, the method outputs a PDF $p(s_t|x, \hat{\boldsymbol{w}})$ per predicted time step t.

In this case, $p(s_t|x, \hat{\boldsymbol{w}})$ is the grid map that stores the probability of the pedestrian occupying every possible cell. On the other hand, the ground truth state grid $q(s_t)$ is a binary grid, where all cells are zero except for the single state that the pedestrian actually occupies.

In the context of neural networks, this can also be understood as a binary classification: either the pedestrian occupies a certain grid cell in the future or not. Accordingly, well-established loss functions from classification tasks can be employed.

One common loss function in classification is the *cross entropy* [56]

$$\begin{aligned} L_{\text{pos}} = &- q(s_t)\log(p(s_t|x, \hat{\boldsymbol{w}})) \\ &- (1 - q(s_t))\log(1 - p(s_t|x, \hat{\boldsymbol{w}})). \end{aligned} \tag{5.22}$$

The complete network can readily be trained from the cross entropy prediction loss. However, convergence may fail for large prediction horizons. This is due to the required destination estimation. It may not converge because the initialization is too uncertain for large prediction horizons.

To overcome this problem, a second loss term is introduced, specifically to guide the RMDN's convergence. The loss is the negative log-likelihood of the predicted PDF evaluated in the ground truth position [21]

$$L_{\text{dest}} = -\log(p(s_T|x, \hat{\boldsymbol{w}})). \tag{5.23}$$

The introduction of this additional loss term is in compliance with many previous publications that introduced additional loss functions to overcome the *vanishing gradient problem* or even only make the network more versatile through multi-task training [45, 84, 161]. The benefit of this loss is the bypassing of the subsequent network in back-propagation. This way, even early stages of the network produce the correct features for the subsequent architecture. Additionally, introduce further loss functions could be introduced to the RMDN, e.g. to predict occlusions, gaze or orientations, to improve CNN training [45, 84].

Especially deep network architectures with millions of parameters are prone to overfitting during training. Parameter regularization mitigates this problem and is thus crucial in training. Reportedly, some CNN architectures do not even converge without regularization [90]. Consequently, parameter regularization as introduced in Section 2.2.4 is applied in all networks as the sum of squares of all network parameters

$$L_{\text{reg}} = \hat{\boldsymbol{w}}^\top \hat{\boldsymbol{w}} \tag{5.24}$$

The final loss function for the full network in training is augmented from the individual parts. These are the prediction loss L_{pos}, the destination loss L_{dest}, parameter regularization L_{reg}, and the filter variance constraint from Section 5.2.3, L_{var}. The full function which is to be minimized then is the weighted sum of losses

$$L = L_{\text{pos}} + \lambda_{\text{dest}} L_{\text{dest}} + \lambda_{\text{reg}} L_{\text{reg}} + \lambda_{\text{var}} L_{\text{var}}, \tag{5.25}$$

where the parameters $\lambda_{\text{dest}}, \lambda_{\text{reg}}, \lambda_{\text{var}}, \lambda_{\text{proxy}} \geqslant 0$ weight the individual loss function parts w.r.t. the prediction cross entropy loss.

The augmentation of a single prediction network and the definition of a joint loss allows for optimal adaption of the individual tasks. Any single network can be trained to only fulfill its own prediction task. However, through the joint loss function in Eq. (5.25), the individual tasks can adopt their behavior to optimally serve the subsequent pipeline. This way, all parts are best fit for the ultimate goal of position prediction.

Chapter 6

Dataset

While many datasets exist for evaluation of recognition systems, they are comparably scarce for pedestrian prediction. In the past, sensor processing already posed such a significant challenge that algorithms in the downstream of the processing chain were rarely evaluated in isolation. Thus, many datasets focus on the evaluation of low-level sensing tasks such as detection and tracking [8, 49, 52, 72]. Only few have been proposed to evaluate pedestrian prediction methods [89, 152]. These, however, mostly feature a limited amount of data or staged scenes. Consequently, there is the need for a suitable dataset in order to train and evaluate the proposed prediction method.

A dataset to assess prediction algorithms requires both, data as well as a benchmarking methodology. The data used in this thesis consists of recorded video sequences of pedestrians. For every pedestrian, ground truth trajectories are generated that are used for both, training and evaluation. Additionally, observations of the environment are extracted in form of semantic grid maps. For benchmarking, an algorithm's expected correctness should be quantified by suitable metrics. But beyond that, these metrics should also reflect the expected *safety* since a single missed prediction might lead to disastrous results

This chapter presents the dataset for evaluation of prediction. First, a detailed overview is given on how the training and evaluation data was created. Then, metrics are presented that allow to assess prediction algorithms in terms of both, accuracy and safety. The dataset and subsets of the metrics were also used for evaluation in [133].

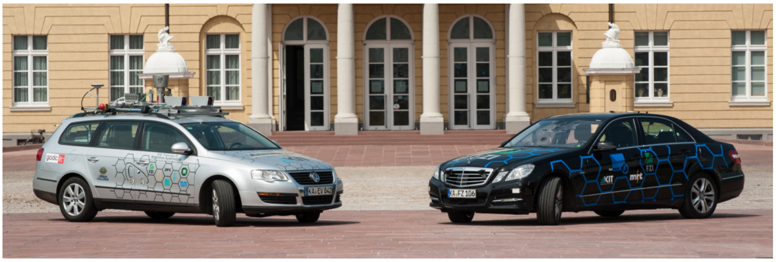


Figure 6.1.: Experimental vehicles of the Institute of Measurement and Control Systems. AnnieWay (left) used in the DARPA Urban Challenge [80] and BerthaOne (right) used in preparation of the Bertha Benz Memorial Drive [184].

6.1. Prerequisites

In order to establish a meaningful dataset, a system setup is desired that can provide all information required for prediction. In research, the common approach is to equip an experimental vehicle with the most accurate sensors available to record ground truth data. Yet, with recent advances, automated vehicles are expected to reach a state of production within the next decade [51]. Consequently, a transition from large and costly sensors towards a smaller and cheaper setup is desired, as seen in the experimental vehicles shown in Fig. 6.1.

This work pays tribute to both paradigms, the high-precision and high-cost sensors for ground truth as well as the potential implementation in production line vehicles. As a sensor setup, a stereo camera system is selected, similar to the ones introduced to mass production. However, due to demands on precision, high-resolution cameras are deployed with industrial-grade lenses shown in Fig. 6.2.

The stereo camera was selected since it provides rich information for both, self and scene perception. Not only does it provide visible light imaging but also means to compute scene depth from two camera images as well as motion estimation from successive image frames. Thus, a stereo camera alone can fulfill all requirements of this work. Consequently, no other sensor is employed for evaluation.

6.1.1. Stereo Setup

The stereo setup is made up from one GENIE TS-C4096 with Bayer Pattern and one monochromatic GENIE TS-M4096. As lenses for the system, industrial-grade ZEISS DISTAGON T* 2.8/15 ZF-I lenses were cho-

Parameter	Left	Right
Chroma	Color	Mono
Resolution	1536×4092	768×2048
Focal Length	15mm	15mm
Frame Rate	10Hz	10Hz
Baseline	-	50cm

Table 6.1.: Parameters of the stereo camera setup.

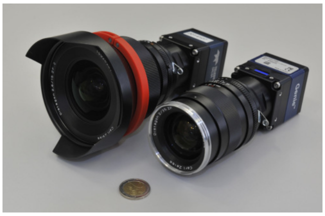


Figure 6.2.: The cameras used in this work.

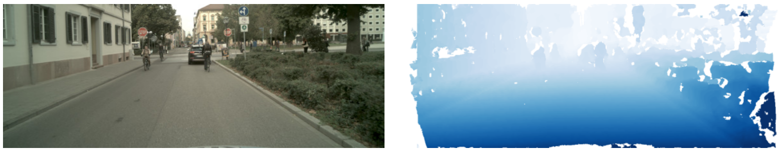


Figure 6.3.: Camera image and corresponding disparity image obtained with [126].

sen with a horizontal field of view of roughly 80°, equipped with IR-cut filters. The parameters of the system are summarized in Table 6.1.

Due to bandwidth limitations, the right camera is recorded at a quarter of the resolution of the left camera. To further reduce the amount of data, a person detector was run during recording and only scenes were captured where pedestrians were present. This scheme reduced the time of recorded video to a third of the total time of driving.

6.1.2. Disparity Computation

A stereo camera allows the perception of scene depth per pixel. For this, the horizontal displacement of a point between left and right image, the *disparity*, is computed. With known camera calibration, the distance of the point can be triangulated.

The cameras have been calibrated prior to data recording by use of coded checkerboard targets [159]. For disparity computation, the left camera image is resized to the size of the right and convert it to grayscale. This also mitigates the impact of the left camera's color pattern on stereo matching. Disparity images are obtained through slanted plane stereo matching [126]. Exemplary results are shown in Fig. 6.3.

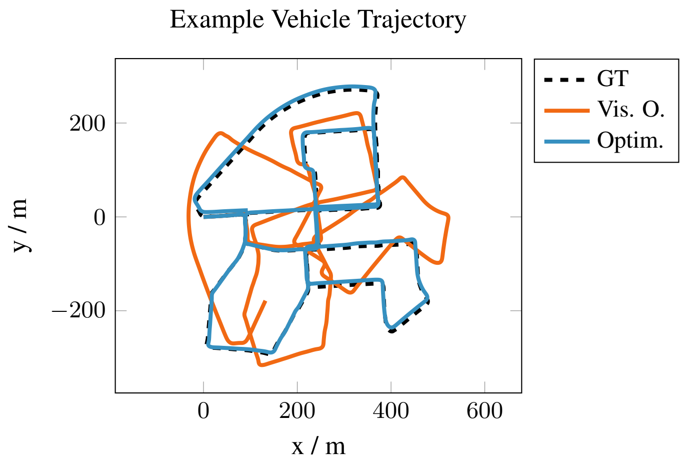


Figure 6.4.: Vehicle trajectory before and after optimization, generated for an exemplary KITTI odometry sequence [52].

6.1.3. Vehicle Motion Estimation

Estimation of pedestrian trajectories in the world requires knowledge of the vehicle's motion. From stereo vision, the ego motion is estimated from image sequences via bundle adjustment [4, 166].

With a sequence of camera images of the same scene, bundle adjustment aims to optimize the camera poses and observed 3D structure jointly. For this, the reprojection error of 3D point features into the camera image is minimized. This requires 3D point features $\boldsymbol{x}$, also called landmarks, the camera pose $\boldsymbol{T} \in SE(3)$ and the camera's projection matrix $\boldsymbol{K}$. Then, the projection function π that projects $\boldsymbol{x}$ to an image point $\boldsymbol{p}$ is defined as

$$\pi(\boldsymbol{T}^{-1}\boldsymbol{x}) := \frac{1}{z}\boldsymbol{K}\boldsymbol{T}^{-1}\boldsymbol{x}, \tag{6.1}$$

where z is a normalization to convert $\boldsymbol{T}^{-1}\boldsymbol{x}$ to homogeneous coordinates. The reprojection error $\hat{\boldsymbol{e}}$ is the difference between an observed point $\boldsymbol{p}$ and the estimated point projection $\hat{\boldsymbol{p}}$

$$\hat{\boldsymbol{e}} = \hat{\boldsymbol{p}} - \boldsymbol{p} = \pi(\hat{\boldsymbol{T}}^{-1}\hat{\boldsymbol{x}}) - \boldsymbol{p}, \tag{6.2}$$

where now both, the camera's pose $\hat{\boldsymbol{T}}$ and the landmark's 3D position $\hat{\boldsymbol{x}}$ are subject to optimization.

For bundle adjustment, a set of landmarks $\hat{\mathcal{L}} = \{\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_N\}$ and poses $\hat{\mathcal{T}} = \{\hat{\boldsymbol{T}}_0, \hat{\boldsymbol{T}}_1, \ldots \hat{\boldsymbol{T}}_M\}$ is optimized to minimize the reprojection error of all landmarks

$$J(\hat{\mathcal{T}}, \hat{\mathcal{L}}) = \sum_{i=0}^{M} \sum_{j=0}^{N} \mathbb{1}_v \cdot (\|\pi(\hat{\boldsymbol{T}}_i^{-1} \hat{\boldsymbol{x}}_j) - \boldsymbol{p}_{i,j}\|^2), \tag{6.3}$$

where $\mathbb{1}_v \in \{0, 1\}$ is the indicator function that only evaluates to 1 if landmark j was observed from the i-th camera pose and $\boldsymbol{p}_{i,j}$ is that observation in the corresponding video frame.

Vehicle poses are obtained by minimizing Eq. (6.3) w.r.t. suitable parameters. For this purposes, the vehicle is assumed to move on a planar surface and motion is constrained by the kinematic single track [138].

The pose increment $\Delta\hat{\boldsymbol{T}}_i$ can be parameterized in traveled distance $\hat{l}$ and curve radius $\hat{r}$ with

$$\Delta\hat{\boldsymbol{T}}_i = \begin{bmatrix} \cos\hat{\psi} & -\sin\hat{\psi} & 0 & \hat{r}\sin\hat{\psi} \\ \sin\hat{\psi} & \cos\hat{\psi} & 0 & \hat{r}(1-\cos\hat{\psi}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6.4}$$

where ψ is the curve angle defined as ${}^{\hat{l}}/_{\hat{r}}$.

Planar motion according to Eq. (6.4) is not sufficient to describe the full camera motion. Furthermore, pitch and roll movements need to be accounted for. Therefore, an additional two-angle bearing of the vehicle $\hat{\boldsymbol{R}}_i$ is estimated for every time step. This is combined with the camera's extrinsic pose $\boldsymbol{T}_c$ to explain its full motion in-between time steps. The final absolute camera pose $\hat{\boldsymbol{T}}_i$ at every time step i is a conjunction of all individual poses

$$\hat{\boldsymbol{T}}_i = \boldsymbol{T}_c \hat{\boldsymbol{R}}_i \Delta\hat{\boldsymbol{T}}_i \Delta\hat{\boldsymbol{T}}_{i-1} \cdots \Delta\hat{\boldsymbol{T}}_0. \tag{6.5}$$

In the final optimization, the pose from Eq. (6.5) is substituted for every observation in Eq. (6.3) and the loss function is minimized w.r.t. all parameters of Eq. (6.4) as well as $\hat{\boldsymbol{R}}_i$. The mounting position $\boldsymbol{T}_c$ and the projection matrix $\boldsymbol{K}$ can be obtained from calibration [159].

In Fig. 6.4, an exemplary result from a sequence from [52] is shown. While incremental visual odometry drifts away from the ground truth trajectory, the result after bundle adjustment stays relatively close to the ground truth. Note that no loop closure detection was employed.

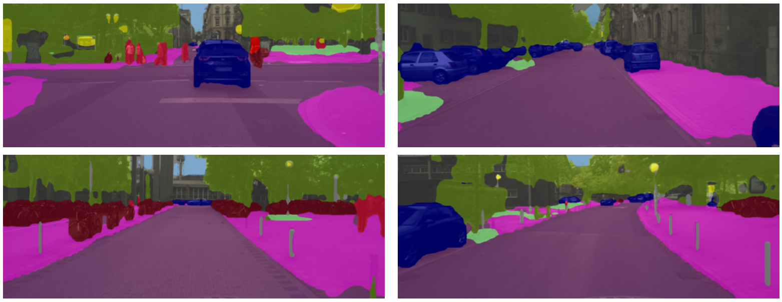


Figure 6.5.: Semantic segmentation of images predicted with an FCN.

6.1.4. Semantic Segmentation

As motivated in Section 1.3, the layout of the environment is one of the key features for prediction. Therefore, it is desired to obtain a deeper understanding of the scene's properties for prediction. This is done via *semantic segmentation* of images, where a semantic class is assigned to every pixel [31, 172, 181]. With this information, parts of the scene can be recognized that are accessible for a pedestrian or even identify customary behavior depending on traffic scene layouts.

In this work, a Fully Convolutional Network (FCN) is employed for semantic segmentation [101] with a GoogleNet feature extractor [161]. The network is initialized with weights for ImageNet classification [146] and trained on the Cityscapes dataset to perform segmentation [37]. Examples are shown in Fig. 6.5

6.1.5. Mapping

The planning networks introduced in Section 5.2 operate on grid structures. To incorporate information on the environment into these grids, occupancy grid mapping [46, 58, 127] is used. For this type of mapping, the vehicle's surrounding is discretized into equally sized cells. Every cell reflects the probability that the cell features a certain property (cf. Fig. 6.6).

Similar to [76], a multi-layered map is generated in which each layer represents one specific property of the environment. One such property is the presence of obstacles. Additionally, one layer is added for the bird's eye view (BEV) of the scene and one for every semantic feature from the segmentation of Section 6.1.4. The update strategy of map cells follows

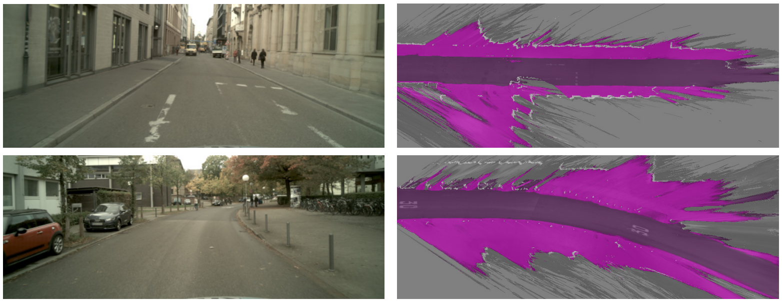


Figure 6.6.: Camera images and multi-layer map of the corresponding scene. The vehicle faces right the maps.

[46]. For the BEV, an exponential average filter is used to track the color value s_t of every cell,

$$s_t = \alpha s_{t-1} + (1 - \alpha) z_t, \tag{6.6}$$

where z_t is the currently observed color value. Here, $\alpha \in (0, 1)$ is a damping parameter that governs the filter's response [76]. Upon initial update of a cell, s_0 is set to the first measurement z_0.

6.2. Pedestrian Data

Evaluation of pedestrian prediction necessitates ground truth pedestrian trajectories. For this purpose, all pedestrians were annotated in video sequences and enriched with further meta information in post-processing.

6.2.1. Manual Annotations

The first step towards ground truth pedestrian trajectories is manual annotation of video sequences. All pedestrian appearances are annotated with enclosing 2D bounding boxes. Individual persons are assigned consistent identification numbers (IDs) over time so that re-identification is possible. Some exemplary annotations are visualized in Fig. 6.7.

As pedestrians might be occluded in the course of a single sequence, we also introduce occlusion tags per bounding box. This allows to explicitly handle it in later stages.

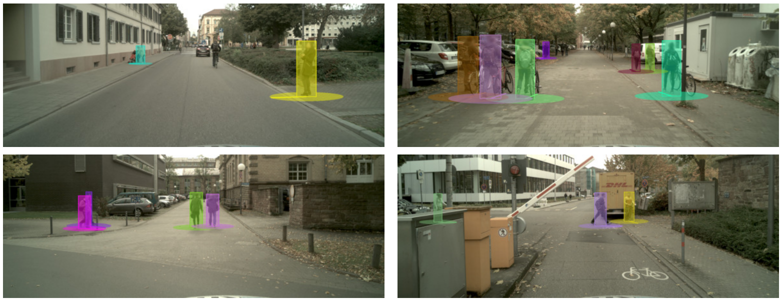


Figure 6.7.: Pedestrians annotations in images. Information includes the enclosing bounding box, position w.r.t. the vehicle and occlusion.

6.2.2. Trajectory Estimation

A full ground truth trajectory is computed from optimization over all individual sightings of the pedestrian. A trajectory is defined as a sequence of N states $\hat{\zeta}{=}(\hat{\boldsymbol{s}}_1, \hat{\boldsymbol{s}}_2 \ldots, \hat{\boldsymbol{s}}_N)$, where each state consists of a three-dimensional vector $\hat{\boldsymbol{s}}_t{=}(\hat{x}_t, \hat{y}_t, \hat{z}_t)^\top$, representing the center of mass of the pedestrian. Also, the derivatives w.r.t. time are computed, $\hat{\dot{\boldsymbol{s}}}_t{=}(\hat{\dot{x}}_t, \hat{\dot{y}}_t, \hat{\dot{z}}_t)^\top$ to obtain velocity. Additionally, the constant height of the pedestrian $\hat{k}$ is estimated. Note that all trajectory states $\hat{\boldsymbol{s}}_t$ are given in the same reference coordinate frame. The pose T_t from Eq. (6.5) is used to obtain the state vector in camera coordinates $\tilde{\boldsymbol{s}}_t{=}(\tilde{x}_t, \tilde{y}_t, \tilde{z}_t)^\top$

$$\tilde{\boldsymbol{s}}_t = \boldsymbol{T}_t^{-1} \hat{\boldsymbol{s}}_t. \tag{6.7}$$

The annotated bounding boxes provide the input information for optimization. For every time step t of a trajectory, they consist of the u- and v-position of the box, its width w and height h in the image together with a binary occlusion label $o_t{\in}\{0, 1\}$. For every pixel (u, v) within a box, disparity information $d_t(u, v)$ is obtained from stereo matching. From these observations, a set of residuals is constructed per annotation which is then minimized w.r.t. the parameters of the trajectory $\hat{\zeta}$. The individual residuals as well as the optimization are explained in the following.

Bounding Box The first residual is the displacement between the reprojection of the center of mass from the center of the bounding box

$$\hat{\boldsymbol{e}}_{c,t} = \begin{pmatrix} f\frac{\tilde{x}_t}{\tilde{z}_t} + c_u \\ f\frac{\tilde{y}_t}{\tilde{z}_t} + c_v \end{pmatrix} - \begin{pmatrix} u_t + \frac{w_t}{2} \\ v_t + \frac{h_t}{2} \end{pmatrix}, \tag{6.8}$$

where f is the camera's focal length and $(c_u, c_v)^\top$ is the center of projection, both in pixels.

Secondly, the deviation between the projection of the pedestrian's height and the height of the bounding box is used as residual,

$$\hat{e}_{h,t} = f\frac{\hat{k}}{\tilde{z}_t} - h_t. \tag{6.9}$$

Disparity For disparities, a boolean validity flag is obtained per pixel (u, v), $g_t(u, v) \in \{0, 1\}$, where $g_t(u, v) = 1$ means that disparity computation for this pixel was successful. The residual in disparity is

$$\hat{e}_{d,t}(u, v) = \mathbb{1}_o \cdot \left(g_t(u, v) \cdot \left(\frac{fb}{\tilde{z}_t} - d_t(u, v) \right) \right), \tag{6.10}$$

where $\mathbb{1}_o \in \{0, 1\}$ denotes a selector function that only evaluates to 1 if the pedestrian is not occluded.

Constant Velocity The velocity estimation yields two different residuals. For once, a residual is computed between the position at $t + 1$ as predicted from position and velocity of t,

$$\hat{\boldsymbol{e}}_{p,t} = \begin{pmatrix} \hat{x}_t \\ \hat{y}_t \\ \hat{z}_t \end{pmatrix} + \Delta t \begin{pmatrix} \hat{\dot{x}}_t \\ \hat{\dot{y}}_t \\ \hat{\dot{z}}_t \end{pmatrix} - \begin{pmatrix} \hat{x}_{t+1} \\ \hat{y}_{t+1} \\ \hat{z}_{t+1} \end{pmatrix}. \tag{6.11}$$

Secondly, another residual is introduced for constant velocity as

$$\hat{\boldsymbol{e}}_{v,t} = \begin{pmatrix} \hat{\dot{x}}_t \\ \hat{\dot{y}}_t \\ \hat{\dot{z}}_t \end{pmatrix} - \begin{pmatrix} \hat{\dot{x}}_{t+1} \\ \hat{\dot{y}}_{t+1} \\ \hat{\dot{z}}_{t+1} \end{pmatrix}. \tag{6.12}$$

Both residuals can be computed for every observation except the last.

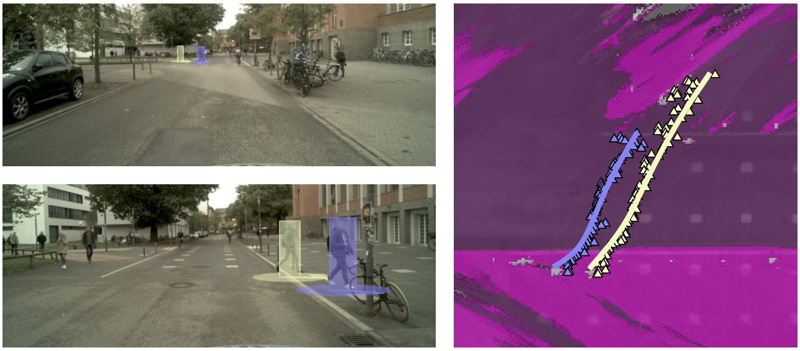


Figure 6.8.: Trajectory optimization: from labels in images (left), initial position estimates are created (right, triangles). These are then optimized for ground-truth-like trajectories (right, solid lines). Note: the gray blobs at the lower end of the optimized trajectories are measurements of the two pedestrians accumulated in the grid.

Final Optimization All residuals are incorporated into one single cost function as the sum of squared residuals from Eqs. (6.8) to (6.12). Since everything else is annotated manually, the disparity measures are the only possible source of outliers. To reduce their impact, the Cauchy Loss function is applied to the disparity residual $\hat{e}_{d,t}(u,v)$.

The final cost function is

$$
\begin{aligned}
J(\hat{\zeta},\hat{l}) = \quad & \underbrace{\sum_{t=1}^{N} \left[\|\hat{\boldsymbol{e}}_{c,t}\|^2 + \|\hat{e}_{h,t}\|^2 \right]}_{\text{Projection of bounding box}} \\
+ & \underbrace{\sum_{t=1}^{N} \sum_{(u,v)\in b_t} \log(1 + \|\hat{e}_{d,t}(u,v)\|^2)}_{\text{Disparity}} \\
+ & \underbrace{\sum_{t=1}^{N-1} \left[\|\hat{\boldsymbol{e}}_{p,t}\|^2 + \|\hat{\boldsymbol{e}}_{v,t}\|^2 \right]}_{\text{Constant velocity}} . \qquad (6.13)
\end{aligned}
$$

Eq. (6.13) is minimized w.r.t. all states within the trajectory $\hat{\zeta}$ together with the person's height $\hat{k}$. The optimization is solved using the Levenberg-Marquardt Algorithm [3].

All positions are initialized from median disparity if a person is not occluded. From this, an initial estimate for the person's height can be derived and, conclusively, all positions can be initialized from the bounding box size and position, even through occlusions. If a person is partially occluded for every sighting, an initial guess of 1.8m is used as height and initialize from this.

An example for optimized trajectories is depicted in Fig. 6.8. The noisy stereo measurements are used as initialization. After optimization, smooth trajectories have been generated that resemble the pedestrians' actual motion in the scene.

6.3. Dataset Statistics

For the dataset, we recorded and annotated 40 minutes of video in total. Of this, 72 individual sequences were selected from urban, commercial and residential Karlsruhe and on campus. The sequences account for almost 9,000 video frames with over 400 annotated pedestrians in them. For independence in evaluation, the data is split into training and test set. The dataset statistics are summarized in Table 6.2.

The distribution of track lengths gives an insight into the expected input data for prediction algorithms. Fig. 6.9 displays their frequency. Depending on the desired prediction horizon, these determine the amount of data available for training. If, for example, we would like to predict trajectories for a time span of four seconds, we can only use tracks that are longer than that. However, of course, longer sequences can be cut into possibly many shorter segments.

	Training Set	Test Set
Number of Sequences	45	27
Total Duration	653.4s	233.9s
Pedestrian Tracks	330	77
Cumulative Track Length	2154s	414s
Avg. Track Length	6.5s	5.4s

Table 6.2.: Overview over dataset statistics.

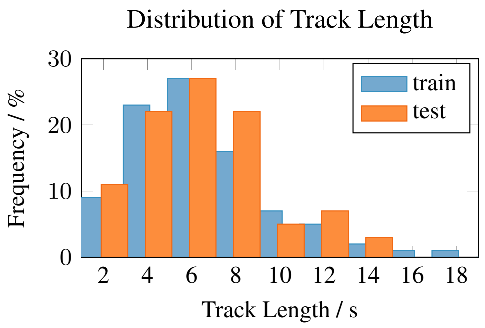


Figure 6.9.: Histogram of track lengths in bins of 2s for both, training and test data.

6.4. Metrics

Throughout this work, it was argued that prediction has to be treated as a probabilistic problem and dealt with in terms of uncertainties. Consequently, metrics to evaluate prediction should pay respect to this. Also, we need to be able to compare parametric models, such as Gaussian Distributions, and non-parametric models.

In order to compare algorithms, a unified representation is required for predictions, regardless of their original parameterization. Grid mapping presents as such a representation. After that, evaluation of predictions will be explained using these grid maps.

6.4.1. Unified Representation for Evaluation

While it may be difficult to obtain a parametric representation from a discretized distribution grid, it is comparably easy to map a parametric distribution to a grid.

For all evaluations, regardless of the approach, the predicted probability distributions at time t is mapped to a grid, Φ_t, of size $W \times H$. This grid consists of fixed size cells $s_{i,j}$, where the indices i, j correspond to the locations within the grid. We then demand

$$0 \leq s_{i,j}, \quad \forall\, i \in 1, 2, \ldots, H,\ j \in 1, 2, \ldots, W, \tag{6.14}$$

$$\sum_{i=1}^{H} \sum_{j=1}^{W} s_{i,j} = 1. \tag{6.15}$$

For the sake of legibility, the indices i, j are omitted in the following.

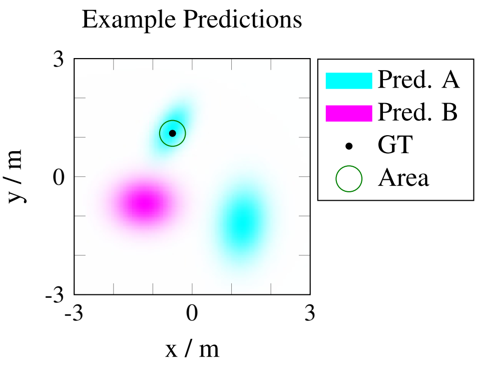


Figure 6.10.: Toy examples of predicted probability density functions (cyan, magenta) with ground truth pedestrian position (black). The green circle approximates the area covered by a standing adult.

Figure 6.10 shows such a mapping. For the magenta ellipse, a single bivariate Gaussian Distribution was mapped onto the grid. The cyan-colored ellipses are a mixture of two such Gaussians.

In contrast to the trajectory, a predicted path is concerned only with the question *where* a pedestrian will be but not *when*. This is represented as a grid without time dependency, Φ_{Path}. It represents the estimated probability that a pedestrian will *ever* occupy a specific cell,

$$s_{Path} = 1 - \prod_t (1 - s_t). \tag{6.16}$$

The same way predictions are mapped to a grid, the ground truth has to be represented as such. Given a pedestrian's position $s_{GT}{=}(x_{GT}, y_{GT})^\top$, the distance to every cell s's center position $s_s{=}(x_s, y_s)^\top$ is computed according to

$$d_s = \|s_s - s_{GT}\|_2. \tag{6.17}$$

With this, a binary grid Φ_{GT} is created that encodes whether a cell is occupied by a pedestrian or not,

$$s = \begin{cases} 1, & \text{if } d_s \leq r \\ 0, & \text{otherwise,} \end{cases} \tag{6.18}$$

	mPP[%]	mNLP	Mode Dist. [m]
Pred. A	**5.76**	**2.85**	2.92
Pred. B.	0.00	12.71	**1.93**
Mean	2.88	7.78	2.42

Table 6.3.: Different metrics evaluated for the example prediction from Fig. 6.10. For mean predicted probability (mPP), better predictions show higher values, perfect score is 100%. For mean log predicted probability (mNLP) and mode distance, good prediction is reflected by lower values, perfect score is zero.

where r is a radius around the pedestrian's position s_{GT} within which the area is considered occupied.

The radius r is selected such that the area covered by ground truth totals to 0.15m^2, which is approximately the space covered by the average standing adult [174]. In Fig. 6.10, this area is shown as the green circle around the ground truth position (black dot).

6.4.2. Probability Distribution Metrics

For evaluation of predicted probability distributions, it is necessary to quantify how well they align with the actual ground truth. Throughout literature, no universal metrics have yet been agreed upon [124, 143, 176]. It is, for example, customary to measure the distance between the maximum mode of a distribution and ground truth [88, 152]. However, this does not incorporate possible multi-modalities. Let us consider the example from Fig. 6.10. Even though Prediction B assigns little probability mass to the actual ground truth, its maximum mode's distance is smaller than Prediction A's (cf. Table 6.3).

To overcome this problem, instead, the probability mass that falls into the area covered by the pedestrian is evaluated [133]. Since the ground truth grid Φ_{GT} is binary, the probability mass of the predicted map Φ_{Pred} that coincides with the ground truth is the Frobenius Product of the two. Given a dataset $\mathcal{D}$, the mean predicted probability (mPP) of the ground truth positions is computed as

$$mPP = \frac{1}{|\mathcal{D}|} \sum_{\zeta \in \mathcal{D}} \frac{1}{N_\zeta} \sum_{\Phi_{Pred,GT} \in \zeta} \langle \Phi_{Pred}, \Phi_{GT} \rangle_F, \tag{6.19}$$

where N_ζ denotes the number of individual observations in a single trajectory. This averaging is done to avoid a bias towards long trajectories in evaluation. Note that this metric is only comparable if maps are normalized according to Eq. (6.15).

The metric defined in Eq. (6.19) can be seen as a measure for average correctness of predictions. Unfortunately, it does not give insights about failure cases. Table 6.3 shows the results for the example in Fig. 6.10. While Prediction A performs well, Prediction B is completely off. In the mean of the two, however, this can't be distinguished from the case where both predictions had included the ground truth with 2.88%, which arguably would be the safer prediction.

As a metric for safety, measure should heavily penalizes erroneous predictions. For this, the mean of the negative log probability (mNLP) is employed [183, 86],

$$mNLP = -\frac{1}{|\mathcal{D}|}\sum_{\zeta\in\mathcal{D}}\frac{1}{N_\zeta}\sum_{\Phi_{Pred,GT}\in\zeta}\log\left(\langle\Phi_{Pred},\Phi_{GT}\rangle_F\right). \tag{6.20}$$

As the predicted probability approaches zero, the negative log probability approaches infinity. It therefore reflects the prediction's *safety*. This can also be seen in Table 6.3: suppose we compared an Algorithm 1, producing Prediction A, to Algorithm 2, returning both, A and B. We therefore compare the value of Prediction A to the mean in Table 6.3. While Algorithm 2 seems to be still half as good as Algorithm 1 in mPP, the mNLP has increased by almost the threefold through the misplaced Prediction B. A comprehensive visualization is presented in Fig. 6.11, where mPP is plotted over mNLP. The further to the top left, the better the prediction.

6.4.3. Classification Metrics

The probability metrics require normalized distributions per grid to be meaningful. In path prediction, however, the grid maps resemble a probability distribution only per cell, but not per grid. Thus, neither the previously presented predicted probability nor log probability are applicable for evaluation of path prediction.

Luckily, the prediction within a grid map can also be considered as a classification problem. The probabilities from Eq. (6.16) can be thresholded to make a decision *is the cell part of the path or not?* With the decision made, classification evaluation concepts can be utilized, such as receiver-operating characteristics (ROC) [69] or Precision-Recall-Curves (PR-Curves) [75]. Both curves can be reduced to a single value by computing the area under curve (AuC), AuROC and AuPR respectively.

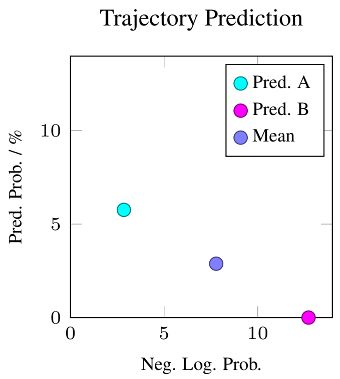


Figure 6.11.: Map of prediction results from example predictions of Fig. 6.10. The further to the top left, the better the prediction performance.

When we consider prediction as classification, we face a severe class imbalance: only few grid cells are part of the path while the vast majority is not. AuROC is known to perform poorly in such conditions [148]. This can also be seen in Table 6.4. Even though Prediction B is far from the correct location, it still has a higher AuROC. This problem is overcome for AuPR. Therefore, only AuPR will be reported in the experiments.

	AuROC [%]	AuPR [%]
Pred. A	0.12	**1.03**
Pred. B	**1.24**	0.20
Combined	0.55	0.69

Table 6.4.: Areas under ROC-Curve and PR-Curve for the example prediction from Fig. 6.10. Higher is better for both metrics, perfect prediction corresponds to 100%.

Chapter 7
Experimental Validation

Previously, a theoretical model for a goal-directed prediction system was proposed. It consists of two individual parts, namely the estimation of possible destinations and a goal-directed prediction towards them. Section 5.1 introduced a recurrent mixture density network (RMDN) for destination prediction. To predict trajectories to these, two different neural networks were presented, one based on the forward-backward (Fwd-Bwd) algorithm in Section 5.2.1 and one based on Markov Decision Processes (MDP) in Section 5.2.2. In this chapter, the proposed methods are validated experimentally. Each of the three aforementioned networks features unique design parameters specific to the respective architecture. In the following, first, the architectures and their parameters are studied in isolation. Finally, the joint network is evaluated in terms of full trajectory prediction performance. For all experiments, the dataset and metrics from Chapter 6 are used. The methods from Section 3.3 serve as a reference. For clarity of presentation, results are only shown visually. Also, details on implementation, generic parameters, and architectures are omitted here. The interested reader is referred to Appendix B for such details. This chapter significantly extends the evaluation published in [133].

7.1. Destination Prediction

The correct inference of destinations is the foundation of planning-based prediction. Thus, the Recurrent Mixture Density Network (RMDN) introduced in Section 5.1 is the first building block to evaluate.

Here, the impact of model parameters as well as different training techniques on performance is investigated. For comparison, the Interacting Multiple Model (IMM) Filter [152] and the DOGMa-CNN [69] are selected from Section 3.3. As presented in Table 3.1, the IMM marks the best mNLP, while for best mPP, the DOGMa-CNN and Activity Forecasting [86] perform comparably. Of these, the DOGMa-CNN is chosen as a representative learned method.

7.1.1. Experiment Setup

In a first set of experiments, architecture-specific parameters and how they impact performance are evaluated. For every experiment, individual parameters are varied while all others remain fixed. For the default parameterization, see Section B.3.1. Note that the following experiments do not employ image features unless stated otherwise.

The training data $\mathcal{D}$ is taken from the dataset presented in Chapter 6. We train on all trajectories, $\zeta \in \mathcal{D}$, where the length of the trajectory N_ζ is longer than the desired prediction horizon T, $N_\zeta > T$. As input $\boldsymbol{x}$ for training, we shift all trajectories by their current state,

$$\boldsymbol{x}_t = \boldsymbol{s}_t - \boldsymbol{s}_{t-1}, \quad \forall t \in 1, \ldots, N_\zeta - T. \tag{7.1}$$

As target outputs $\boldsymbol{y}$, the difference between the current state $\boldsymbol{s}_t$ and the state at prediction horizon T in the future is used,

$$\boldsymbol{y}_t = \boldsymbol{s}_{t+T} - \boldsymbol{s}_t, \quad \forall t \in 1, \ldots, N_\zeta - T. \tag{7.2}$$

7.1.2. Loss

Training of the neural network requires an adequate loss to minimize. The objective is to find the neural network's parameters $\hat{\boldsymbol{w}}^*$ that minimize the loss L over the dataset $\mathcal{D}$,

$$\hat{\boldsymbol{w}}^* = \underset{\hat{\boldsymbol{w}}}{\operatorname{argmin}}\, L(\mathcal{D}, \hat{\boldsymbol{w}}). \tag{7.3}$$

The minimum is found using Adam as iterative numerical solver [85]. For this, the parameter updates are not computed on the full dataset $\mathcal{D}$, but on small samples $\mathcal{B}$ of it, so-called mini batches.

The network described in Section 5.1 predicts a probability distribution of the future states $\hat{\boldsymbol{y}}_i$, given the current input $\boldsymbol{x}_i$ and the network's parameters $\hat{\boldsymbol{w}}$, which is abbreviated as p for sake of legibility,

$$p := p(\hat{\boldsymbol{y}}_i|\boldsymbol{x}_i, \hat{\boldsymbol{w}}). \tag{7.4}$$

To optimize probability distributions, it is customary to use the mean negative log likelihood as a loss function [20],

$$L_{\text{Mean}}(\mathcal{B}, \hat{\boldsymbol{w}}) = -\frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{y}_i, \boldsymbol{x}_i \in \mathcal{B}} \log p. \tag{7.5}$$

However, for prediction, it is desired to pay special attention to rare events, as these can be the most dangerous in traffic. In previous works, it was shown that hard example mining can improve performance [133]. For this, only the worst k samples are selected from every training batch to form a new batch $\mathcal{B}_k$. The loss of Eq. (7.5) is adjusted accordingly so that still the mean negative log likelihood is calculated, but now only for the hardest k samples.

Another way of emphasizing the difficult examples in training is focal loss [98]. In this loss, predictions are reweighted according to how well the network performs on them already. This is done by multiplication with the complementary of the predicted probability,

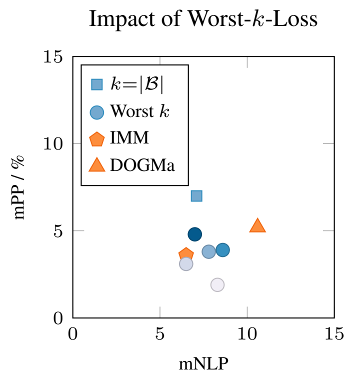
$$L_{\text{Focal}}(\mathcal{B}, \hat{\boldsymbol{w}}, \gamma) = -\frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{y}_i, \boldsymbol{x}_i \in \mathcal{B}} (1-p)^{\gamma} \log p, \tag{7.6}$$

where the exponent γ is a hyper parameter that governs the impact of well-predicted samples on the total loss.

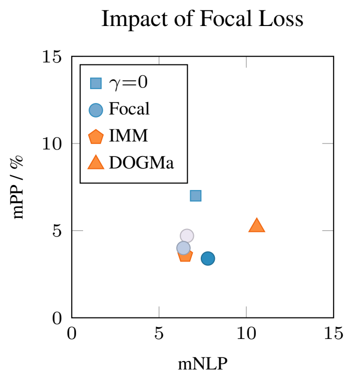
The results in Fig. 7.1 indeed show that focus on hard examples can improve mNLP. However, this comes at the expense of significantly decreased mPP. Consequently, Eq. (7.5) is utilized in all following experiments.

7.1.3. Number of Mixtures

The number of mixture components in the network determines the maximum number of possible motion hypotheses it can predict. Therefore, it is a crucial design parameter. Generally it is expected that more mixture components lead to better performance. If not all are needed to explain the data's variance, they might either collapse during training or be trained as redundant predictors, e.g. through help of dropout.



(a) Worst-k-Loss, from $k{=}1$ (bright) to $k{=}8$ (dark). $k{=}|\mathcal{B}|$ is the loss from Eq. (7.5).



(b) Focal Loss, with $\gamma{\in}\{1,2,4\}$ (bright to dark). $\gamma{=}0$ is the loss from Eq. (7.5).

Figure 7.1.: Prediction performance for networks trained with and without emphasis on hard examples.

At the same time, however, their count increases computational demands, training efforts as well as the risk of overfit. Therefore, the minimum possible number of components should be found that delivers acceptable performance.

To evaluate the number of mixtures needed, mixture density networks are trained with component counts from one to sixteen. Of course, an MDN with just one component is no longer a mixture but simply predicts a single Gaussian. For training of this single Gaussian, no dropout is applied (cf. Section 7.1.4). The performance of all networks is visualized in Fig. 7.2. While mPP is comparable for all experiments, larger variance is seen in mNLP. The network with only a single Gaussian significantly falls behind any MDN, even compared to a mixture with only two components. Adding more components improves mNLP further until it reaches saturation for more than eight. This shows that tracking more hypotheses aids neither exactness nor robustness.

7.1.4. Dropout

A common problem of mixture density networks are vanishing components. When single components of the mixture explain most of the variance, they get preferred in training, which in turn leads to better explanatory capabilities in these. Unfortunately, this hinders the prediction of multiple

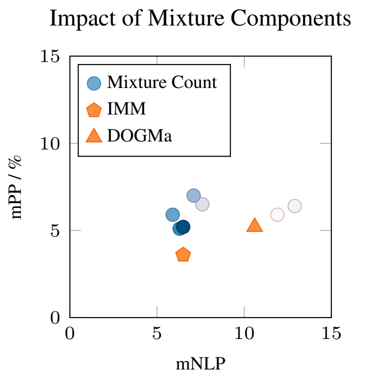


Figure 7.2.: Destination prediction performance for different numbers of mixture components from 1 (bright) to 16 (dark).

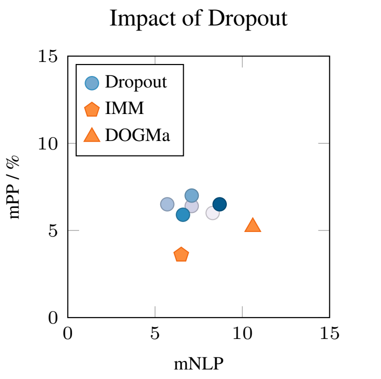


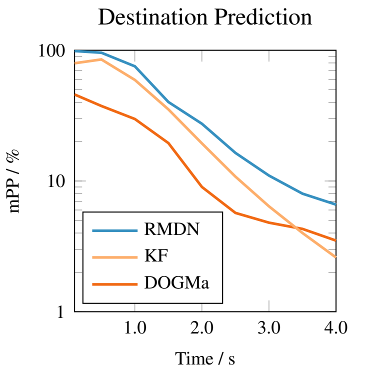
Figure 7.3.: Destination prediction performance for different dropout rates from r=0.0 (bright) to r=0.5 (dark).

modes in the distribution. To overcome this, a portion of the mixture components is dropped randomly in training [65], leading to more diverse updates on the mixture. As another positive effect, the network learns to not rely on individual components too much, but instead, train multiple redundant predictors. The dropout is performed randomly per component with a probability of r.

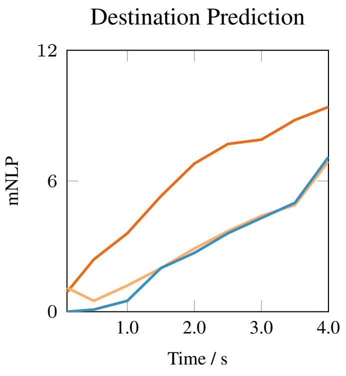
The impact of dropout on prediction is evaluated by varying the dropout rate r. Experiments are run for dropout rates from r=0% to r=50% in steps of 10%. The results are depicted in Fig. 7.3. While the dropout rate has little effect on the mPP, it significantly impacts the mNLP. Both, small and large rates lead to deterioration in performance. This is expected as with no dropout, many components collapse while at large rates, the network cannot develop meaningful individual components. As a good compromise between mPP and mNLP, in the following, r=30% is used.

7.1.5. Prediction Horizon

Performance of destination prediction greatly depends on how far into the future a forecast is desired. For short prediction horizons, motion is dominated mostly by dynamics while decisions and intention only play a minor role. However, arguably, they become the primary cue for prediction for longer lookahead times. To validate this, destination prediction is evaluated for different time horizons.



(a) Predicted probability of destinations. Higher is better.



(b) Neg. Log. Predicted probability of destinations. Lower is better.

Figure 7.4.: Destination prediction performance for different time horizons.

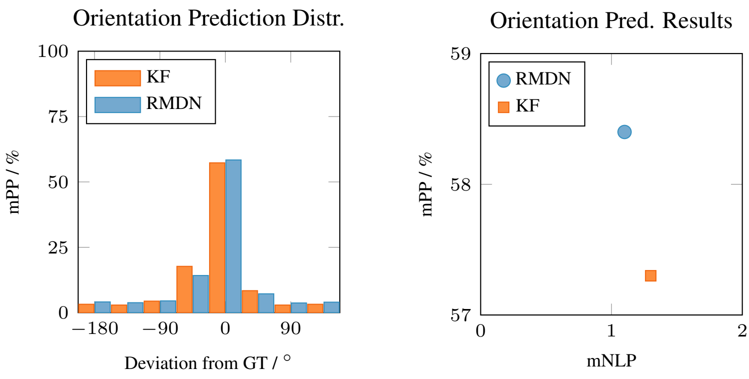
Figure 7.4 shows mPP and mNLP for destination prediction. For reference, KF and DOGMa are included in the evaluation as explained in Section 3.3. As expected, the dynamics-based KF performs well for short prediction times. From roughly 2.0s on, however, the pattern recognition algorithms in form of DOGMa-CNN and RMDN show their potential. As their uncertainty is selected depending on data, they can provide informative predictions even for long prediction horizons.

For mNLP, the covariance estimate of both KF and RMDN gives them a benefit in safety. The DOGMa-CNN on the other hand is trained to predict occupancies of limited extent. If these miss the ground truth, mNLP is greatly increased. As a result, the DOGMa-CNN falls far behind RMDN and KF in safety.

In conclusion, the RMDN delivers the best performance for both measures jointly, i.e. accuracy in terms of mPP as well as safety measured by mNLP.

7.1.6. Orientation Estimation

A pedestrian's heading provides valuable information about future directions of motion. Eq. (5.1) defined the mixture density prediction of the RMDN to also predict a distribution of pedestrian orientations in the destination. After previously having evaluated the destination location prediction, let us now also take a look at heading prediction. In analogy to the



(a) Distribution of orientation prediction centered around GT.

(b) Average orientation prediction performance.

Figure 7.5.: Evaluation of orientation prediction.

prediction grids from Section 6.4.1, the heading is discretized into equally-sized bins for evaluation.

Unfortunately, none of the reference methods from Section 3.3 directly provides heading information. However, the Kalman Filter's velocity estimate can be utilized to obtain comparable predictions. For this, the distribution of velocities is mapped onto a grid, similar to Fig. 4.1.

Then, for every orientation bin, the probabilities of all grid cells in the corresponding angular sector are summed. The result is a categorical distribution of predicted heading angle ranges. For the RMDN, the von-Mises part of the distribution from Eq. (5.3) can directly be evaluated in the individual bins.

In analogy to the experiments for destination prediction, the predicted probability distributions are evaluated relatively to the ground truth orientation. The corresponding histogram is displayed in Fig. 7.5a. The bars show the average probability with which deviations from the ground truth orientation were predicted. Compared to the KF, the RMDN shows a slightly better concentration of predictions around the zero-centered bin with higher values in the tails. As a result, it scores better in both, mPP and mNLP than the KF as shown in Fig. 7.5b.

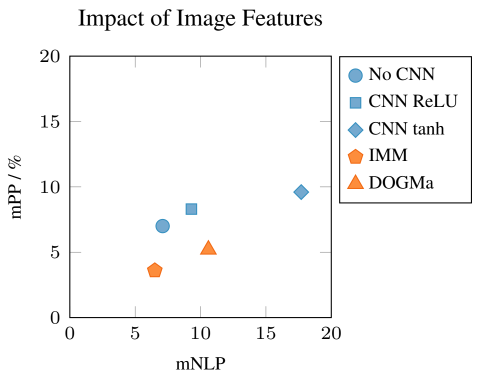


Figure 7.6.: Destination prediction performance with image features as additional input. The CNN architectures only differ in the activation of the final layer.

7.1.7. Visual Features

Pedestrian's intentions are often visible in posture, gait and gaze direction. All these clues can be observed from visual features. In the previous experiments, the only feature used for prediction was the past trajectory.

As explained in Section 5.3.2, the architecture allows for the use of arbitrary input modalities. In the following, the benefit of video input to prediction is evaluated. For this, a CNN architecture inspired by VGG-16 is employed as a feature extractor [158]. The visual feature embeddings are then concatenated with the current position as input to the RMDN. The full architecture can be found in Appendix B.3.1.

Image processing helps to improve the mPP over pure trajectory features as shown in Fig. 7.6. However, this comes at the cost of increased mNLP. The reason lies in the CNN's inclination towards overfit. Measures can be taken to avoid this, e.g. stronger regularization, augmentations or network design towards less overfit. As an example, the impact of the feature activation is demonstrated with ReLU and $\tanh$ functions in Fig. 7.6. Using $\tanh$ results in the highest mPP, while the mNLP is deteriorated. The ReLU, on the other hand, comes at slightly worse mPP while still scoring significantly better in mNLP. These results show that the model is well able to learn meaningful image interpretation. However, due to the comparably small dataset, it is prone to overfit and therefore not integrated in subsequent experiments.

7.1.8. Qualitative Results

An impression of prediction performance can be given through qualitative examples. Fig. 7.7 shows frames of video sequences together with corresponding predictions in different situations. For all examples, results of Kalman Filter and of the RMDN are shown as heat maps.

The first example shows the standard case in traffic: a pedestrian walking straight. We see that the RMDN correctly identifies the situation from the second observation on and outputs narrow predicted probability densities. In contrast to this, the linearly growing covariance of the KF results in predictions that are spread out much more.

The second case shows a person with constant velocity that changes direction in the course of the sequence. Both, the RMDN and KF approximate this behavior in their prediction but underestimate the turn. The KF again accounts for this with a larger uncertainty ellipse in all directions. The RMDN, on the other hand, predicts a rotated covariance that better reflects constant velocity but uncertainty about the heading.

Finally, the third example is that of a person hesitating before eventually crossing the street. Both RMDN and KF cannot predict the correct crossing velocity as the person reaches the curb. However, since the RMDN can adapt to the situation, it outputs two modes, one for smaller and one for larger speed of the pedestrian. As time passes, the modes move towards the true velocity but retain a higher uncertainty than the KF. These results demonstrate the benefit of a data-driven estimate of uncertainty as compared to the rigid model of the KF.

7.2. Goal-Directed Prediction

If a persons destination is known, prediction breaks down to inferring the way towards it. Chapter 4 introduced neural networks for this task in form of a Forward-Backward-Net (Fwd-Bwd) as well as one for Markov Decision Processes (MDP). First, the potential of the two planning-based prediction techniques is studied in isolation. Thus, initially, ground truth destinations are provided to identify optimal parameters for both architectures as well as their training.

In the experiments, the focus lies on those parameters that are unique to goal-directed architectures. Firstly, the regularization techniques from Section 5.2.3 will be studied, as standard weight decay does not apply (cf. Section 2.2.4). Since the planning-based algorithms infer a series of actions, they are evaluated in detail. Finally, the influence of different environment traits on prediction is analyzed.

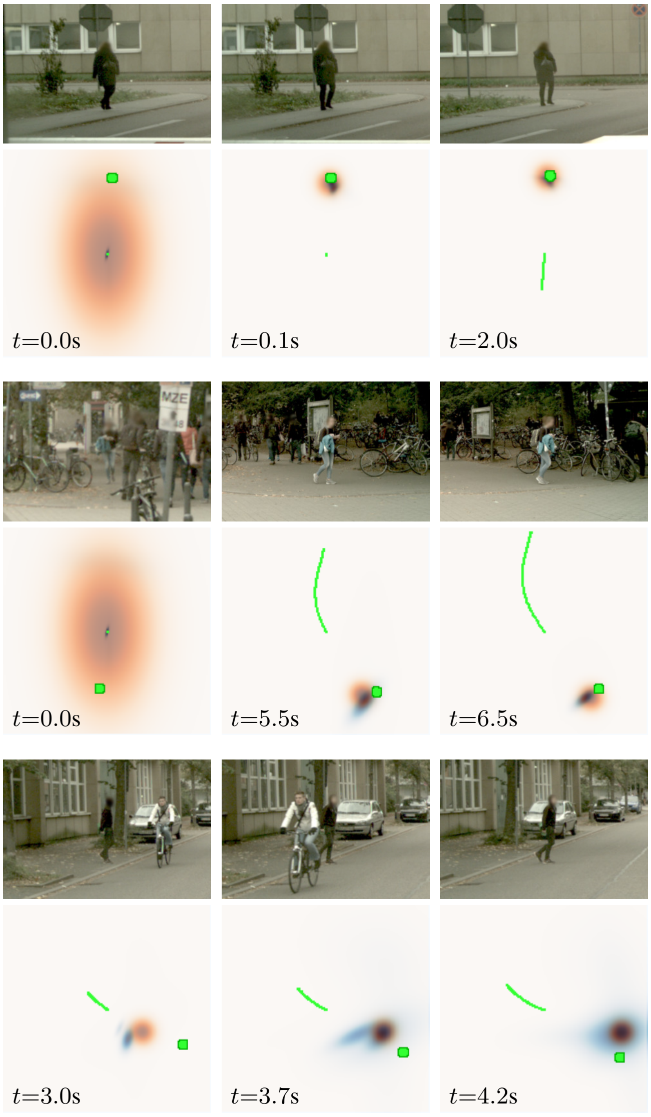
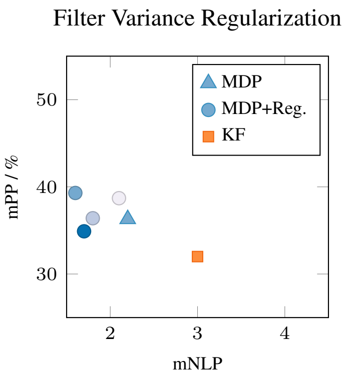
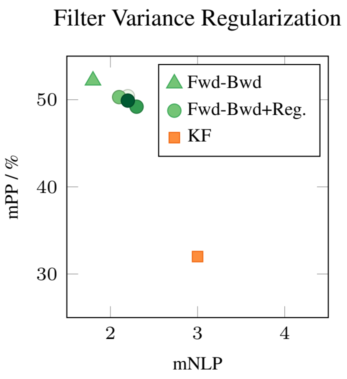


Figure 7.7.: Qualitative examples of destination prediction with RMDN (blue), Kalman Filter [19] (orange). Past trajectories shown as green lines, ground truth destinations as green circles. All top views are 16m×16m in size. The displayed times show to the time of observation, all predictions are 4.0s into the future.



(a) Impact of filter variance regularization on MDP performance.



(b) Impact of filter variance regularization on Fwd-Bwd performance.

Figure 7.8.: Impact of filter variance regularization with different weighting from 10^{-8} (dark) to 10^{-4} (light). The higher the regularization, the more condensed filters are preferred. While regularization helps the MDP, it harms Fwd-Bwd performance.

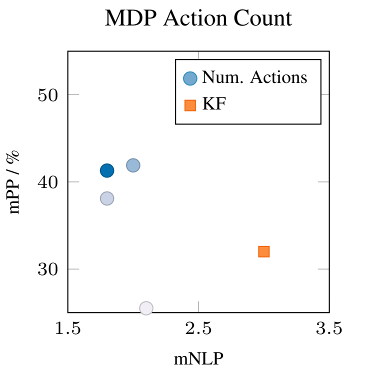
As it is the best reference method from Table 3.1, the simple Kalman Filter [19] is selected for all upcoming experiments.

7.2.1. Regularizations

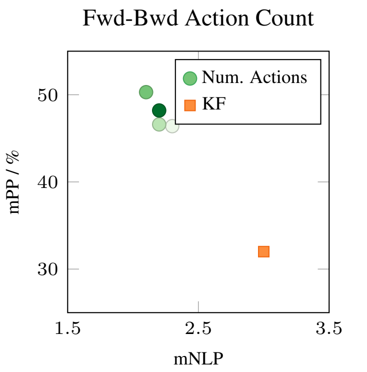
Regularization techniques are a key ingredient for successful neural network training. Depending on the network's application, these regularizations are tailored to avoid overfitting and stabilize training [56]. For standard CNN architectures, weight decay and dropout have proven effective [65, 90]. However, they do not translate well to decision processes.

Instead, as explained in Section 5.2.3, regularizations are designed specifically to improve learning of action patterns. Therefore, it was proposed to minimize the filter variance Eq. (5.20) as regularization to penalize multiple motion patterns within a single action. The variance is added to the full training loss weighted by the factor λ_{var}.

Figure 7.8 shows how the performance changes with different regularization weights λ_{var}. While moderate regularization helps improve performance for the MDP (left, light circles), predictions degrades for higher weights (dark circles). In these cases, the regularization becomes too strict of a constraint . In contrast to that, regularization does not improve Fwd-Bwd performance as the method inherently is goal-driven and does not benefit from additional guidance.



(a) Impact of action count on MDP performance.



(b) Impact of action count on Fwd-Bwd performance.

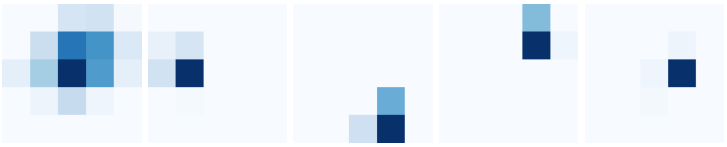
Figure 7.9.: Prediction performance depending on number of individual actions from 5 (light) to 21 (dark). For both methods, the benefit of more action saturates at 13.

7.2.2. Number of Actions

Planning-based prediction infers actions that an agent is most likely to take in the future. In the network architectures, these actions are reflected by trainable convolutional filter kernels. Each kernel represents a probability distribution of transitioning between grid cells. Consequently, the number of these kernels is a design parameter to be specified prior to training. Intuitively, it defines the set of possible decisions for which an individual probability distribution is learned.

The number of actions is varied to evaluate its impact on prediction performance. Since it is expected that the actions correspond to different motion vectors, their number is selected to reflect major directions and velocities. With five actions, the space can be divided into the center point and four cardinal directions. Adding four more allows for diagonal motion. Consequently, every additional step in the hyperparameter analysis would allow for more precise classification of motion patterns.

The results of the evaluation of both, MDP- and Fwd-Bwd-Net are shown in Fig. 7.9. As expected, we see that adding more possible actions improves performance for both networks. While this effect saturates for the MDP-Network (left, blue), the performance of the Fwd-Bwd-Network degrades from more than thirteen actions on (right, green). Note that, similar to the regularization results, even the worst Fwd-Bwd prediction outperforms the best MDP predictions.


(a) State transition filters learned by MDP-Network

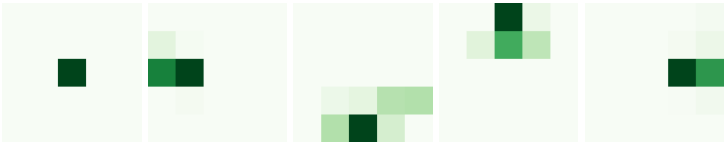

(b) State transition filters learned by Fwd-Bwd-Network

Figure 7.10.: Action transition filters learned in actual prediction networks. Every image corresponds to a different action where the pixels' brightness values represent the probability of transition from the middle to a neighboring state. Each filter learned a major direction of motion as well as the uncertainty about it. Note that these filters were not predefined but emerged automatically in training.

The validity of learned transition models can be verified when looking at the trained filters in Fig. 7.10. Each filter represents a 0.5m$\times$0.5m grid of 0.1m resolution around the pedestrian's current position. The values of the cells reflect the probability where a pedestrian will end up when executing the corresponding action. The results impressively demonstrate the success of the training: every filter covers a part of the neighborhood and, thus, reflects a major direction of motion.

7.2.3. Map Features

The environment plays a vital role in behavior of traffic participants. Our planning-based prediction therefore is designed to reason about future trajectories also based on the layout of the environment (cf. Section 4). As a representation, grid maps are used that are converted into topology maps for a planning network, as shown in Fig. 5.9. The input to the topology network is given as the layers of the multi-layer grid mapping from Section 6.1.5.

Different features have impact on pedestrian motion. From intuition, we can identify the locations of obstacles, road, sidewalk, and curb as relevant traits of a scene. While these can be extracted from stereo vision and semantic segmentation, this information can also be extracted directly from the bird's eye view (BEV), which is another layer in the grid map.

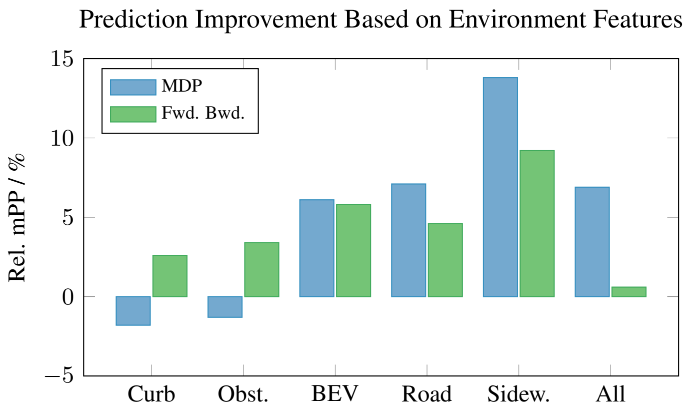


Figure 7.11.: Improvement of prediction results based on different environment features. All results are relative to the performance achieved when no information on the environment is used for prediction.

With the features at hand, it is of interest which of these is the most beneficial for the prediction architecture. In order to analyze the impact of the individual features, train both, the MDP- and the Fwd-Bwd-Net, are trained with only single feature maps as input. All trainings, however, have access to the current pedestrian's position as well as destination and distances to both since these can be obtained independent of the environment grid map.

The benefit of different features is compared against prediction networks that had no access to the environment grid. For this, the prediction was trained solely with start and destination information as input. This allows to assess the impact of features compared to a common baseline. The relative change in prediction performance is displayed in Fig. 7.11. Since in the previous experiments the mNLP only varied slightly, now, mPP is reported exclusively. Also note that the zero line in Fig. 7.11 reflects different starting points for MDP and Fwd-Bwd as the latter performs better even in the environment-agnostic case.

The benefit of scene layout information depends on both, the prediction algorithm as well as the kind and quality of features. Firstly, both MDP and Fwd-Bwd can improve with knowledge about individual environment traits. The least improvement (or even degradation for MDP) is observed for curb and obstacle features. This is due to the fact that these two features are the most noisy in mapping and, therefore, may even perturb the network. Instead, the more stable road, sidewalk and BEV features improve

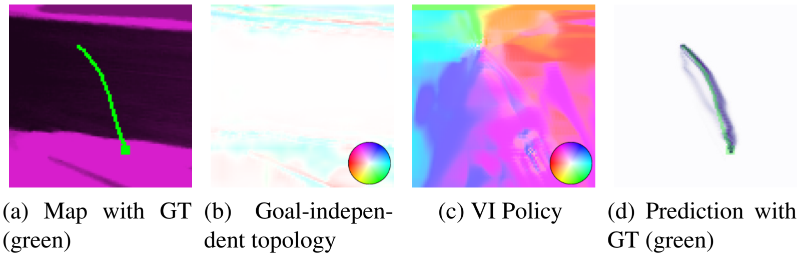

(a) Map with GT (green) (b) Goal-independent topology (c) VI Policy (d) Prediction with GT (green)

Figure 7.12.: Planning in MDP. The colors of topology and policy display the predominant predicted direction of motion, indicated by the color wheel. White areas mean all orientations are favored equally. When no destination is presented to the network, only the tendency towards walking alongside the sidewalk is predicted (b). Given a pedestrian's destination, a policy is generated where all directions point towards it (c). The resulting prediction correctly models the curvature of the trajectory (d). The green box indicates the start of the trajectory.

prediction the most. Interestingly, the Fwd-Bwd-Network cannot benefit from access to all features as much. This can, again, be attributed to the network's limited learning capacity in the presence of noisy observations.

In order to better understand how features contribute to the policy in the MDP, the process is visualized in Fig. 7.12. The MDP's action cost topology is computed via an FCN (cf. Appendix B.4.1) for a map of sidewalk features. This map is shown in Fig. 7.12a together with the ground truth path of a pedestrian crossing the street.

If all trajectory-related features are omitted, i.e. start and end position and the respective distances, common motion directions independent of the current pedestrian can be retrieved as displayed in Fig. 7.12b. The map only imposes a slight bias on preferred directions, like walking *along* the sidewalk, while for most regions exhibit no distinct directional bias. If now, pedestrian features are incorporated and Value Iteration is run, the policy shown in Fig. 7.12c is obtained. As this policy introduces the goal-directed nature in the MDP, it now shows clear directed motion towards the trajectory's endpoint. Also, it incorporates slight nuances on motion patterns depending on local features of the map. Therefore, the final prediction in Fig. 7.12d not only shows the actual curved motion, but also accounts for the possibility of a more straight path towards the destination.

The selection of actions in the Fwd-Bwd-Network is fundamentally different to the policy of the MDP. The same situation as before is visualized in Fig. 7.13. In the Fwd-Bwd-Net, the drive towards the goal does not come from the action topology itself, but from the backward pass.

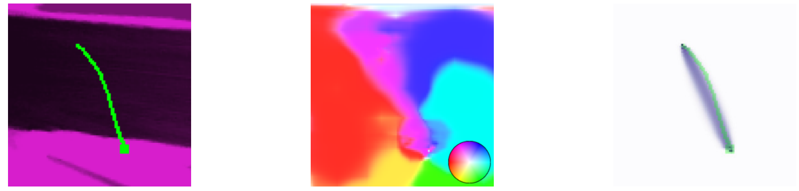

(a) Map with GT (green)
(b) Goal-directed topology
(c) Prediction with GT (green)

Figure 7.13.: Planning map in Forward-Backward Network. Again, colors display the predominant motion direction, indicated by the color wheel. All motion leads away from the starting point of the trajectory (green box), but includes local variations. The drive to the goal is induced through the backward pass.

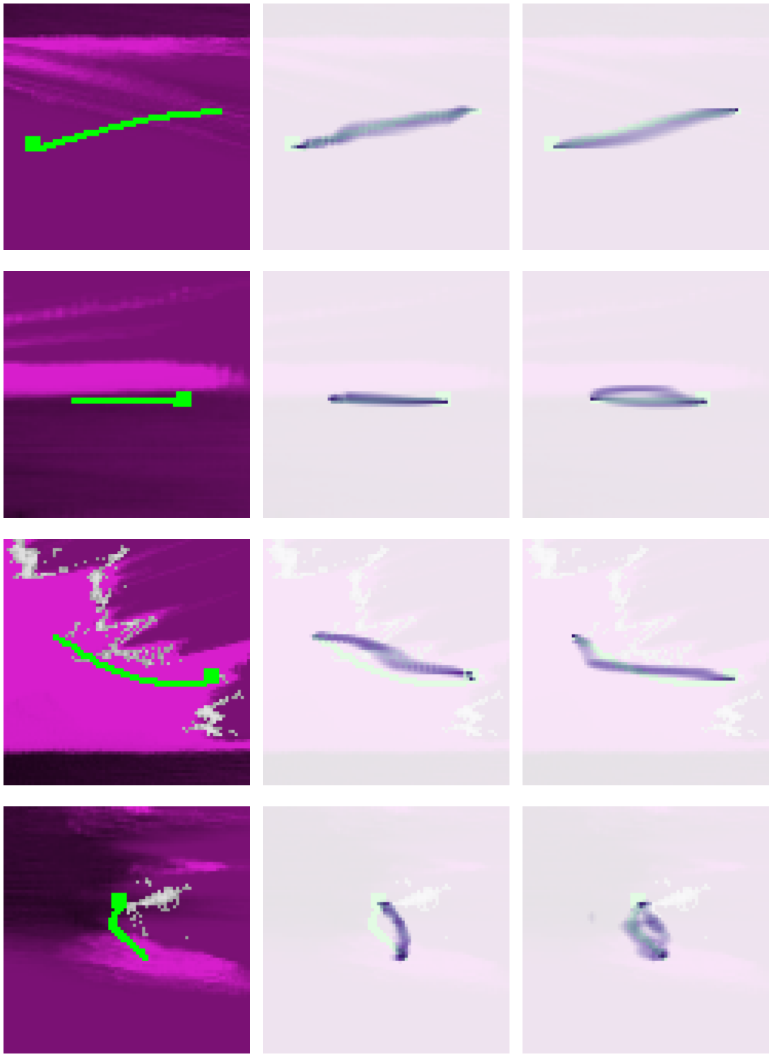
Therefore, the topology does not have to incorporate any goal information, but instead only reflects variations in propagation away from the start of the trajectory as shown Fig. 7.13b. In contrast to the discrete decisions of the MDP, the Fwd-Bwd-Net embeds the uncertainty about the future into a wider distribution in prediction Fig. 7.13c.

7.2.4. Qualitative Results

The implications of planning-based prediction can be studied from illustrations of trajectory forecasts. Destinations were given as ground truth. For clarity of presentation, paths are shown rather than trajectories in Fig. 7.14. The left column displays the feature map (cf. Section 6.1.5). The middle column shows predictions generated with the MDP-Net, the right column those of the Fwd-Bwd-Net. All are overlayed with the ground truth path (green), the starting point marked with a small square.

In the first row, we see the prediction of a trajectory with laterally displaced goal. While the MDP predicts a straight connection together with larger uncertainty in the middle, the Fwd-Bwd-Net outputs a sigmoidal transition. Both models, however, account for the actual trajectory within their uncertainty.

The trajectory of the second row stems from a pedestrian that walked on the road, yet close to the sidewalk. As the destination is known, the MDP predicts the direct route from start to end, staying on the road at all times. The Fwd-Bwd-Net, however, shows two possible outcomes, namely the direct connection as well as one that pays tribute to pedestrians' preference of walking on the sidewalk.



(a) Semantic maps. (b) MDP predictions. (c) Fwd-Bwd predictions.

Figure 7.14.: Qualitative examples of predictions given known destinations (blue) together with the ground truth path (green). All maps are 8m×8m of size. The green box marks the start of the trajectory. For their forecasts, environment knowledge from semantic maps is incorporated both, the MDP- and Fwd-Bwd-Network. This allows reasoning such as that static obstacles should be avoided or sidewalk is preferred.

The final two rows show interesting cases in which obstacles obstructed the pedestrians' direct path to the destination. In both cases, the MDP decides for only one of the homotopic classes of trajectories. In contrast, the Fwd-Bwd-Net outputs the correct option in the third row and predicts two modes in the fourth. These examples demonstrate overconfidence as a shortcoming of the MDP.

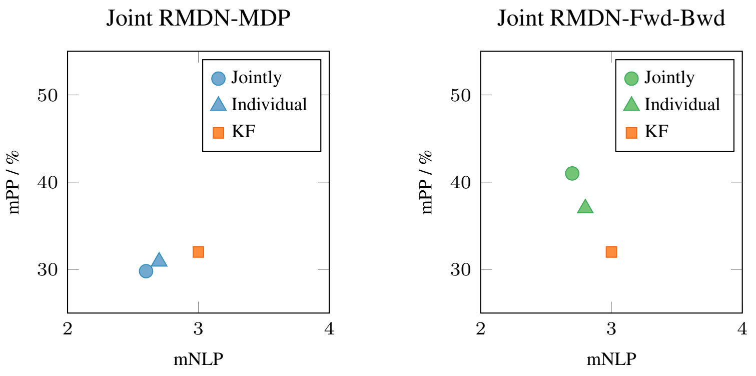
7.3. Joint Goal and Trajectory Prediction

The goal-directed prediction system infers goals and the path to them at the same time. For both, neural networks are employed. So far, only those two parts were evaluated in isolation. It was shown that accurate destination prediction can be achieved from observations through an RMDN. Furthermore, both planning algorithms, the MDP- and the Fwd-Bwd-Network, performed well in predicting reasonable trajectories towards a given destination. The ultimate goal, however, is to integrate the two networks into one so that the RMDN predicts the goal as a latent variable for the planning-based prediction systems.

The two stages, the destination prediction and planning are conjoined in a single, monolithic network (cf. Fig. 5.9). As they work together to generate predictions, they should also be trained as such. The modeling of the two tasks as fully differentiable networks allows for gradient flow between them. Therefore, the destination prediction can be trained to adopt to the needs of the subsequent goal-directed prediction network.

In the previous sections, the optimal design parameters for the individual components were identified. When destination prediction and planning networks are trained jointly, these findings are used and only the respective best networks are evaluated together. Road, sidewalk and obstacle maps serve as features of the environment.

Figure 7.15 shows the evaluation of the full goal-directed prediction for both of the presented planning networks. To verify the benefit of joint training, the two stages were also evaluated trained together, but with the gradient flow from trajectory to destination prediction prohibited. This way, the planning stage may adopt to the predictions of the goals but cannot pass on corrective information to the RMDN. As can be seen, both, the MDP- and Fwd-Bwd-Net's mPP performance drop by roughly ten percentage points as compared to the case with known goal. While the Fwd-Bwd-Net still outperforms the KF reference by far, this no longer is the case for the MDP. It now performs comparably to the KF in mPP, yet slightly better in mNLP. Also, the networks with backpropagation to destination prediction enabled are compared to those without it. As the MDP's goal-direction results from



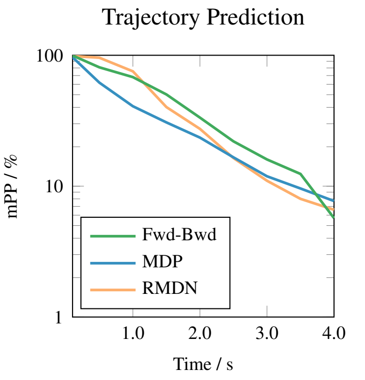
(a) Joint RMDN-MDP results.

(b) Joint RMDN-Fwd-Bwd results.

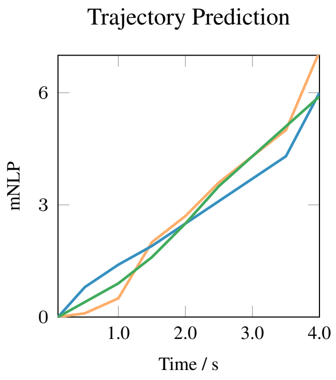
Figure 7.15.: Results of joint destination prediction and planning networks. The networks are either trained individually without gradient flow between them, or jointly with gradient flow from the planning to destination prediction stage.

the cost map rather than the actual goal localization, its general performance does not change significantly. The Fwd-Bwd-Network, on the other hand, improves in both, mPP and mNLP.

Prediction of trajectories also involves a time component. Therefore, we evaluate the prediction for every time step up to the time to goal T. Section 7.1.5 showed that the RMDN can outperform the reference methods for destination prediction for various lookahead times. Naturally, these intermediate destinations could be treated as way points along a trajectory. Therefore, Fig. 7.16 shows the corresponding results compared to the RMDN instead of the reference models from Chapter 3.3. As in Fig. 7.15, the MDP-Net performs worst of the three in mPP. It comes out on top only at the long range of the prediction horizon. This means that on the one hand, goal localization has improved as compared to the RMDN alone. However, the underlying planning algorithm cannot benefit from this since its prediction is less accurate in the early time steps. The Fwd-Bwd-Net, however, performs best over most of the prediction range. Only for the final destination, the mPP drops. This suggests that a slightly widened destination estimate is beneficial for trajectory prediction. Finally, it may be noted that in terms of mNLP, the planning methods perform same or better than the RMDN except for very short prediction horizons. These results show that planning-based prediction can outperform pure prediction of way points as it reasons about decisions, also with regards to properties of the environment.



(a) Predicted probability of destinations. Higher is better.



(b) Neg. Log. Predicted probability of destinations. Lower is better.

Figure 7.16.: Trajectory prediction performance evaluated for all time steps within prediction horizon.

Finally, we look at the absolute numbers in comparison to the reference methods from Section 3.3 shown in Table 7.1. As expected from the previous results, the reference algorithms are outperformed in all evaluated metrics by at least one of the goal-directed prediction methods. While the MDP-Net with RMDN performs best for destination forecasting, the Fwd-Bwd counterpart ranks top for trajectory prediction. Both, however, are well suited to correctly predict a pedestrian's path. It should be noted that the proposed methods perform competitively in all evaluated metrics *simultaneously*. As a dynamics-based, yet learned model of the underlying decision process, the proposed model can combine the strengths of all reference algorithms without inheriting their weaknesses.

Finally, the combination of destination prediction and goal-directed planning are demonstrated with qualitative examples. In analogy to Section 7.2.4, path forecasts are displayed that were generated from inferred destinations in Fig. 7.17. The left column again shows the input maps together with the ground truth path, the middle and right columns show the corresponding results from MDP and Fwd-Bwd, respectively. Compared to Fig. 7.14, we study the same situations except for the straight trajectory as it is trivial and therefore less interesting. All examples in Fig. 7.17 stem from curved trajectories to showcase the strength of planning-based prediction.

Method	Trajectory		Path	Destination	
	mNLP	mPP/%	AuPR	mNLP	mPP/%
Kalman Filter [19]	3.0	32.0	43.8	6.9	2.7
IMM [152]	3.0	27.2	45.0	6.5	3.6
DOGMa-CNN [69]	6.0	16.2	28.6	10.6	5.2
Activity Forec. [86]	9.8	9.1	12.0	10.4	5.4
RMDN + MDP	**2.6**	29.8	**53.6**	5.9	**8.2**
RMDN + Fwd-Bwd	2.7	**41.0**	52.6	**5.8**	5.3

Table 7.1.: Comparison of goal-directed prediction networks with reference methods. The proposed method performs competitively in all metrics simultaneously.

In the first two rows, we see cases in which the ground truth deviates from the expectation of both algorithms but is still contained within the predicted distributions. While the MDP's estimate covers this case with broader destination and path uncertainty, the Fwd-Bwd realization addresses this with multiple goal hypotheses and, thus, trajectories.

The benefit of map knowledge is indicated in the second and third row. While barely visible, the planning methods take obstacles into account and predict the majority of the probability mass around them. However, this effect is not extremely pronounced. The reason lies in time-varying occupancy in the dataset. The network has learned that, while obstacles are mostly avoided, the possibility exists that these are actually moving objects and, therefore, could be traversed at a future time. This effect could be mitigated if occupancy was classified as static or dynamic, e.g. through filtering [120] or semantic information.

Finally, the last row shows an especially challenging case in which the pedestrian's state of motion changes from standing to walking. At this time, the network has just picked up the hint of acceleration. Consequently, the predicted destinations are still close to the current position and, therefore, the predicted path too short. However, the direction and impact of the environment is anticipated correctly.

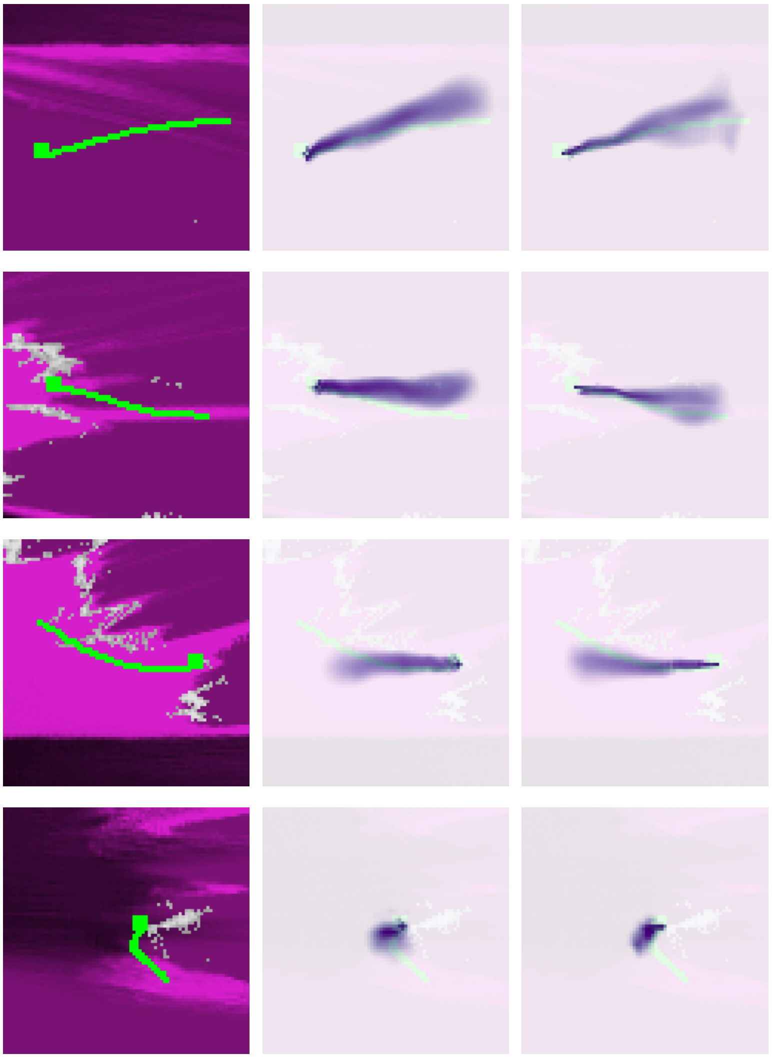

(a) Semantic maps. (b) MDP predictions. (c) Fwd-Bwd predictions.

Figure 7.17.: Qualitative examples of predictions including destination inference (blue) together with ground truth trajectories (green). The green box marks the start of the trajectory. All maps are 8m×8m of size.

Chapter 8

Conclusion

For an autonomous vehicle sharing the space with humans, it is crucial to predict their motion to plan its own actions. While humans perform predictions of even highly complex situations seemingly effortlessly, it is still challenging for machines. In this context, the work at hand aimed to contribute towards cautious yet precise prediction of pedestrians in traffic.

8.1. Summary

Whenever a human navigates through traffic, it is to fulfill some intention. Most commonly, this intention is to reach a specific destination. In the course of doing so, a human's motion is affected by internal and external factors. It is constrained by both, limits of dynamics and scene geometry. At the same time, collisions with other traffic participants and infrastructure should be avoided. Furthermore, other factors such as traffic rules and habits influence a person's behavior. This great variety of factors makes human motion tremendously hard to model explicitly. Moreover, it can never be predicted with absolute certainty. Instead, it has to be approached in a probabilistic manner.

Previous works have only considered some of the aspects of prediction mentioned above. Early works understand the problem as mere extrapolation of observed dynamics [152]. This, however, neglects the impact of both, intentions and the world around it. For this, planning-based techniques have been proposed [86, 182]. These reason about destinations from how well observations align with preassumptions.

Unfortunately, these cannot be used in dynamically changing environments such as traffic where the goal depends on both, the situation and the person in question. Due to the complex nature of the problem, prediction has recently been addressed with end-to-end learning [69, 147]. Albeit the most promising, these methods discard all prior knowledge on the underlying processes. Instead, the hope is to automatically deduce them from data. Therefore, even simple relationships, e.g. dynamic limits, need to be learned by increasingly complex implicit models.

In this context, a solution was proposed that pays respect to all paradigms. Consequently, prediction is modeled as a learnable probabilistic goal-directed planning problem. In this sense, a pedestrian's intention is modeled as a set of possible destinations that are inferred from observations. These destinations are treated as a latent variable for a goal-directed planning stage. Consequently, prediction is interpreted as planning possible trajectories towards the latent destinations. Two separate methods were prosed in this thesis, both of which originate in probabilistic modeling of the physical process of motion. As it still remains infeasible to explicitly parameterize, the entire model is realized as an artificial neural network that can be trained end-to-end. As a result, the proposed method can incorporate features such as video images of the pedestrians or top view maps of the environment to produce the prediction. The model thus combines the benefits of intention-based reasoning with a physical model in a fully learned system.

The inference of destinations is the prerequisite of goal-directed prediction. Since the destination is used as a latent variable, it needs to be represented in form of a probability distribution. To this end, the parameters of a mixture density function are estimated with a neural network. By utilizing a mixture density function, multi-modalities such as different intentions can be captured by the model. As the destination should be predicted from a time series of observations, a recurrent network is employed as a backbone architecture. As input, sequences of positions and visual features are used. From these, the model accurately infers a pedestrian's most likely destination from observations.

Given the destinations, prediction is interpreted as a probabilistic planning problem. Two different approaches are proposed, namely the use of the forward-backward (Fwd-Bwd) algorithm and of Markov Decision Processes (MDPs). In Fwd-Bwd, the model computes the joint probability of traveling from start to a specific location and from that to the destination. In contrast, the MDP aims to find actions that maximize an expected reward, e.g. by reaching the destination. Both methods are formulated as recurrent convolutional neural networks. Through this, the respective networks are fully differentiable w.r.t. motion models and the topology the planning operates on.

Therefore, they can be learned to imitate observed human behavior and, thus, serve as prediction model. Finally, as both, the goal-directed planning and the goal inference are modeled as neural networks, they can easily be combined and optimized jointly.

For optimization and evaluation, a dataset of real-world driving situations was created. As sensor input, only a stereo camera setup was used. Using depth imaging together with semantic segmentation, semantic top view maps of the environment were created. These are then used as input for the planning method. All pedestrians in the scenes were obtained from manual annotation. Finally, in order to meaningfully evaluate probabilistic prediction, the use of two metrics was proposed. The mean predicted probability of a pedestrian's future position reflects how precise pedestrian motion can be anticipated. The mean negative logarithm of it, on the other hand, pays more respect to coverage than precision. Both measures are used jointly to assess algorithm performance.

Using the proposed dataset and metrics, the proposed approaches were analyzed in detail. First, the impact of design parameters unique to the proposed architectures were studied. Finally, the full goal-directed prediction was evaluated with representative methods from literature as reference. The results show that the proposed approach outperforms the references in all evaluated metrics.

Both, the MDP and Fwd-Bwd networks have their unique advantages. While both methods perform comparably in safety, the MDP is more accurate in the destination forecast and path layouts. The Fwd-Bwd network, on the other hand, produces more precise trajectory predictions.

Verified by the evaluations, the proposed method combines the strengths of a physics-based probabilistic model with those of a learning system without inheriting their weaknesses. By integration of the destination as a latent variable, a pedestrian's intention is detected implicitly. Instead of relying on pure dynamics, the subsequent planning architecture successfully mimics the decision process underlying human motion. These two components allow to incorporate a great variety of features of both, the pedestrian in question and the surrounding. In contrast to other learned end-to-end prediction systems, ours explicitly models the physics of motion together with their uncertainty. Consequently, it results in dynamically feasible as well as explainable predictions that pay respect to the environment's constraints and the pedestrian's intention.

8.2. Outlook

In vehicle automation, prediction is no end in itself. Instead, it forms the link between the pure perception, i.e. what currently can be observed, and the planning of future actions. When the behavior of others can be foreseen, an automated vehicle can plan accordingly. Thus, it helps prevent collisions and enable safer, more comfortable traffic flow.

The prediction system proposed in this work contributes towards this goal. It is particularly suited for long-term forecasts of multiple seconds ahead. With its accurate yet safe predictions, it can provide valuable information for anticipatory motion planning. As a result, it can find its applications in systems such as advanced driver assistance, automated vehicles, or even robotics.

In the past, the handling of rare events posed a severe challenge for prediction algorithms. However, as the proposed method learns from observed situations, this can be addressed with a greater database. For the work at hand, only a comparably small dataset was created by manual annotation. As an alternative, datasets could be created automatically, leveraging today's impressive object detection performance [164]. Such automatic data mining can quickly generate large amounts of data [30]. With enough samples, even the noise introduced through inexact perception could be tolerated as long as no consistent bias exists.

When training data can be generated automatically, a learning system could even improve while in operation. If the proposed method was deployed in an automated vehicle, it could compare the actual course of a trajectory with its own predictions in hindsight. This comparison can be used as a lifelong training supervision to ever improve the system. This process could greatly reduce the effort in preparation and deployment of such a system and allow it to adopt to new conditions. In fact, as shown in the experiments, not even a semantic understanding of a scene is necessary as meaningful predictions can already be generated purely from a synthetic bird's eye view. Even the required stereo camera setup can learn its calibration *in situ* [128]. Thus, the full stereo camera-based prediction system could be implemented with no prior knowledge included other than a detection and tracking algorithm. Instead, the proposed algorithm would develop itself in deployment.

Appendix A

Artificial Neural Networks

In this work, artificial neural networks were trained to perform pedestrian prediction. While the foundations of these were presented in Section 2.2, the in-depth details were omitted there for sake of conciseness. In the following, the key operations are explained together with reference network architectures that form the blueprint of the presented implementations.

A.1. Layers

Artificial neural networks are composed of atomic building blocks, so-called layers. The following section will give a brief overview of the most relevant layers.

A.1.1. Full Connection

In a fully connected layer, every output neuron computes a weighted sum of all input neurons and adds a bias [21]. Mathematically, this is realized as a multiplication of an N-dimensional input vector $\boldsymbol{x}$ with an $M \times N$-dimensional weight matrix $\hat{\boldsymbol{W}}$ plus an M-dimensional bias vector $\hat{\boldsymbol{b}}$, resulting in an M-dimensional output vector $\hat{\boldsymbol{y}}$ to which a final activation function $f(\circ)$ is applied

$$\hat{\boldsymbol{y}} = f(\hat{\boldsymbol{W}}\boldsymbol{x} + \hat{\boldsymbol{b}}). \tag{A.1}$$

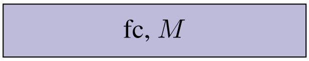


Figure A.1.: Visualization used to represent a fully connected layer of output neuron count M.

In this work, this is visualized as a purple box, possibly including naming and output neuron count as shown in Fig. A.1. The input neuron count is given by the input layer and, thus, not shown.

A.1.2. Convolutional Layer

A convolutional layer operates on rasterized inputs, e.g. images. It uses convolutions with weight kernels to compute the activations of each neuron [94]. Every neuron corresponds to both, a spatial position and a convolution kernel. Given an input image or feature map $\boldsymbol{X}$ with channel count C and a set of N kernels $\hat{\boldsymbol{W}}_c, c \in 1, .., N$ of size $W \times H \times C$, the output neuron at position (u, v, c) in the image is computed as

$$\hat{\boldsymbol{Y}}(u, v, c) = f\left(\sum_{\tilde{u}=1}^{W} \sum_{\tilde{v}=1}^{H} \sum_{\tilde{c}=1}^{C} \boldsymbol{X}(u' + \tilde{u}, v' + \tilde{v}, \tilde{c}) \hat{\boldsymbol{W}}_c(\tilde{u}, \tilde{v}, \tilde{c})\right). \quad \text{(A.2)}$$

Here, $u' = u - \frac{w-1}{2}$ and $v' = v - \frac{h-1}{2}$ are used to center the filter at (u, v) and $f(\circ)$ again is a non-linearity. Convolutional layers are visualized as a blue box with filter size $W \times H$ and output channel count N. Possibly, these filters are not computed for every pixel but in a so-called *strided* fashion, where only every k-th location is evaluated in both, u- and v-directions. The graphical representation is shown in Fig. A.2.

Figure A.2.: Visualization used to represent a convolutional layer with kernel size $W \times H$, N output channels and stride k.

A.1.3. Pooling

Generally speaking, pooling operations are meant to reduce the number of neurons that convey relevant information [90, 95]. In most cases, this is also used to realize invariance towards configurations in feature space, e.g. exact localization of parts of an object.

One example is *max-pooling* where a feature map is partitioned into $M \times M$-sized blocks per channel of neurons. Of these, only the maximum feature response is retained. Equivalent to strides in convolutions, these blocks are evaluated centered around every k-th pixel in both u- and v-directions. As a result, the number of neurons decreases by a factor of k^2. The channel count remains constant in this example. The layer is visualized as a red box with size of the pooling blocks and their stride, shown in Fig. A.3.

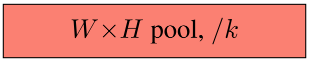


Figure A.3.: Visualization used to represent a pooling layer with kernel size $W \times H$ and stride k.

A.1.4. Transposed Convolutional Layer

Transposed convolutions (sometimes called *inverse convolutions* or *deconvolutions*) are the opposite of convolution operations. As realization, these layers flip the forward and backward pass of a convolution layer. If they are used in a strided fashion, the resulting image resolution is increased by a factor of the stride. As a special usecase, they can be used to compute bilinear upsampling through suitable kernel values [101]. In this work, we use them either as such or as trainable transposed convolutions with kernel size equal to the stride. The symbol is a green box that displays either of the two cases and the stride as shown in Fig. A.4.

Upsample, $\times k$

Figure A.4.: Visualization used to represent a transposed convolution layer used for upsampling by a factor of k.

A.1.5. Local Response Normalization

For various reasons, input features may vary in magnitude. To compensate for this, neuron activations can be normalized in a local neighborhood [90]. Local Response Normalization (LRN) is a technique where neuron activations are normalized according to

$$y_i = \frac{x_i}{(k + \alpha \sum_j x_j^2)^\beta} \tag{A.3}$$

where $k{>}0, \alpha{>}0, \beta{>}0$ and the sum over j includes neighbors of the current neuron, either in spatial or in channel direction. In all experiments, a neighborhood of ± 5 is used in channel direction, $k{=}1$, $\alpha = 1$ and $\beta{=}0.5$. As it is rarely used, LRN is depicted as a function layer as in Fig. A.7.

A.1.6. Group Normalization

Another technique to limit neuron activations is the so-called Group Normalization [177]. For this particular scheme, mean and variance of a layer's activations are computed for the entire layer, but in *groups* of channels. Then, all neuron activations within a group are shifted by their mean and divided by their variance. This way, the activation's statistics can be used even for single inputs, as opposed to batch normalization that requires multiple examples [177]. Due to their rare use and direct attachment to convolution operations, group normalization are not explicitly visualized but stated where it was used.

A.1.7. Others

Some layers are straight forward in their operations but should still be mentioned. To combine two inputs, element-wise operations are used. These might be addition, division, etc., shown as orange boxes labeled with their respective operation as shown in Fig. A.5.

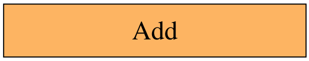


Figure A.5.: Visualization used to represent an element-wise operation of two inputs, here addition as an example.

Joining multiple inputs into one can be done by concatenation. In such layers, the data of all inputs is appended in one dimension. This is reflected as an aquamarine box as shown in Fig. A.6.

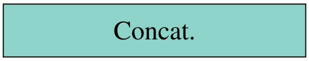


Figure A.6.: Visualization used to represent concatenation of inputs.

Finally, other layers exist that are specialized in their design and rarely used. These are symbolized with a light green box that states the type of layer and may even represent a combination of layers for clarity of presentation. The box displays the represented function as depicted in Fig. A.7.

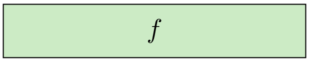


Figure A.7.: Visualization for other layers.

A.2. Non-Linearities

A crucial ingredient in network design is the use of suitable non-linear activations that are applied to individual neurons. These activations may be selected to satisfy constraints imposed on outputs, e.g. those of a probability distribution. Furthermore, they control the descriptive capabilities of a network and impact its convergence properties [35, 61]. Theoretically, all piecewise differentiable non-linear functions can be used as activations. Some, however, have been established as de-facto standard in neural networks. These are presented in the following.

A.2.1. Rectified Linear Units, ReLU

Rectified linear units are used as a common non-linearity that is applied to neuron outputs. It has a linear activation for positive inputs while for negative inputs, it defaults to zero,

$$y = \max(0, x) . \tag{A.4}$$

ReLUs have the advantage of simple and stable gradients for positive activations [113]. However, for negative inputs, the gradient is zero. The ReLU function is shown in Fig. A.8.

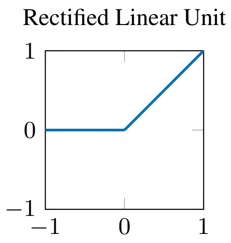


Figure A.8.: Activation of Rectified Linear Unit.

A.2.2. Exponential Linear Units, ELU

Exponential linear units [35] are similar to Rectified Linear Units. The only difference is that they output an exponential function shifted by one for negative inputs,

$$y = \begin{cases} x, & \text{if } 0 < x \\ e^x - 1, & \text{otherwise} \end{cases} \tag{A.5}$$

which leads to non-zero gradients for negative inputs and continuous differentials. The ELU function is shown in Fig. A.9.

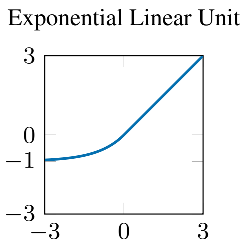


Figure A.9.: Activation of Exponential Linear Unit.

A.2.3. Sigmoid

Sigmoid functions model a soft binary decision [21]. One of the most commonly used sigmoid functions is the function

$$y = \frac{1}{1 + e^{\text{-}x}}. \tag{A.6}$$

For $x \rightarrow \text{-}\infty$, it asymptotically approaches zero while for $x \rightarrow \infty$, y goes to 1. The function is shown in Fig. A.10.

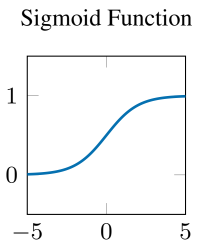


Figure A.10.: Sigmoid Activation.

A.2.4. Softmax

The softmax is an extension of the sigmoid function to multiple inputs,

$$y_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}, \tag{A.7}$$

which can be used to represent a *one-of-many* decision [21]. The name *softmax* points to another interpretation: it provides a differentiable maximum selection of its input.

A.2.5. Dropout

Different appearances of objects result in the need for a multitude of possible features. In network training, however, few dominant features may overrule the optimization of others. To avoid this, a percentage of neurons can be set to zero at random during training time [65]. At test time, this so-called *dropout* is typically disabled to make full use of all possible features.

A.3. Network Architectures

Ongoing research deals with the design of ever-better network architectures for specific tasks. In this work, networks known from literature have been adopted to solve the tasks required for prediction. In the following, architectures used as a basis in any of the previous sections are presented. Note that this is only a very small subset of existing networks.

A.3.1. LeNet-5

LeNet-5, originally designed for handwritten digit recognition, still serves as a blueprint for many convolutional network architectures today [94]. The reason lies in its simplicity. Two pairs of convolution and pooling layers quickly compress information until their output is fed into a classifier stage that consists of three fully connected layers (cf. Fig. A.11). While the original network used the hyperbolic tangent as activation, today most variants employ ReLUs.

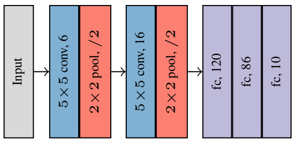


Figure A.11.: LeNet-5 Architecture.

A.3.2. VGG-16

For more challenging problems such as the ImageNet recognition challenge [146], deeper architectures have been proposed. The network designed by Simonyan et al. of the Visual Geometry Group (VGG) still features the same building blocks as LeNet-5 [158]. The network consists of six stages, five convolutional and one fully connected stage. The first two stages consist of two subsequent convolutional layers followed by 2×2 max pooling, the later of three convolutions plus max pooling. All convolutions use 3×3-sized filters, the number of channels per convolution is doubled for every stage. Finally, the fully connected layers use two times 4096 channels before the final classifier, which comprises 1000 outputs corresponding to the classes of ImageNet. The architectures is visualized in Fig. A.12.

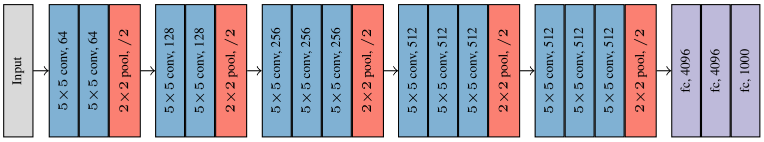
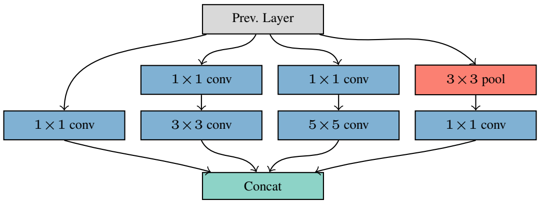


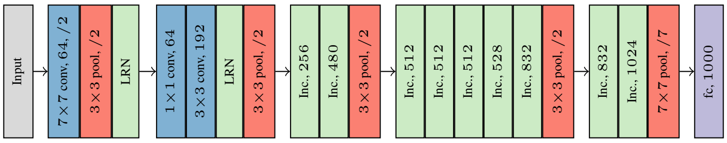
Figure A.12.: VGG-16 Architecture.

A.3.3. Inception

Very deep architectures suffer from slow convergence and high computational demands [62]. The Inception Network aims to overcome these issues through smaller building blocks [161]. These so-called *Inception Blocks*, shown in Fig. A.13a, consist of four parallel paths. They all employ one convolutional layer with kernel size of 1×1 for reduced computational complexity. Additionally, two of the paths use larger convolutional filters while the fourth draws local information from pooling. Finally, the results are concatenated and fed to the next stage. The Inception network (cf. Fig. A.13b) uses nine such blocks, all with different numbers of channels in the individual layers.



(a) Inception Block. Every block features six convoluational layers of which the channel count is a hyperparameter.



(b) Full Inception Network. For clarity, only the channel count after concatenation is reported here.

Figure A.13.: Inception Architecture: the network consists of so-called *Inception Blocks* that combine parallel branches of convolutions and pooling.

A.3.4. Fully Convolutional Network (FCN)

All previously presented architectures were designed for image classification, i.e. they predict a single class probability vector per image. For pixel-wise classification, Fully Convolutional Networks were proposed [101]. In these networks, the final fully-connected stage is exchanged for a convolutional layer with 1×1 kernels. With this, predictions can be made per pixel using any backbone architecture. However, since most backbones feature downsampling throughout the network, it has to be inverted by upsampling. This is done by transposed convolutions that act as bilinear interpolation. Upsampling is applied stepwise so that predictions for different resolutions can be combined. For this, so-called *skip* connections are used. These 1×1 convolutions skip some of the downsampling part of the network and directly contribute to the prediction at the corresponding scale. The architecture can be seen in Fig. A.14.

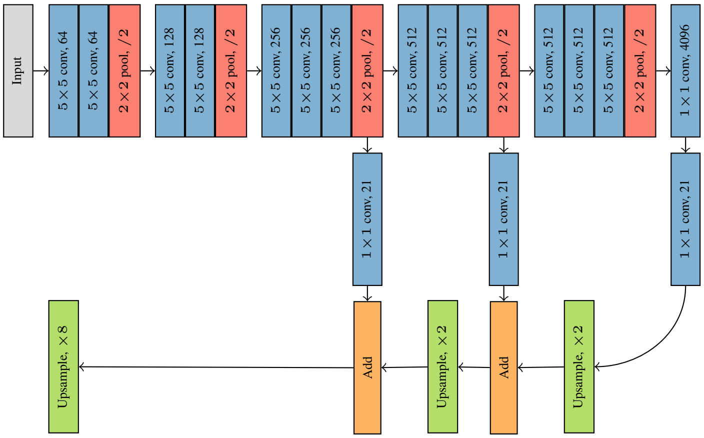


Figure A.14.: Fully Convolutional Network with VGG-16 backbone. The 1×1 convolutions act as skip to incorporate fine grained knowledge into upsampling.

Appendix B

Details on Experiments

The experiments presented in Section 3 and Section 7 include design decisions and hyperparameters that are not the scope of the individual chapters. Therefore, details on implementations, architectures and parameters are summarized in the following.

B.1. Numerical Considerations

In this work, numerical optimization is employed. Therefore, different measures have been taken to ensure numerical stability.

Normalization of probability distributions, e.g. for Eq. (6.15), is computed as a sum of all elements clipped at zero and with an additional small constant ϵ to avoid division by zero, so that

$$s_i = \frac{\max(0, \tilde{s}_i) + \epsilon}{\sum_{\tilde{s}_j \in \tilde{\Phi}} \left[\max(0, \tilde{s}_j) + \epsilon\right]}, \tag{B.1}$$

where the tilde $\tilde{\circ}$ denotes unnormalized values. The constant ϵ=10^{-30} is chosen close to the numerical limit of the precision. Thus, in practice, it only impacts computation in cases close to that.

Limits in logarithms are clipped to some numerical limit,

$$y = \log(\max(\epsilon, x)), \tag{B.2}$$

where x is the value the logarithm is applied to and ϵ=10^{-30} again a small constant close to the precision's numerical limits.

For log-likelihood loss functions, numerically stable versions are used, i.e. log-sum-exp operations for Eq. (5.23) [114] and cross entropy in log space.

The normalization of the Von-Mises-Distribution Eq. (5.3) requires the modified Bessel function $I_0(\kappa)$. Since this needs to be differentiated w.r.t. κ, it is implemented as a Taylor approximation of degree 18. To avoid large deviations between the actual modified Bessel function and its Taylor approximation, κ is clipped for values above 20 .

Since grids are normalized numerically according to Eq. (B.1), the filters for state propagation can be normalized to arbitrary values. To avoid vanishing probabilities, this normalization is chosen per experiment as the highest possible stable value, mostly in $[1, 3]$.

B.2. Reference Experiments

The methods of Section 3 are presented in literature and were implemented as close to the original publications as possible. The Kalman Filter [19] allows for no freedom of choice other than the parameters of initial, process and measurement covariance. For the inverse reinforcement learning, we used available code from the publication [86]. Only the CNN architecture used for Dynamic Occupancy Grid Map (DOGMa) prediction was not fully specified in the publication [69]. Therefore, it was implemented to the according to the referred models [97, 119] including architecture search. The final network is displayed in Fig. B.1.

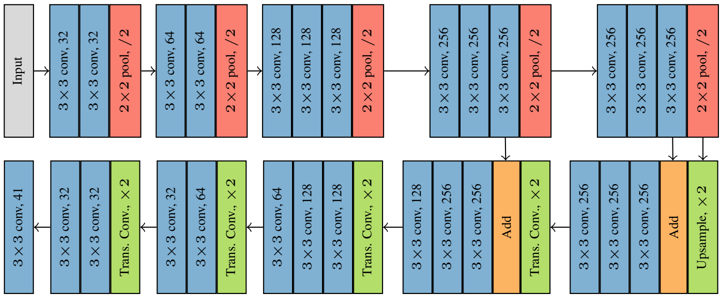


Figure B.1.: The network architecture used for DOGMa-CNN experiments [69]. All convolution and transposed convolution layers use ReLU activations, the final activation is sigmoidal.

B.3. RMDN Experiments

In Section 5.1, a Recurrent Mixture Density Network (RMDN) was presented to predict pedestrian destinations. In the RMDN's evaluation in Section 7.1, design parameters specific to the architecture are assessed. In the following, the default parameters, network design and value tables corresponding to the figures shown in Section 7.1 are presented.

B.3.1. Default Parameters

For the RMDN, architecture and hyperparameter tuning was conducted for all experiments. The parameters found for the network training are listed in Table B.1.

Parameter	Value
Loss	Mean
Optimizer	Adam
Dropout Rate	30%
Base Learning Rate	0.001
Max. Iterations $i_{\max}$	$500,000$
Learning Rate Scaling for Iteration i	$\sqrt{\frac{i_{\max}-i}{i_{\max}}}$
Weight Decay	10^{-6}
Mixture Components	8
LSTM Cell Size	16
Fully-Connecetd Neurons	64
CNN	✗
Prediction Horizon	$4.0s$

Table B.1.: Default parameters used in Section 7.1 for parameter search.

The network architecture for all experiments in Section 7.1 is shown in Fig. B.2. Note that the CNN path is only present in experiments if explicitly stated.

B.3.2. Result Tables

For clarity of presentation, the results of RMDN experiments were shown as plots in Section 7.1. For the sake of completeness, the numeric results of all experiments are listed in tables below.

Parameter	mNLP	mPP / %
Mean	7.1	**7.0**
Worst 1	8.3	1.9
Worst 2	6.5	3.1
Worst 4	7.8	3.8
Worst 6	8.6	3.9
Worst 8	7.0	4.8
Focal γ=1	6.6	4.7
Focal γ=2	**6.4**	4.0
Focal γ=4	7.8	3.4

Table B.2.: Results of RMDN with different loss functions explained in Section 7.1.2.

Parameter	mNLP	mPP / %
N_{Mix}=1	11.9	5.9
N_{Mix}=2	12.9	6.4
N_{Mix}=4	7.6	6.5
N_{Mix}=8	7.1	**7.0**
N_{Mix}=10	**5.9**	5.9
N_{Mix}=12	6.3	5.1
N_{Mix}=16	6.5	5.2

Table B.3.: Results of RMDN with different mixture component counts explained in Section 7.1.3.

Parameter	mNLP	mPP / %
r=0.0	8.3	6.0
r=0.1	7.1	6.4
r=0.2	**5.7**	6.5
r=0.3	7.1	**7.0**
r=0.4	6.6	5.9
r=0.5	8.7	6.5

Table B.4.: Results of RMDN with different mixture component dropout rates explained in Section 7.1.4.

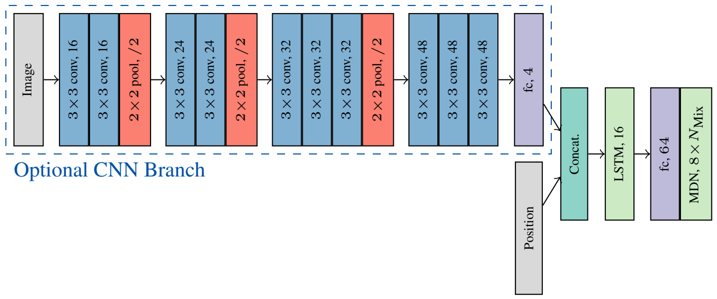


Figure B.2.: Default RMDN architecture used in experiments. All convolution layers feature ELU activations followed by Group Normalization with G=4. For experiments without CNN, the top branch is omitted.

T	mNLP			mPP / %		
	KF	DOGMa	RMDN	KF	DOGMa	RMDN
0.1s	1.1	0.8	**0.0**	79.6	68.0	**98.7**
0.5s	0.5	4.1	**0.1**	85.1	46.7	**95.8**
1.0s	1.2	7.6	**0.5**	59.3	30.7	**75.4**
1.5s	**2.0**	8.5	**2.0**	35.3	20.2	**40.2**
2.0s	2.9	9.0	**2.7**	19.4	15.6	**27.4**
2.5s	3.7	9.1	**3.6**	10.8	10.4	**16.4**
3.0s	4.4	11.3	**4.3**	6.4	9.2	**11.0**
3.5s	**4.9**	11.6	5.0	4.0	**8.1**	8.0
4.0s	**6.9**	10.3	7.1	2.6	**6.7**	6.6

Table B.5.: Destination prediction results for different time horizons T explained in Section 7.1.5.

B.4. Planning Network Experiments

Section 5.2 presented two different takes to goal-directed prediction using neural networks, namely one for Markov-Decision-Processes (MDP) and one for forward-backward propagation (Fwd-Bwd). In the respective evaluation in Section 7.2, design parameters specific to the architectures were analyzed. In the following, the default parameters, network design and value tables corresponding to the figures shown in Section 7.2 and Section 7.3 are presented.

B.4.1. Topology Network

The planning-based prediction networks introduced in this thesis depend on inference of most likely actions given a map of the environment. Regardless of the planning-based prediction method used, the same network architecture is used to predict these actions for all experiments in Section 7.2 and Section 7.3. This allows for fair comparison between the two methods. The corresponding architecture is shown in Fig. B.3.

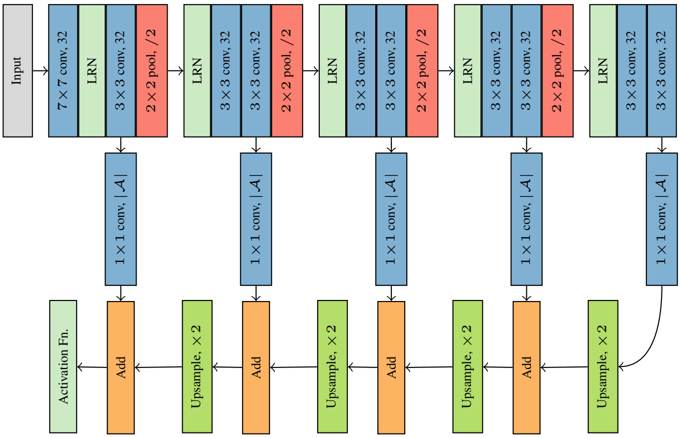


Figure B.3.: Default topology network architecture used in all planning network experiments. The output channel count is the number of possible actions $|\mathcal{A}|$. As activations, ReLU is used in the encoder part, no nonlinearity for skip connections and a final activation depending on the subsequent planning network.

B.4.2. Default Parameters

For both, the Fwd-Bwd- and MDP-Network, hyperparameter tuning for all experiments was conducted. The parameters found for training are listed in Table B.6 and Table B.7. The network architecture is imposed by the respective prediction method and therefore not described here but Section 5.2.

B.4.2.1. Result Tables

For clarity of presentation, the results of the trajectory experiments were shown as plots in Section 7.2 and Section 7.3. For the sake of completeness, the numeric values of the results are listed in the following tables.

Parameter	Value
Loss	Mean
Optimizer	Adam
Base Learning Rate	0.001
Max. Iterations $i_{\max}$	$500,000$
Learning Rate Scaling for Iteration i	$\sqrt{\frac{i_{\max}-i}{i_{\max}}}$
Weight Decay	10^{-6}
Prediction Horizon	$4.0s$
Map Features	Obstacles, road, sidewalk, destination, start, distance to destination, distance to start
Map Size	$16\text{m}\times16\text{m}$
Cell Size	$0.1\text{m}\times0.1\text{m}$
Filter Size	$0.5\text{m}\times0.5\text{m}$
Action Count	13

Table B.6.: Default parameters of Fwd-Bwd network used in Section 7.2 and Section 7.3 for parameter search.

Parameter	Value
Loss	Mean
Optimizer	Adam
Base Learning Rate	0.002
Max. Iterations $i_{\max}$	$500,000$
Learning Rate Scaling for Iteration i	$\sqrt{\frac{i_{\max}-i}{i_{\max}}}$
Weight Decay	10^{-8}
Prediction Horizon	$4.0s$
Map Features	Obstacles, road, sidewalk, destination, start, distance to destination
Map Size	$16\text{m}\times16\text{m}$
Cell Size	$0.1\text{m}\times0.1\text{m}$
Filter Size	$0.9\text{m}\times0.9\text{m}$
Action Count	13
Filter Variance Regularization	10^{-6}
Discount Factor γ	0.99
Value Iteration Steps	80

Table B.7.: Default parameters of MDP network used in Section 7.2 and Section 7.3 for parameter search.

Parameter	mNLP	mPP / %
$\lambda_{\text{var}}=0$	2.2	36.3
$\lambda_{\text{var}}=10^{-8}$	2.1	38.7
$\lambda_{\text{var}}=10^{-7}$	1.8	36.4
$\lambda_{\text{var}}=10^{-6}$	**1.6**	**39.3**
$\lambda_{\text{var}}=10^{-5}$	1.7	34.9

Table B.8.: Performance of MDP depending on the regularization weight of the filter variance explained in Section 7.2.1.

Parameter	mNLP	mPP / %
$\lambda_{\text{var}}=0$	**1.8**	**52.2**
$\lambda_{\text{var}}=10^{-8}$	2.2	50.4
$\lambda_{\text{var}}=10^{-6}$	2.1	50.3
$\lambda_{\text{var}}=10^{-5}$	2.3	49.2
$\lambda_{\text{var}}=10^{-4}$	2.2	49.9

Table B.9.: Performance of Fwd-Bwd depending on the regularization weight of the filter variance explained in Section 7.2.1.

Parameter	mNLP		mPP / %	
	MDP	Fwd-Bwd	MDP	Fwd-Bwd
$\vert\mathcal{A}\vert=5$	2.1	2.3	25.5	46.4
$\vert\mathcal{A}\vert=9$	**1.8**	2.2	38.1	46.6
$\vert\mathcal{A}\vert=13$	2.0	2.1	41.9	**50.3**
$\vert\mathcal{A}\vert=21$	**1.8**	2.2	41.3	48.2

Table B.10.: Performance of prediction depending on the number of possible actions explained in Section 7.2.2.

Features	mNLP		mPP / %	
	MDP	Fwd-Bwd	MDP	Fwd-Bwd
Curb	2.1	2.0	38.5	51.3
Obstacle	2.2	1.8	38.7	51.7
Topview	2.0	2.2	41.6	52.9
Road	**1.7**	2.2	42.0	52.3
Sidewalk	2.0	2.1	44.6	**54.6**
All	2.0	2.1	41.9	50.3
None	1.9	2.1	39.2	50.0

Table B.11.: Performance of prediction depending on the input features of the map explained in Section 7.2.3.

Setup	mNLP		mPP / %	
	MDP	Fwd-Bwd	MDP	Fwd-Bwd
Separate Training	2.7	2.8	30.9	37.0
Joint Training	**2.6**	2.7	29.8	**41.0**

Table B.12.: Performance of joint prediction of destinations and trajectory explained in Section 7.3.

T	mNLP			mPP / %		
	RMDN	MDP	Fwd-Bwd	RMDN	MDP	Fwd-Bwd
0.1s	**0.0**	**0.0**	**0.0**	98.7	96.6	**99.6**
0.5s	**0.1**	0.8	0.4	**95.8**	61.8	80.9
1.0s	**0.5**	1.4	0.9	**75.4**	40.8	68.1
1.5s	2.0	1.9	**1.6**	40.2	30.7	**50.2**
2.0s	2.7	**2.5**	**2.5**	27.4	23.5	**33.4**
2.5s	3.6	**3.1**	3.5	16.4	16.6	**22.0**
3.0s	4.3	**3.7**	4.3	11.0	11.9	**16.0**
3.5s	5.0	**4.3**	5.1	8.0	9.6	**12.4**
4.0s	7.1	6.0	**5.9**	6.6	**7.7**	5.7

Table B.13.: Performance of joint prediction for different time horizons T explained in Section 7.3.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.* (2016, March) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467. https://arxiv.org/abs/1603.04467 [Online. Accessed: 2020-07-25]

[2] L. F. Abbott, "Lapicque's Introduction of the Integrate-and-Fire Model Neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.

[3] S. Agarwal, K. Mierle, and Others. (2018) Ceres solver. http://ceres-solver.org [Online. Accessed: 2020-07-26]

[4] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building Rome in a Day," in *IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 2009, pp. 72–79.

[5] C. C. Aggarwal, *Data Classification: Algorithms and Applications*. Boca Raton, FL, USA: CRC Press, 2014.

[6] 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)*, Las Vegas, NV, USA, June 2016, pp. 961–971.

[7] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous Vehicle Technology: A Guide for Policymakers*. Santa Monica, CA, USA: Rand Corporation, 2014.

[8] M. Andriluka, S. Roth, and B. Schiele, "Monocular 3D Pose Estimation and Tracking by Detection," in *IEEE Conference on Computer*

Vision and Pattern Recognition (CVPR), San Francisco, CA, USA, June 2010, pp. 623–630.

[9] I. Asimov, *I, Robot*. New York City, NY, USA: Gnome Press, 1950, vol. 1.

[10] A. Barth and U. Franke, “Where Will the Oncoming Vehicle be the Next Second?” in *IEEE Intelligent Vehicles Symposium (IV)*, Eindhoven, Netherlands, June 2008, pp. 1068–1073.

[11] T. Batz, K. Watson, and J. Beyerer, “Recognition of Dangerous Situations Within a Cooperative Group of Vehicles,” in *IEEE Intelligent Vehicles Symposium (IV)*, Xi’an, China, June 2009, pp. 907–912.

[12] B. Benfold and I. Reid, “Guiding Visual Surveillance by Tracking Human Attention,” in *British Machine Vision Conference (BMVC)*, London, UK, September 2009, pp. 14.1–14.11.

[13] M. Bennewitz, W. Burgard, and S. Thrun, “Using EM to Learn Motion Behaviors of Persons With Mobile Robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, Lausanne, Switzerland, September 2002, pp. 502–507.

[14] A. Bera, T. Randhavane, and D. Manocha, “Aggressive, Tense or Shy? Identifying Personality Traits from Crowd Videos.” in *International Joint Conference on Artificial Intelligence*, Melbourne, Australia, August 2017, pp. 112–118.

[15] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron *et al.*, “Theano: Deep Learning on GPUs With Python,” in *Advances in Neural Information Processing (NeurIPS) Workshop*, vol. 3, Granada, Spain, December 2011, pp. 1–48.

[16] M. Bertozzi, A. Broggi, A. Fascioli, A. Tibaldi, R. Chapuis, and F. Chausse, “Pedestrian Localization and Tracking System With Kalman Filtering,” in *IEEE Intelligent Vehicles Symposium (IV)*, Parma, Italy, June 2004, pp. 584–589.

[17] G. Best and R. Fitch, “Bayesian Intention Inference for Trajectory Prediction With an Unknown Goal Destination,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, October 2015, pp. 5817–5823.

[18] R. A. Best and J. Norton, “A New Model and Efficient Tracker for a Target With Curvilinear Motion,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 3, pp. 1030–1037, 1997.

[19] E. Binelli, A. Broggi, A. Fascioli, S. Ghidoni, P. Grisleri, T. Graf, and M. Meinecke, “A Modular Tracking System for Far Infrared Pedestrian Recognition,” in *IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, NV, USA, June 2005, pp. 759–764.

[20] C. M. Bishop, “Mixture Density Networks,” Department of Computer Science and Applied Mathematics, Aston University, Tech. Rep., February 1994.

[21] C. M. Bishop, *Machine Learning and Pattern Recognition*. Heidelberg, Germany: Springer, 2006.

[22] G. Bishop and G. Welch, “An Introduction to the Kalman Filter,” in *Proceedings of SIGGRAPH*, vol. 8, no. 27599-23175, Los Angeles, CA, USA, August 2001, p. 41.

[23] S. Bonnin, T. H. Weisswange, F. Kummert, and J. Schmuedderich, “Pedestrian Crossing Prediction Using Multiple Context-Based Models,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, October 2014, pp. 378–385.

[24] L. Bottou and O. Bousquet, “The Tradeoffs of Large Scale Learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, December 2008, pp. 161–168.

[25] D. S. Broomhead and D. Lowe, “Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks,” Royal Signals and Radar Establishment Malvern (United Kingdom), Malvern, England, Tech. Rep., July 1988.

[26] N. Brouwer, H. Kloeden, and C. Stiller, “Comparison and Evaluation of Pedestrian Motion Models for Vehicle Safety Systems,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, Brazil, October 2016, pp. 2207–2212.

[27] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*. Heidelberg, Germany: Springer, 2007, vol. 36.

[28] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Heidelberg, Germany: Springer, 2009, vol. 56.

[29] Y. Cai, N. de Freitas, and J. J. Little, “Robust Visual Tracking for Multiple Targets,” in *European Conference on Computer Vision (ECCV)*, Graz, Austria, May 2006, pp. 107–118.

[30] 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)*, Long Beach, CA, USA, 2019, pp. 8748–8757.

[31] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic Image Segmentation With Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[32] Z. Chen, D. C. Ngai, and N. Yung, “Pedestrian Behavior Prediction Based on Motion Patterns for Vehicle-to-Pedestrian Collision Avoidance,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Beijing, China, October 2008, pp. 316–321.

[33] S. Chik, C. Yeong, E. Su, T. Lim, Y. Subramaniam, and P. Chin, “A Review of Social-Aware Navigation Frameworks for Service Robot in Dynamic Human Environments,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 8, no. 11, pp. 41–50, 2016.

[34] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. (2014, June) Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078. https://arxiv.org/abs/1406.1078 [Online. Accessed: 2020-07-25]

[35] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. (2015, November) Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289. https://arxiv.org/abs/1511.07289 [Online. Accessed: 2020-07-25]

[36] R. Collobert, C. Farabet, K. Kavukcuoglu *et al.*, “Torch,” in *Advances in Neural Information Processing (NeurIPS) Workshop*, vol. 113, Vancouver, Canada, December 2008.

[37] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June 2016, pp. 3213–3223.

[38] P. Coscia, F. Castaldo, F. A. Palmieri, A. Alahi, S. Savarese, and L. Ballan, “Long-Term Path Prediction in Urban Scenarios Using Circular Distributions,” *Image and Vision Computing*, vol. 69, pp. 81–91, 2018.

[39] P. A. Devijver, “Baum’s Forward-Backward Algorithm Revisited,” *Pattern Recognition Letters*, vol. 3, no. 6, pp. 369–373, 1985.

[40] E.-D. Dickmanns, “Vehicle Guidance by Computer Vision,” in *High Precision Navigation*. Heidelberg, Germany: Springer, 1989, pp. 86–96.

[41] N. Djuric, V. Radosavljevic, H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, N. Singh, and J. Schneider, “Uncertainty-Aware Short-Term Motion Prediction of Traffic Actors for Autonomous Driving,” in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Village, CO, USA, March 2020, pp. 2095–2104.

[42] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A Deep Convolutional Activation Feature for Generic Visual Recognition,” in *International Conference on Machine Learning (ICML)*, Beijing, China, June 2014, pp. 647–655.

[43] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June 2015, pp. 2625–2634.

[44] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, “FlowNet: Learning Optical Flow With Convolutional Networks,” in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 2015, pp. 2758–2766.

[45] D. Eigen and R. Fergus, “Predicting Depth, Surface Normals and Semantic Labels With a Common Multi-Scale Convolutional Architec-

ture," in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 2015, pp. 2650–2658.

[46] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[47] J. Elfring, R. Van De Molengraft, and M. Steinbuch, "Learning Intentions for Improved Human Motion Prediction," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 591–602, 2014.

[48] A. Elnagar, "Prediction of Moving Objects in Dynamic Environments Using Kalman Filters," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. Banff, Canada: IEEE, July 2001, pp. 414–419.

[49] A. Ess, B. Leibe, K. Schindler, , and L. van Gool, "A Mobile Vision System for Robust Multi-Person Tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AL, USA, June 2008, pp. 1–8.

[50] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2001, vol. 1, no. 10.

[51] P. Furgale, U. Schwesinger, M. Rufli, W. Derendarz, H. Grimmett, P. Mühlfellner, S. Wonneberger, J. Timpner, S. Rottmann, B. Li *et al.*, "Toward Automated Driving in Cities Using Close-to-Market Sensors: An Overview of the V-Charge Project," in *IEEE Intelligent Vehicles Symposium (IV)*, Gold Coast, Australia, June 2013, pp. 809–816.

[52] A. Geiger, P. Lenz, and R. Urtasun, "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, RI, USA, June 2012, pp. 3354–3361.

[53] F. A. Gers and J. Schmidhuber, "Recurrent Nets That Time and Count," in *IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJNN)*, vol. 3, Como, Italy, July 2000, pp. 189–194.

[54] T. Gindele, S. Brechtel, and R. Dillmann, "A Probabilistic Model for Estimating Driver Behaviors and Vehicle Trajectories in Traffic Environments," in *IEEE International Conference on Intelligent Transportation Systems*, Madeira Island, Portugal, September 2010, pp. 1625–1631.

[55] X. Glorot and Y. Bengio, “Understanding the Difficulty of Training Deep Feedforward Neural Networks,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Sardinia, Italy, May 2010, pp. 249–256.

[56] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, vol. 1.

[57] Y. Gu, Y. Hashimoto, L.-T. Hsu, and S. Kamijo, “Motion Planning Based on Learning Models of Pedestrian and Driver Behaviors,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, Brazil, October 2016, pp. 808–813.

[58] H. Harms, E. Rehder, and M. Lauer, “Grid Map Based Free Space Estimation Using Stereo Vision,” in *IEEE Intelligent Vehicles Symposium (IV) Workshop*, Seoul, Korea, June 2015.

[59] M. Harrower and C. A. Brewer, “ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps,” *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003.

[60] I. Hasan, F. Setti, T. Tsesmelis, A. Del Bue, F. Galasso, and M. Cristani, “MX-LSTM: Mixing Tracklets and Vislets to Jointly Forecast Trajectories and Head Poses,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018, pp. 6067–6076.

[61] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep Into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification,” in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 2015, pp. 1026–1034.

[62] 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)*, Las Vegas, NV, USA, June 2016, pp. 770–778.

[63] D. Helbing and P. Molnar, “Social Force Model for Pedestrian Dynamics,” *Physical Review E*, vol. 51, no. 5, p. 4282, 1995.

[64] P. Henry, C. Vollmer, B. Ferris, and D. Fox, “Learning to Navigate Through Crowded Environments,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, May 2010, pp. 981–986.

[65] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. (2012, July) Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. arXiv:1207.0580. https://arxiv.org/abs/1207.0580 [Online. Accessed: 2020-07-25]

[66] T. Hirakawa, T. Yamashita, T. Tamaki, and H. Fujiyoshi, "Survey on Vision-Based Path Prediction," in *International Conference on Distributed, Ambient, and Pervasive Interactions*, Las Vegas, NV, USA, July 2018, pp. 48–64.

[67] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[68] P. G. Hoel, *Introduction to Mathematical Statistics.* Hoboken, NJ, USA & London, UK: John Wiley & Sons, Inc., & Chapman & Hall, Ltd., 1984, no. 5th Ed.

[69] S. Hoermann, M. Bach, and K. Dietmayer, "Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach With Fully Automatic Labeling," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 2056–2063.

[70] J. J. Hopfield, "Neural Networks and Physical Systems With Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[71] S. Huang, X. Li, Z. Zhang, Z. He, F. Wu, W. Liu, J. Tang, and Y. Zhuang, "Deep Learning Driven Visual Path Prediction From a Single Image," *IEEE Transactions on Image Processing*, vol. 25, no. 12, pp. 5892–5904, 2016.

[72] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The Apolloscape Dataset for Autonomous Driving," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop*, Salt Lake City, UT, USA, June 2018, pp. 954–960.

[73] D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[74] T. Ikeda, Y. Chigodo, D. Rea, F. Zanlungo, M. Shiomi, and T. Kanda, "Modeling and Prediction of Pedestrian Behavior Based on the Sub-Goal Concept," *Robotics*, vol. 10, pp. 137–144, 2013.

[75] H. O. Jacobs, O. K. Hughes, M. Johnson-Roberson, and R. Vasudevan, "Real-Time Certified Probabilistic Pedestrian Forecasting," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2064–2071, 2017.

[76] H. Jaspers, M. Himmelsbach, and H.-J. Wuensche, "Multi-Modal Local Terrain Maps From Vision and LiDAR," in *IEEE Intelligent Vehicles Symposium (IV)*, Redondo Beach, CA, USA, June 2017, pp. 1119–1125.

[77] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *ACM International Conference on Multimedia (MM)*, Orlando, FL, USA, November 2014, pp. 675–678.

[78] N. Jourdan, E. Rehder, and U. Franke, "Identification of Uncertainty in Artificial Neural Networks," in *Workshop Fahrerassistenzsysteme*, Walting, Germany, 2020, pp. 1–6.

[79] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[80] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schöder, M. Thuy, M. Goebl, F. von Hundelshausen, O. Pink, C. Frese, and C. Stiller, *Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007.* Heidelberg, Germany: Springer, 2009, vol. 56.

[81] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous Land Vehicle Project at CMU," in *ACM Conference on Computer Science (CSC)*, Cincinnati, OH, USA, Feburary 1986, pp. 71–80.

[82] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-Aware Long-Term Prediction of Pedestrian Motion," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016, pp. 2543–2549.

[83] C. Keller and D. Gavrila, "Will the Pedestrian Cross? A Study on Pedestrian Path Prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 494–506, April 2013.

[84] A. Kendall, Y. Gal, and R. Cipolla, “Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018, pp. 7482–7491.

[85] D. P. Kingma and J. Ba. (2014, December) Adam: A Method for Stochastic Optimization. arXiv:1412.6980. https://arxiv.org/abs/1412.6980 [Online. Accessed: 2020-07-25]

[86] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, “Activity Forecasting,” in *European Conference on Computer Vision (ECCV)*, Florence, Italy, October 2012, pp. 201–214.

[87] S. Köhler, M. Goldhammer, S. Bauer, K. Doll, U. Brunsmann, and K. Dietmayer, “Early Detection of the Pedestrian’s Intention to Cross the Street,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Anchorage, AK, USA, September 2012, pp. 1759–1764.

[88] J. F. P. Kooij, F. Flohr, E. A. I. Pool, and D. M. Gavrila, “Context-based path prediction for targets with switching dynamics,” *International Journal of Computer Vision*, vol. 127, no. 3, pp. 239–262, March 2019.

[89] J. F. P. Kooij, N. Schneider, F. Flohr, and D. M. Gavrila, “Context-Based Pedestrian Path Prediction,” in *European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, September 2014, pp. 618–633.

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet Classification With Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, Lake Tahoe, NV, USA, December 2012, pp. 1097–1105.

[91] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, “Human-Aware Robot Navigation: A Survey,” *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013.

[92] F. Kuhnt, R. Kohlhaas, T. Schamm, and J. M. Zöllner, “Towards a Unified Traffic Situation Estimation Model-Street-Dependent Behaviour and Motion Models,” in *International Conference on Information Fusion (Fusion)*, Washington, DC, USA, July 2015, pp. 1223–1229.

[93] P. A. Lasota, T. Fong, J. A. Shah *et al.*, “A Survey of Methods for Safe Human-Robot Interaction,” *Foundations and Trends® in Robotics*, vol. 5, no. 4, pp. 261–349, 2017.

[94] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[95] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional Networks and Applications in Vision,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Paris, France, May 2010, pp. 253–256.

[96] S. Lefèvre, D. Vasquez, and C. Laugier, “A Survey on Motion Prediction and Risk Assessment for Intelligent Vehicles,” *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[97] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, July 2017, pp. 2117–2125.

[98] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” in *IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, October 2017, pp. 2980–2988.

[99] M. L. Littman, “Value-Function Reinforcement Learning in Markov Games,” *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[100] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single Shot Multibox Detector,” in *European Conference on Computer Vision (ECCV)*, Amsterdam, Netherlands, October 2016, pp. 21–37.

[101] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June 2015, pp. 3431–3440.

[102] M. Luber, J. A. Stork, G. D. Tipaldi, and K. O. Arras, “People Tracking With Human Motion Predictions From Social Forces,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, May 2010, pp. 464–469.

[103] W. Luo, B. Yang, and R. Urtasun, “Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting With a Single Convolutional Net,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018, pp. 3569–3577.

[104] W.-C. Ma, D.-A. Huang, N. Lee, and K. M. Kitani, “Forecasting Interactive Dynamics of Pedestrians With Fictitious Play,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, July 2017, pp. 774–782.

[105] K. V. Mardia and P. E. Jupp, *Directional Statistics*. Hoboken, NJ, USA: John Wiley & Sons, 2009, vol. 494.

[106] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, Eds., *Autonomous Driving*. Heidelberg, Germany: Springer, 2016.

[107] S. Merity, B. McCann, and R. Socher. (2017, August) Revisiting Activation Regularization for Language RNNs. arXiv:1708.01009. https://arxiv.org/abs/1708.01009 [Online. Accessed: 2020-07-25]

[108] R. Q. Mínguez, I. P. Alonso, D. Fernández-Llorca, and M. Á. Sotelo, “Pedestrian Path, Pose, and Intention Prediction Through Gaussian Process Dynamical Models and Pedestrian Activity Recognition,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 5, pp. 1803–1814, 2018.

[109] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, “Trajectory Analysis and Prediction for Improved Pedestrian Safety: Integrated Framework and Evaluations,” in *IEEE Intelligent Vehicles Symposium (IV)*, Seoul, Korea, June 2015, pp. 330–335.

[110] B. T. Morris and M. M. Trivedi, “A Survey of Vision-Based Trajectory Learning and Analysis for Surveillance,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 8, pp. 1114–1127, 2008.

[111] D. Munoz, J. A. Bagnell, and M. Hebert, “Stacked Hierarchical Labeling,” in *European Conference on Computer Vision (ECCV)*, Heraklion, Crete, Greece, September 2010, pp. 57–70.

[112] V. Murino, M. Cristani, S. Shah, and S. Savarese, *Group and Crowd Behavior for Computer Vision*. Cambridge, MA, USA: Academic Press, 2017.

[113] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in *International Conference on Machine Learning (ICML)*, Haifa, Israel, June 2010, pp. 807–814.

[114] F. Nielsen and K. Sun, “Guaranteed Bounds on the Kullback–Leibler Divergence of Univariate Mixtures,” *IEEE Signal Processing Letters*, vol. 23, no. 11, pp. 1543–1546, 2016.

[115] N.N., “"Phantom-Auto" Will Tour City,” *The Milwaukee Sentinel*, December 1925.

[116] N.N., “Science: Radio auto,” *Time Magazine*, vol. 6, no. 6, August 1925.

[117] N.N., *Verkehrsunfälle 2020*. Wiesbaden, Germany: Statistisches Bundesamt, July 2020, vol. 8, no. 7.

[118] N.N. (2020) Waymo. Mountain View, CA, USA. http://waymo.com [Online. Accessed: 2020-07-26]

[119] H. Noh, S. Hong, and B. Han, “Learning Deconvolution Network for Semantic Segmentation,” in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 2015, pp. 1520–1528.

[120] D. Nuß, “A Random Finite Set Approach for Dynamic Occupancy Grid Maps,” Doctoral thesis, Universität Ulm, Ulm, Germany, 2017.

[121] S. Pellegrini, A. Ess, and L. Van Gool, “Improving Data Association by Joint Modeling of Pedestrian Trajectories and Groupings,” in *European Conference on Computer Vision (ECCV)*, Heraklion, Crete, Greece, September 2010, pp. 452–465.

[122] M. Pfeiffer, U. Schwesinger, H. Sommer, E. Galceran, and R. Siegwart, “Predicting Actions to Act Predictably: Cooperative Partial Motion Planning with Maximum Entropy Models,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, October 2016, pp. 2096–2101.

[123] F. Previtali, A. Bordallo, L. Iocchi, and S. Ramamoorthy, “Predicting Future Agent Motions for Dynamic Environments,” in *IEEE International Conference on Machine Learning and Applications (ICMLA)*, Anaheim, CA, USA, December 2016, pp. 94–99.

[124] J. Quehl, H. Hu, Ö. S. Tas, E. Rehder, and M. Lauer, "How Good is My Prediction? Finding a Similarity Measure for Trajectory Prediction Evaluation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, October 2017, pp. 1–6.

[125] R. Quintero, J. Almeida, D. F. Llorca, and M. Sotelo, "Pedestrian Path Prediction Using Body Language Traits," in *IEEE Intelligent Vehicles Symposium (IV)*, Dearborn, MI, USA, June 2014, pp. 317–323.

[126] B. Ranft and T. Strauß, "Modeling Arbitrarily Oriented Slanted Planes for Efficient Stereo Vision Based on Block Matching," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, October 2014, pp. 1941–1947.

[127] E. Rehder and A. Albrecht, "Submap-Based SLAM for Road Markings," in *IEEE Intelligent Vehicles Symposium (IV)*, Seoul, Korea, June 2015, pp. 1393–1398.

[128] E. Rehder, C. Kinzig, P. Bender, and M. Lauer, "Online Stereo Camera Calibration From Scratch," in *IEEE Intelligent Vehicles Symposium (IV)*, Redondo Beach, CA, USA, June 2017, pp. 1694–1699.

[129] E. Rehder, H. Klöden, and C. Stiller, "Planungsbasierte Fußgängerprädiktion," in *Workshop Fahrerassistenzsysteme*, Walting, Germany, 2015, p. 129.

[130] E. Rehder and H. Kloeden, "Goal-directed pedestrian prediction," in *IEEE International Conference on Computer Vision (ICCV) Workshop*, Santiago, Chile, December 2015, pp. 50–58.

[131] E. Rehder, H. Kloeden, and C. Stiller, "Head Detection and Orientation Estimation for Pedestrian Safety," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, October 2014, pp. 2292–2297.

[132] E. Rehder, J. Quehl, and C. Stiller, "Driving Like a Human: Imitation Learning for Path Planning Using Convolutional Neural Networks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop*, Vancouver, Canada, September 2017.

[133] E. Rehder, F. Wirth, M. Lauer, and C. Stiller, "Pedestrian Prediction by Planning using Deep Neural Networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 1–5.

[134] N. Rhinehart, K. M. Kitani, and P. Vernaza, "R2P2: A Reparameterized Pushforward Policy for Diverse, Precise Generative Path Forecasting," in *European Conference on Computer Vision (ECCV)*, Munich, Germany, September 2018, pp. 772–788.

[135] N. Rhinehart, R. McAllister, and S. Levine. (2018, October) Deep Imitative Models for Flexible Inference, Planning, and Control. arXiv:1810.06544. https://arxiv.org/abs/1810.06544 [Online. Accessed: 2020-07-25]

[136] D. Ridel, E. Rehder, M. Lauer, C. Stiller, and D. Wolf, "A Literature Review on the Prediction of Pedestrian Behavior in Urban Scenarios," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, November 2018, pp. 3105–3112.

[137] D. A. Ridel, N. Deo, D. Wolf, and M. Trivedi, "Understanding Pedestrian-Vehicle Interactions with Vehicle Mounted Vision: An LSTM Model and Empirical Analysis," in *IEEE Intelligent Vehicles Symposium (IV)*, Paris, France, June 2019, pp. 913–918.

[138] P. Riekert and T.-E. Schunck, "Zur Fahrmechanik des Gummibereiften Kraftfahrzeugs," *Ingenieur-Archiv*, vol. 11, no. 3, pp. 210–224, 1940.

[139] C. Rösmann, M. Oeljeklaus, F. Hoffmann, and T. Bertram, "Online Trajectory Prediction and Planning for Social Robot Navigation," in *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, Munich, Germany, July 2017, pp. 1255–1260.

[140] M. Roth, F. Flohr, and D. M. Gavrila, "Driver and Pedestrian Awareness-Based Collision Risk Analysis," in *IEEE Intelligent Vehicles Symposium (IV)*, Gotenburg, Sweden, June 2016, pp. 454–459.

[141] A. Rudenko, L. Palmieri, and K. O. Arras, "Predictive Planning for a Mobile Robot in Human Environments," in *IEEE International Conference on Robotics and Automation (ICRA), Workshop on PlanRob*, Singapore, May 2017.

[142] A. Rudenko, L. Palmieri, and K. O. Arras, “Joint Long-Term Prediction of Human Motion Using a Planning-Based Social Force Approach,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 1–7.

[143] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, “Human Motion Trajectory Prediction: A Survey,” *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[144] A. Rudenko, L. Palmieri, A. J. Lilienthal, and K. O. Arras, “Human Motion Prediction Under Social Grouping Constraints,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 2018, pp. 3358–3364.

[145] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Representations by Back-Propagating Errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[146] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[147] A. Sadat, S. Casas, M. Ren, X. Wu, P. Dhawan, and R. Urtasun, “Perceive, Predict, and Plan: Safe Motion Planning Through Interpretable Semantic Representations,” in *European Conference on Computer Vision (ECCV)*, Virtual Conference, 2020, pp. 414–430.

[148] T. Saito and M. Rehmsmeier, “The Precision-Recall Plot is More Informative Than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets,” *PloS one*, vol. 10, no. 3, pp. 1–21, 2015.

[149] H. Sak, A. Senior, and F. Beaufays, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling,” in *Fifteenth Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Singapore, September 2014, pp. 338–342.

[150] S. Schmidt, B. Färber, and A. Pérez Grassi, “Geht er oder geht er nicht?–Ein FAS zur Vorhersage von Fußgängerabsichten,” in *Workshop Fahrerassistenzsysteme*, Walting, Germany, 2008, pp. 2–4.

[151] S. Schmidt and B. Färber, "Pedestrians at the Kerb–Recognising the Action Intentions of Humans," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 12, no. 4, pp. 300–310, 2009.

[152] N. Schneider and D. M. Gavrila, "Pedestrian Path Prediction with Recursive Bayesian Filters: A Comparative Study," in *German Conference on Pattern Recognition (GCPR)*, Saarbrücken, Germany, September 2013, pp. 174–183.

[153] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, "What the Constant Velocity Model can Teach us About Pedestrian Motion Prediction," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1696–1703, 2020.

[154] A. T. Schulz and R. Stiefelhagen, "A Controlled Interactive Multiple Model Filter for Combined Pedestrian Intention Recognition and Path Prediction," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Las Palmas, Gran Canaria, Spain, September 2015, pp. 173–178.

[155] T. Shankar, S. K. Dwivedy, and P. Guha, "Reinforcement Learning via Recurrent Convolutional Neural Networks," in *International Conference on Pattern Recognition (ICPR)*, Cancún, Mexico, December 2016, pp. 2592–2597.

[156] M. Shen, G. Habibi, and J. P. How, "Transferable Pedestrian Motion Prediction Models at Intersections," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 2018, pp. 4547–4553.

[157] N. Silberman and S. Guadarrama. (2016) TensorFlow-Slim Image Classification Model Library. https://github.com/tensorflow/models/tree/master/research/slim [Online. Accessed: 2020-07-25]

[158] K. Simonyan and A. Zisserman. (2014, September) Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. https://arxiv.org/abs/1409.1556 [Online. Accessed: 2020-07-25]

[159] T. Strauß, J. Ziegler, and J. Beck, "Calibrating Multiple Cameras With Non-Overlapping Views Using Coded Checkerboard Targets," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, October 2014, pp. 2623–2628.

[160] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998, vol. 135.

[161] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June 2015, pp. 1–9.

[162] S. Tadokoro, Y. Ishikawa, T. Takebe, and T. Takamori, "Stochastic Prediction of Human Motion and Control of Robots in the Service of Human," in *IEEE Systems, Man and Cybernetics Conference (SMC)*, vol. 1, Le Touquet, France, October 1993, pp. 503–508.

[163] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, Barcelona, Spain, December 2016, pp. 2154–2162.

[164] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10 781–10 790.

[165] P. Trautman and A. Krause, "Unfreezing the Robot: Navigation in Dense, Interacting Crowds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010, pp. 797–803.

[166] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment-A Modern Synthesis," in *IEEE International Conference on Computer Vision (ICCV) Workshop*, Corfu, Greece, September 1999, pp. 298–372.

[167] V. V. Unhelkar, C. Pérez-D'Arpino, L. Stirling, and J. A. Shah, "Human-Robot Co-Navigation Using Anticipatory Indicators of Human Walking Motion," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, May 2015, pp. 6183–6190.

[168] P. Vasishta, D. Vaufreydaz, and A. Spalanzani, "Natural Vision Based Method for Predicting Pedestrian Behaviour in Urban Environments," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, October 2017, pp. 1–6.

[169] D. Vasquez, "Novel Panning-Based Algorithms for Human Motion Prediction," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016, pp. 3317–3322.

[170] A. Vemula, K. Muelling, and J. Oh, "Modeling cooperative navigation in dense human crowds," in *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 1685–1692.

[171] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to Sequence-Video to Text," in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, December 2015, pp. 4534–4542.

[172] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding Convolution for Semantic Segmentation," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Lake Tahoe, NV, USA, March 2018, pp. 1451–1460.

[173] C. J. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[174] U. Weidmann, "Transporttechnik der Fußgänger: Transporttechnische Eigenschaften des Fußgängerverkehrs, Literaturauswertung," *IVT Schriftenreihe*, vol. 90, 1993.

[175] P. M. Williams, "Using Neural Networks to Model Conditional Multivariate Densities," *Neural Computation*, vol. 8, no. 4, pp. 843–854, 1996.

[176] F. Wirth, S. Krane, M. Loos, E. Rehder, and C. Fernandez, "What Does a Good Prediction Look Like?" in *IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, October 2019, pp. 1594–1599.

[177] Y. Wu and K. He, "Group Normalization," in *European Conference on Computer Vision (ECCV)*, Munich, Germany, September 2018, pp. 3–19.

[178] S. Xiao, Z. Wang, and J. Folkesson, "Unsupervised Robot Learning to Predict Person Motion," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, May 2015, pp. 691–696.

[179] S. Yi, H. Li, and X. Wang, “Pedestrian Behavior Modeling From Stationary Crowds With Applications to Intelligent Surveillance,” *IEEE Transactions on Image Processing*, vol. 25, no. 9, pp. 4354–4368, 2016.

[180] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu, “Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June 2016, pp. 4584–4593.

[181] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid Scene Parsing Network,” in *IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, July 2017, pp. 2881–2890.

[182] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. Bagnell, M. Hebert, A. Dey, and S. Srinivasa, “Planning-Based Prediction for Pedestrians,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 2009, pp. 3931–3936.

[183] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum Entropy Inverse Reinforcement Learning,” in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 8, Chicago, IL, USA, July 2008, pp. 1433–1438.

[184] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller *et al.*, “Making Bertha Drive-An Autonomous Journey on a Historic Route,” *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.