On-Board Vehicle Tracking and Behavior Anticipation for Advanced Driver Assistance Systems

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard-Karls-Universitat Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Andreas Alin
aus Schweinfurt

Tübingen
2014

Tag der mündlichen Qualifikation:	03.12.2014
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Martin V. Butz
2. Berichterstatter:	Dr.-Ing. habil. Jannik Fritsch

Acknowledgments

First, I want to thank my supervisors, Martin Butz and Jannik Fritsch. Martin Butz, for his support and encouragement, and the demanding, but great time at his lab. I will surely remember and miss this time for a very long time. Jannik Fritsch, particularly, for supporting me with the changing challenges that come along with a business concern, and for his constructive criticism, when I was presenting the research done.

Next, I want to thank the members of the department of cognitive modeling for all the stimulating discussions and the great time we had at the conferences and at the excursions. Furthermore I want to thank my colleagues at the Honda Research Institute Europe (HRI-EU) for supporting me with their technical know-how and for providing the real-world dataset. I also want to thank the HRI-EU for the funding that enables this thesis.

I also want to say thank you to Christian Weber, Johannes Lohmann and Patrick Stalph for the helpful reviews.

Last but not least I want to thank all persons that enabled and encouraged me to study or finally to write my thesis. All my indulgent and encouraging teachers I had, particularly Mr. Hubertus Karsten and Mrs. Petra Brauetigam. My fiancée for her great support and for being so patient with me. And my parents for giving me the opportunity to learn and study.

Abstract

This thesis develops a probabilistic vehicle tracking and behavior anticipation system for advanced driver assistance systems. Advanced Driver Assistance System(s) (ADAS) need to know where other vehicles in the surrounding of the ego-vehicle are or will be in a few seconds, in order to maintain a safe driving state by activating evasive actions or issuing warnings to the driver in the event of a critical situation. The difficulty of this task are the uncertainties in driver behavior and sensor readings. Since sensors are never accurate and behavior cannot be predicted exactly, the internal estimate of the surrounding world state has to be modeled probabilistically.

In contrast to most state-of-the-art approaches in the vehicle tracking domain, this thesis represents the uncertainty of an individual vehicle position estimate by a probabilistic grid representation also known as a Bayesian Histogram Filter (BHF). This representation handles multi-modal distributions by saving the probability in individual grid cells and propagates them through the grid by model assumptions that simulate real-world vehicle kinematics and driver behavior.

In this thesis, first, the probabilistic position and velocity representation by the grid cells is discussed, as well as the models that propagate the probabilities. Then, the ego-movement compensation and how to use the representation for position tracking is illustrated. Next, the thesis deals with specific errors that emerge due to the discrete grid representation. The BHF is then further developed towards an Iterative Context Using Bayesian Histogram Filter (ICUBHF) approach by introducing an attractor-driven behavior model. This addition enables the anticipation of the behavior of the monitored vehicles' by comparing the likelihood of different behavior alternatives. Surveys of different comparison measures and of related research are provided as well. Finally, the ICUBHF is evaluated in different real-world settings.

The evaluation results confirm that the ICUBHF approach is able to track a vehicle and anticipate the behavior in a real-world intersection scenario. In conclusion, we outlined possible improvements necessary to create a productive ADAS application that deals with arbitrary real-world intersection scenarios. Such a system would allow an ADAS to work in complex urban scenarios in which it could track other vehicles and anticipate their behavior.

Contents

List of Figures

List of Algorithms

Glossary

behavior mode The mode is a behavior state that a certain agent has. The behavior state is categorized in modes by language or other categorization criteria. XIX, 135

behavior model The behavior model is an artificial construct in our brain or in software that gives rules as to how an agent will move. In contrast to the behavior mode, the model is defined by the algorithm. XIX, 135

dead-reckoning A method of estimating the current position by an extrapolation of the previous position using the known velocity and direction. 181

free intersection area The area of a street intersection in which the vehicle is not guided by lanes. 185

lane-constrained 1D representations within a multi-dimensional space that are often bound to *lanes*. 169

LIDAR A range sensing technology emitting laser beams and analyzing the reflected image. High end LIDARs like the well-known Velodyne LIDAR allow the creation of very accurate 360 degree three dimensional maps of the surroundings. 179

mode See behavior mode. 136

model See behavior model. 136

motor noise A term from the control engineering field for the noise that is added to the system during the prediction step. It results from the uncertainty in the system's locomotion. 79

multiple-model A tracking algorithm is a *multiple-model* tracking algorithm when it uses more than one sensor or prediction model. 137

Point estimators Estimators that output a single value, given a distribution as input. The output is a kind of best guess on a random variable. The Maximum a Posteriori (MAP) estimator or the Minimum Mean Square Error (MMSE) are well known examples of estimators. 138

Acronyms

ADAS Advanced Driver Assistance System(s). 1, A

AMM Autonomous Multiple Model. 138, 151

BHF Bayesian Histogram Filter. 2, 24, A

CAN Controller Area Network. 61, 174

CMM Cooperating Multiple-Model. 137

CPD Conditional Probability Distribution. 10

ECCF Ego-Centered-Coordinate-Frame. 60

HRI-EU Honda Research Institute Europe. D, 76, 117

ICUBHF Iterative Context Using Bayesian Histogram Filter. 115, A

IMU Inertial Measurement Unit. 61, 174

JPD Joint Probability Distribution. 8

JPT Joint Probability Table. 7

MAP Maximum a Posteriori. XX, 142

MMSE Minimum Mean Square Error. XX, 142

OSM Open Street Map. 174, 177

PMF Point-Mass Filter. 24

TORCS The Open Racing Car Simulator. 80

VSMM Variable-Structure Multiple-Model. 137

Chapter 1

Introduction

Mobility is highly in demand in today's society. However, commuting is time-consuming and inconvenient. According to [67], female commuters in Germany spend on average 51 minutes a day and male commuters 66 minutes a day traveling to and from the work place. These times are in the middle of the field of the worldwide comparison. Most of these hours are spent in private transport. Driving during rush hour is not fun, so it can be assumed that drivers have good reasons to take the car. Time spent sitting behind the steering wheel is not available for more useful or pleasant activities, such as reading a book or looking up a fact on the internet. In addition to the annoyance of wasting time while driving the same route each day the risk of having an accident is higher for inattentive or bored drivers.

At some time in the future there will be autonomous vehicles straight from the assembly belt. On the way to this overall goal, more and more Advanced Driver Assistance System(s) (ADAS) are currently being developed or are already in use, helping drivers avoid accidents. Some cars are also already driving partially autonomously in certain defined highway situations. In order to broaden the scope of situations the systems have to cope with, such as inexpensive and noisy sensors, bad driving and vision conditions like rain, snow and sun reflections, the systems need to be able to perform anticipatory driving and to react to unexpected behavior from other traffic participants.

In order to fulfill these tasks the ADAS needs an overview of the situation or in other words an internal representation of all relevant information on the traffic situation. However, agents (biological or technical ones such as vehicles with an ADAS) do not know the exact state of the real world. Instead they observe sensors. These sensors are probabilistically correlated with the real world. This work is about how this internal model can be built in a probabilistic way and how the knowledge of the real state can be optimized in a street environment in order to improve advanced driving assistant systems which need, by definition, knowledge about the world.

Most of today's ADAS systems, like adaptive cruise control, only work in certain situations in which a small defined set of rules can be applied. Such situations are mostly available in highway situations or in special cases like the autonomous parking system. In generic inner city situations the rule-set is much more complex, the search-tree for decision making is complex and the sensors rely on fewer assumptions. For example, while parking each eventuality can more or less be specified in advance (e.g. a pedestrian walks between our vehicle and another vehicle, or the other vehicle starts moving), while a usual inner-city drive is much less static, and uncertainties in the actions of others and in the surrounding are present. In such situations common state-of-the-art approaches are at their limit. Instead of approaching the problem from the state-of-the-art engineer perspective with deterministic or probabilistic representations limited to Gaussian distributions we decided to start from a biologically inspired representation, a grid of neurons or cells, which turn out to be identical with the so called Bayesian Histogram Filter (BHF).

ADAS systems need this internal representation in order to know about the current situation and in order to anticipate the behavior of vehicles in the surrounding area. Figure 1.1 shows the sketch of such an ADAS system, which can be applied to intersection scenarios, where an oncoming vehicle has the right-of-way. When the ego-driver is reacting in the form of braking, there is no need to intervene. But when the vehicle is not reacting, the ADAS needs more information on the other vehicle in form of a behavior anticipation. Is the other vehicle probably going to collide with the ego-vehicle? In order to determine this, behavior detection is used to extrapolate the internal estimate on the other vehicle state. In the case of a dangerous situation the ADAS will warn the inattentive driver or execute an emergency braking.

This thesis deals with two important modules of ADAS systems, the *vehicle position tracking*, which estimates the position of another vehicle based on noisy sensor detections, and the *behavior detection module*, which deduces the behavior of another vehicle.

An existing probabilistic representation, the Bayesian histogram filter (BHF), was adapted and optimized to on-board ego-centered vehicle tracking. An attractor approach was developed in order to generically model behavior in this representation, based on the surrounding context. A process detecting the right behavior by comparing different probabilistic representations was implemented and evaluated. Despite being fundamental research, the concept was evaluated with simulation data and tested on real-world data.

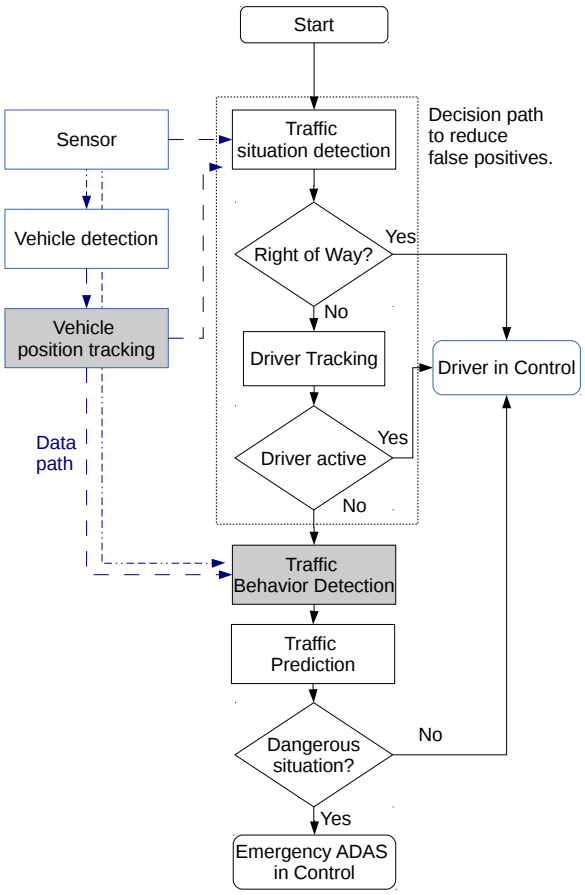


Figure 1.1: Notional decision graph of an ADAS system for accident avoidance in intersection scenes. It is a process of elimination in order to reduce false positives. That means that the driver keeps control of the vehicle unless all requested conditions are applicable. The data is flowing on the dashed blue path, beginning by the sensor. Black is the decision graph path. The gray modules are the modules on which this thesis is focused.

Part I

Background

Chapter 2

Mathematical Background

The vehicle tracking and behavior anticipation technique presented in this thesis are founded on the theoretical background of Bayesian networks and Bayesian filters. It is mandatory for the reader to have knowledge about these concepts. This chapters gives a brief introduction into the necessary foundation.

2.1 Bayesian Networks

A Bayesian network is an acyclic directed graph consisting of stochastic variables as nodes. The edges represent dependencies between the variables. Bayesian networks are therefore a compact representation to model probabilistic relationships. The following sections will equip the reader with all the fundamental knowledge needed to understand Bayesian filtering, which is based on this concept. For the more interested reader the tutorial [55] or the comprehensive introduction [16] is recommended.

The reader should already be familiar with the basics of probability theory and the idea of stochastic variables. In this thesis we denote a certain value from the set of all states of a stochastic variable with lower case letters, e.g. x. The stochastic variable itself is denoted with big letters, e.g. X, a vector of values with bold type, e.g. $\mathbf{x}$, and a vector of variables with $\mathbf{X}$.

2.1.1 Designing Bayesian Networks

Bayesian networks are graphical models illustrating dependencies between stochastic variables. Stochastic variables X can have a certain value x from a set of possible values and each value arises with a certain probability. As a simple example with an academic background, assume you are an university employee and you forgot the password for your desktop computer. The sysadmin could help you to reset the password but it is absolutely mandatory to wait for the right moment to ask him to do that. He is much more likely

to do it when he is in a good mood than a bad mood. In order to solve this problem we need to build a model world from our knowledge about sysadmins in general or even better about the individual sysadmin. The sysadmin's mood is denoted as stochastic variable M with two possible states $M = \{OK, Bad\}$ and set by observation the probability of $M = OK$ to $P(M = OK) = 0.2$. This knowledge alone does not help us in our task, since we only know that we wont get the password in 80% of all naive trials. At this time our sysadmin model consists of only one stochastic variable, but to reason a day when the sysadmin is probably in a good mood we need to augment our model with additional stochastic variables.

Next we denote a stochastic variable $F = \{won, lost\}$ for the event that the local football club has won or lost the last match. It is possible that an event like this will influence the sysadmin's mood, but since the sysadmin is not into football the probability $P(M|F)$ that the sysadmin is in a certain mood giving the football results F will not improve our knowledge. Therefore assuming independence, we can write $P(M|F) = P(M)$. The resulting Bayesian network consists of two separated subgraphs (c. f. Fig 2.1) indicating that M and F are independent variables. This shows two issues when designing Bayesian networks. First, stochastic variables must be chosen carefully, and second (expert) knowledge about the dependencies between them are needed. Failure in these tasks will produce misleading results when deriving the variable M.

A useful Bayesian network needs dependent variables. We denote the stochastic variable $P = \{pizza, nopizza\}$ for the event that there was a pizza event at the institute. When a pizza event P takes place, the sysadmin's mood will be better, since it is a well known fact that sysadmins love pizza. This means that $P(M|P) \neq P(M)$ (c. f. Fig 2.1).

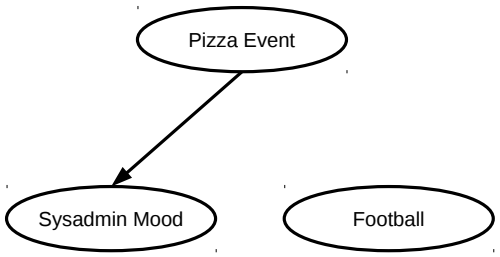


Figure 2.1: A Bayesian network consisting of two dependent nodes that model how a pizza event influences the sysadmin's mood and an independent subgraph containing the football event.

The above example represents the very basics of Bayesian networks. Usu-

ally Bayesian networks are used for more complex models. For simple models the reasoning can be done using a single Joint Probability Table (JPT), whose complexity scales with $O(2^n)$. The Bayesian network scales much better since it represents the joint probability distribution with small tables in each node instead of using a single big JPT. The small tables are the result of the exploitation of the independence. Fig. 2.2 from the example in the next section shows the tables called conditional probability tables (CPTs), on top of the Bayesian network nodes. It states the conditional probability in each node dependent from the values of the incoming variables. The usage of a CPT is only possible when all incoming variables are discrete variables. In other cases conditional continuous density distributions are necessary (CPD).

In this section Bayesian networks were introduced and the concept of dependent and independent variables were outlined in a practical example. The next sections cover reasoning in Bayesian networks and learning.

2.1.2 Reasoning in Bayesian Networks

The main goal when designing Bayesian networks is to compute the probability or probability distribution of a certain variable, given existing probabilistic knowledge like the network structure and the observable event variables. In our example we want to derive the probability of the sysadmin's mood in order to maximize the chance of getting a password reset.

This process is called (probabilistic) reasoning or inference in the literature. It is mandatory to introduce variable types used in Bayesian Networks discussing probabilistic reasoning. The stochastic variables can be divided into *evidence variables*, *hidden variables* and *query variables*. *Evidence variables* contain given knowledge about observed events. E.g. P can have a value of the set {pizza, no pizza}. *Query variables* are unknown variables which need to be inferred. In the example the sysadmin's mood will be inferred. *Hidden variables* are the left variables, for which neither knowledge of the value exist nor is the value is needed as output. The distinction between those last two variables is somewhat artificial for our application, as our example will show. Both, the hidden variables and query variables have an unknown value.

In the sysadmin example only query and evidence variables exist, and we want to know the probability that the sysadmin is in a good mood given the pizza event $P(M = OK|p = pizza)$ in order to gain a new password. At the moment the example is very basic and does not contain any hidden variables, but let's introduce some. Assume that a good mood of the sysadmin is a necessary but not a sufficient condition to gain our system password. This is the case because the sysadmin's password reset system is very buggy and does not work every day. The stochastic variable that the system will work or not is denoted with $S = \{work, \neg work\}$. This means that a new query

variable reset $R = \{true, false\}$ has to be introduced which depends on S and the sysadmin's mood M. The new Bayesian network can be seen in Fig. 2.2. For the sake of clarity the football variable was removed from the figure, which can be done with fully independent variables. The attentive reader will recognize that M has become a hidden variable, which together with the hidden variable S, influences the probability that our password will be reset R. Hidden variables may become query variables or vice versa if the question (aka. query) to the system is changed.

We now introduce the last piece of information about the state of the variables in order to reason about them. The Joint Probability Distribution (JPD) allows us to reason about $P(R|p)$. Details about the JPD can be found in [65]. As mentioned, the JPD represented by Bayesian networks is much more sparse than a decomposition using the multiplication rule, which contains non-necessary factors leading to a computation effort of $O(2^n)$. The JPD can be easily constructed by reading the Bayesian Network graph. Every variable in the graph is a factor, which is dependent from all its parent variables. The JPD of the sysadmin example is therefore: $P(P, M, S, R) = P(R|M, S)\, P(M|P)\, P(P)\, P(S)$

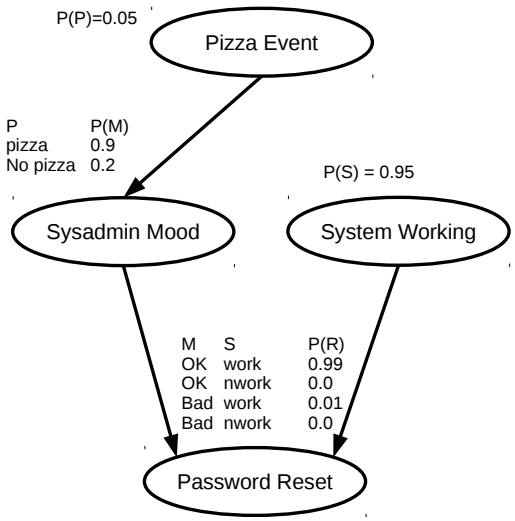


Figure 2.2: A Bayesian network modeling the dependency of four probabilistic variables. The Conditional Probability Table is noted for each variable. Note that the sum over a column does not need to be 1. In case of the the sysadmin's mood this means that a pizza event leads to a change of 0.9 that the sysadmin is in a good mood. In the case that there was no pizza, the sysadmin's mood is good with a 0.2 chance.

$$P(X|\mathbf{e}) = \alpha P(X, \mathbf{e}) = \alpha \sum_{y} P(X, \mathbf{e}, \mathbf{y}) \tag{2.1}$$

$$P(R|p) = \alpha P(R, p) = \alpha \sum_{m} \sum_{s} P(R, m, s, p) \tag{2.2}$$

In order to convert conditional probabilities into unconditional probabilities Equation 13.9 from [62] can be used, which is restated here as Eq. 2.1. α is a normalization term asserting that the probability over all possible states of X sums to 1, X the unknown query variable, $\mathbf{e}$ are the evidence variables and $\mathbf{y}$ are the unobserved hidden variables. When a variable is not known like the unobserved hidden variable it is possible to marginalize over all possible states of the unknown variable. The sum is therefore considering all possible combinations of the hidden variables. Using that equation we can infer Eq. 2.2 and inserting the specific JPD gives us an expression (Eq. 2.3) in which we can directly incorporate the system knowledge. Constant products can be moved out of the sums and thereby we receive Eq. 2.4. The query can be solved by entering the specific values into the equation.

$$P(r|p) = \alpha \sum_{m} \sum_{s} P(r|m, s)\, P(s|p)\, P(p)\, P(s) \tag{2.3}$$

$$P(r|p) = \alpha\, P(p) \sum_{s} P(s|p)\, P(s) \sum_{m} P(r|m, s) \tag{2.4}$$

Note that this method can also be used to derive the probability of a pizza event when the observer knows that there was a password reset. The pizza event becomes more likely when you know that a colleague was successful in getting his password reset while the probability of a pizza event gets smaller when you know that your colleague was frightened away by the sysadmin. This is called *explaining away* in Bayesian network terms. The knowledge over a certain variable in the network, like the password reset event, can be used to explain away reasons for this variable state. Bayesian networks can be used for inference in the direction of causality and against the direction of causality in the same way. In our case the calculation beginning with Eq. 2.2 has to be changed to derive the probability $P(P|r)$ that there was a pizza event P when there was a password reset r, instead of deriving $P(R|p)$. The JPD product in the resulting equation is identical, but the sum differ.

In the previous examples exclusively **discrete** stochastic variables were used, but it is also possible to model continuous variables. A Bayesian network can consist of both continuous and discrete variables. Continuous stochastic variables are often denoted by ovals in the literature while discrete stochastic variables are denoted by rectangles in graphs. This is demonstrated with an basic example from the automotive domain. The system consists of two stochastic variables. **Y** represents a sensor which detects

the position of other vehicles and $\mathbf{X}$ represents the real position of the other vehicle. The query variable will be the $\mathbf{X}$ of the other vehicle and the evidence variable will be the sensor's state $\mathbf{Y}$. (cf. Fig. 2.5). Each variable is two dimensional and therefore represented by a vector and denoted in bold type.

Both variables have not just two but an unlimited amount of possible states. Imagine that the sensor is mounted on a satellite and gives one particular sensed two dimensional UTM position $\mathbf{Y} = \{\mathbf{y}\} \ \wedge \ \mathbf{y} \in R \times R$ (the altitude is neglected here). This means that the possible state is a position somewhere on the earth's surface.

As in the discrete case expert knowledge is needed to derive the CPT, which gives the probability of all states of $\mathbf{Y}$ given a certain $\mathbf{x}$. In the continuous case the CPT becomes a Conditional Probability Distribution (CPD). For each $\mathbf{x}$ another probability distribution is assigned to $\mathbf{Y}$.

In Fig. 2.3 a CPD $P(\mathbf{Y}|\mathbf{X})$ for a certain $\mathbf{x}$ is seen. This special CPD will be later introduced as part of the *sensor model*. It gives the probability that the sensor will respond at a certain position $\mathbf{y}$ given the true vehicle position $\mathbf{x}$. In a perfect world $\mathbf{x}$ would exactly match to one $\mathbf{y}$, but in reality sensor measurements are noisy. Therefore a vehicle at a certain position $\mathbf{x}$ will create a sensor feedback at a certain position in $\mathbf{Y}$ with a (usually given) sensor specific probability distribution.

Fig. 2.4 gives us the resulting distribution for $\mathbf{Y}$ given a certain $\mathbf{x}$.

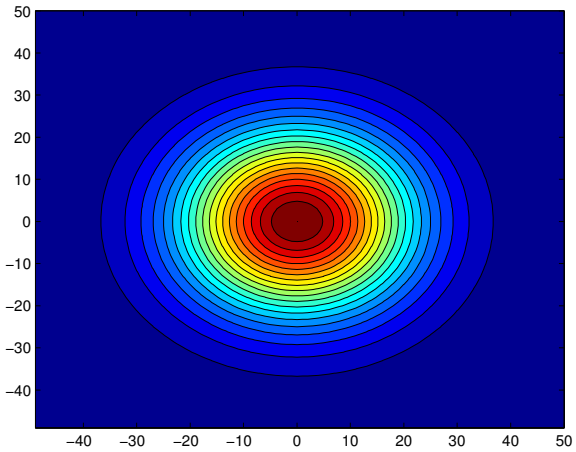


Figure 2.3: The CPD $P(Y|X)$ is visualized by showing it at an arbitrary x position $P(Y|x)$. The (Gaussian) function is represented by its equiprobability lines. Since the sensors characteristics are not dependent on x, the reader can imagine the CPD as an unvarying Gaussian shiftable over the whole coordinate system depending on x.

This is also a good opportunity to show a reasoning in the opposite direction of causality. $P(X|Y)$ is the query with the unknown query variable

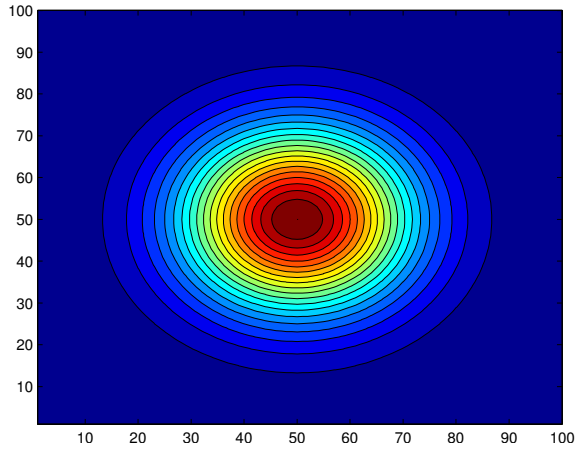


Figure 2.4: The sensor model $P(\mathbf{Y}|\mathbf{x} = (50, 50)^T)$ given the true state x=50. The plot shows the probability that the sensor will output a certain position given the state estimate $\mathbf{x} = (50, 50)^T$ for this specific sensor. (In other words the sensor output takes place in the same coordinate frame as the state. Generally the graph shows the distribution over Y)

x and the evidence variable y. The direction of reasoning is different of the direction of causality, since we want to know the position of the vehicle given the sensor reading, despite the fact that the vehicle position causes the sensor input. The JPD is generated by the Bayesian network interpretation: $P(X, Y) = P(Y|X)P(X)$ and the reasoning formula (like Eq. 2.3) $P(x|y) = \alpha P(y|x) \cdot P(x)$. Note that sometimes the PDF $P(y|x)$ is not given, but the inverse PDF $P(x|y)$ is. Then the above equation cannot be used for reasoning directly. The Bayesian rule must then be used first to substitute "wrongly directed" PDFs (that can be visualized as edges) in the graph. The *inverse sensor model* can be calculated by using the Bayesian rule on the sensor model resulting in $P(Y|X) = P(X|Y)P(Y)/P(X)$. In this example, the forward sensor model $P(y|x)$ is given (cf. Fig. 2.4) so that the Bayesian rule is not needed.

This Bayesian network introduction is based on [65]. A more comprehensive introduction in Bayesian networks can be found there. A short but outstanding tutorial is found in [54].

2.1.3 Learning Bayesian Networks

Bayesian networks can be created using existing world knowledge or the models can be learned. In the scope of this thesis, we used the assumption that the world knowledge is sufficiently known. Vehicle kinematics and sensor models are given by specification. However learning would also make sense when focusing on biological plausibility [33] or when adapting to drivers, country-specifics, or sensors (e.g. recalibration of a sensor).

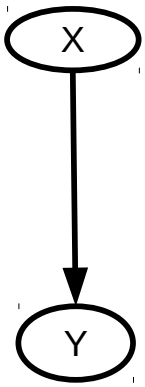


Figure 2.5: The Sensor model as a Bayesian network. The sensor output Y depends on the true state X.

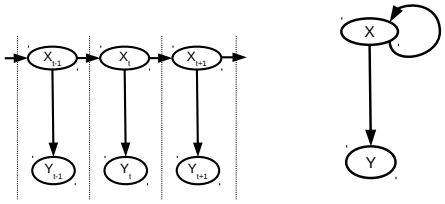
2.2 Bayesian Filters

The Bayesian Network examples in 2.1.2 are all static thus far. This means the reasoning process takes place within only one time step. *Bayesian filters* or *recursive Bayesian estimation* in contrast reason over several time steps. They are a subset of the set of Bayesian networks dividing the variables into time slices. The knowledge from the previous time-step is transferred into the next time slice using a probabilistic transition model. Recursive Bayesian estimation is the standard probabilistic concept used to accumulate sensor knowledge over time.

In the first subsection we will introduce the Hidden Markov Model (HMM) as a subclass of Bayesian Networks by introducing the concept of time slices and the Markov assumption. This theory leads to the Bayesian Filter concept. In the next subsection the most important implementations of the Bayesian filters are introduced. A comparison with the Bayesian occupancy filter concept is done. In practice there is much confusion between the two concepts. The last subsection deals with world representation and how to discretize the world in a useful way to generate reasonable computer models. The tradeoff between computing time and accuracy is explicated.

2.2.1 Bayesian Filter Concept

The Bayesian filter or recursive Bayesian estimator is a special kind of Bayesian network for a recursive estimation of the internal state of a sys-



(a) A HMM Model separated into time slices.

(b) The Bayesian Network with a loop.

Figure 2.6: Loops can be avoided by unfolding the Bayesian network into different time slices

tem. Using the nomenclature of Bayesian networks, the state of the query variable is reasoned over several time steps. It is useful to introduce *hidden Markov models (HMM)* before starting with Bayesian filtering.

It is necessary to use the Markov assumption when accumulating sensor data over several time steps. The Markov assumption states "that the current state $\mathbf{X}_t$ depends only on a finite fixed number k of previous states $\mathbf{X}_{t-k:t-1}$"[65]. All models used in this thesis are first-order Markov processes, meaning that the current state depends only on the previous state and the current evidence. The Markov assumption can be used by Bayesian modelers, when they assume that the (hidden) state variables (and the query variables) contain all information needed in order to derive the state in the next time slice. Equation 2.5 shows this first-order independence assumption in a probabilistic language.

$$P(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = P(\mathbf{X}_t|\mathbf{X}_{t-1}) \tag{2.5}$$

When looking back at the vehicle position example from the last subsection with the vehicle UTM coordinates provided by a satellite, the reader should notice that the position estimation could be improved by incorporating position measurements over several time steps. This is exactly the idea behind HMMs. We will first limit our view to static processes (e.g. the vehicle does not change its position) and then broaden to dynamic processes.

Figure 2.6a shows a HMM in which the vehicle position is measured at subsequent time slices. The HMM was simply created by using the Bayesian network in Fig. 2.5 as a *time slice*. A time slice contains all stochastic variables in the Bayesian network for a certain time step. Instead of creating a loop (cf. Fig. 2.6b) the network can be unfolded into an infinite amount of time slices.

The transitions from one time step to the next can be modeled with the transition model $P(X_t|X_{t-1})$, which emerges directly from the Markov assumption 2.5.

In a *static* process the hidden state X would not change over time. In each time step a new measurement is done. The process of adding new sensor information is called *filtering*. By adding more and more sensory information by subsequent filtering the estimate becomes more and more accurate. For $t \to \infty$ the distribution in X would converge to a dirac distribution. Since, however, vehicles are not static objects, the process has to be treated as dynamic. The transition function is then a distribution instead of a 1:1 matching (which is also a dirac distribution). The higher the uncertainty in the transition model, the more the information from the previous time step is blurred and the less accurate the estimate of X.

Recursive Bayesian estimation allows us to estimate the system state of *dynamic* processes. In comparison to static processes the real state of the system can change over time. In order to use sensor information from previous time steps the estimation has to be transformed from one time step to the next. This transformation of the estimate by using a transition model is called *prediction*.

The Recursive Bayesian estimation therefore consists of the filter step (introduced above for the static process) and the prediction step. The probabilistic formulas emerge by using Bayes' rule and the Markov assumption [62] on the Bayesian network query.

The Bayesian network query (Eq. 2.6) considers the complete network. The query wants to know the vector of internal state variables $\mathbf{X}_t$ at the current time t given a certain time series of evidence variable $\mathbf{y}_{1:t}$. The query is split (Eq. 2.6) and then the Bayes' rule is used (Eq. 2.7).

$$
\begin{aligned}
P(\mathbf{X}_t|\mathbf{y}_{1:t}) =& P(\mathbf{X}_t|\mathbf{y}_{1:t-1}, \mathbf{y}_t) && (2.6)\\
=& \alpha P(\mathbf{y}_t|\mathbf{X}_t, \mathbf{y}_{1:t-1}) P(\mathbf{X_t}|\mathbf{y}_{1:t-1}) && (2.7)\\
=& \alpha P(\mathbf{y}_t|\mathbf{X}_t) P(\mathbf{X_t}|\mathbf{y}_{1:t-1}) && (2.8)
\end{aligned}
$$

Equation 2.8 uses the Markov assumption on the sensor model. The current sensor state already includes the information of all past sensor inputs. The result is the *filter step* of the Bayesian estimation.

The prediction step is derived after the filter step. The prediction step emerges from a marginalization over all possible previous states $\mathbf{x}_{t-1}$ combined with another Markov assumption application.

$$
\begin{aligned}
P(\mathbf{X}_t|\mathbf{y}_{1:t}) =& \alpha P(\mathbf{y}_t|\mathbf{X}_t) \int_{\mathbf{x}_{t-1}} P(\mathbf{X_t}|\mathbf{x}_{t-1}, \mathbf{y}_{1:t-1}) P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) && (2.9)\\
=& \alpha P(\mathbf{y}_t|\mathbf{X}_t) \int_{\mathbf{x}_{t-1}} P(\mathbf{X_t}|\mathbf{x}_{t-1}) P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) && (2.10)
\end{aligned}
$$

With Eq. 2.10 the Bayesian estimation derivation is complete. Within the integral term of the equation, the estimation from the last time step is predicted into the future and the new sensor input is fed into the result of the integral. The result is the state estimate of the current time step. Note that the integral is used in continuous state space. A discrete process can be solved by using a sum term instead, which directly represents the *discrete Bayesian filter*. Some readers will also notice that the integral (or sum) term is a convolution, which blurs the internal state distribution from the current time step to the next time step.

The blurring effect takes the uncertainty into account. How will the state of a system change from one time step to the next? The less we know about the system's dynamic, the higher the blurring effect. The more we know about the system's dynamic, the steeper the transition model. More explanation about transition models can be found in Section II.

Looking at the example in this section the reader could ask why use Bayesian statistics instead of heuristic algorithms. Heuristic algorithms would improve the position knowledge, by applying for example, an arithmetic mean over the measurements. This would be equivalent to the Bayesian approach under two constraints: The process must be static and the noise must be fixed over time. If one of the above statements is not fulfilled a Bayesian filter with reasonable models will outperform every heuristic algorithm, since Bayesian filters can cope with changing noise and variable system states. In an automotive tracking task the sensor noise and the vehicle position is changing over time and therefore Bayesian filters are better suited than heuristic approaches.

System inputs

For the sake of completeness we will introduce a further stochastic variable $\mathbf{u}$. In literature $\mathbf{u}$ is often used to account for external non-noise influences on the system. In other words it is an input variable, which influences the dynamic of the system's state. When looking at the vehicle position example the vehicle position will change over time following the vehicle kinematics. However, the driver can make an input to the system in terms of steering or accelerating. This will affect the velocity and the direction of the vehicle in the next time step. In practice it is often irrelevant if an input variable $\mathbf{u}$ is introduced or if the system state is expanded so that u becomes part of $\mathbf{x}$. For example, the driver can be considered to be a deterministic but unknown part of the vehicle system to eliminate the extra variable $\mathbf{u}$. This also makes sense when the input is unknown. By introducing the variable $\mathbf{u}$ the current state $\mathbf{x_t}$ of the system depends on its previous state $\mathbf{x_{t-1}}$ and the current input $\mathbf{u_t}$, which increases the complexity of the transition model. It is then a CPD depending on the state dimensions and the control dimensions. The resulting Bayesian filter formula (Eq. 2.11) is depicted as

a Bayesian model in Fig. 2.7. For reasons of clarity and comprehensibility the input **u** will be sometimes omitted in formulas and figures without loss of generality in this thesis.

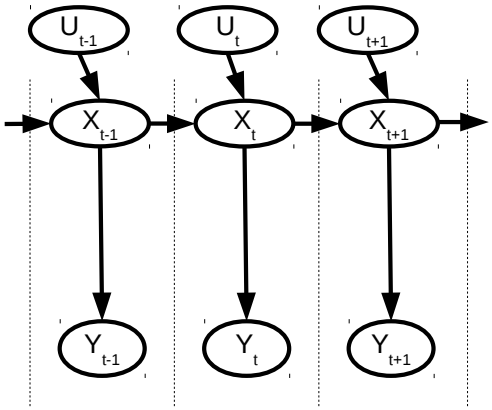


Figure 2.7: The HMM with system input U. Note that U and Y are observable variables, while X is the query variable.

$$P(\mathbf{X}_t|\mathbf{y}_{1:t}\mathbf{u}_{1:t}) = \alpha P(\mathbf{y}_t|\mathbf{X}_t) \int_{\mathbf{x}_{t-1}} P(\mathbf{X_t}|\mathbf{x}_{t-1}, \mathbf{u}_t) P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) \quad (2.11)$$

The Bayesian Filter as a Two Step Algorithm

The complete Bayesian filter mathematics are already derived by Eq. 2.10 or 2.11. As previously stated it contains a prediction and a filtering step. To be more explicit the equation is usually split up into these two steps. The result of the split is a two step iterative algorithm. For each state $\mathbf{x_t}$ in $\mathbf{X_t}$ the prior distribution $P(\mathbf{X_t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1})$ and the posterior distribution $P(\mathbf{X}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t})$ are calculated consecutively. By splitting Eq. 2.11 the equations 2.12 and 2.13 emerge.

$$P(\mathbf{X}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) = \int_{\mathbf{x}_{t-1}} P(\mathbf{X}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1}) \quad (2.12)$$

$$P(\mathbf{X}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}) = \alpha P(\mathbf{y}_t|\mathbf{X}_t) P(\mathbf{X}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) \quad (2.13)$$

The first equation is the prediction step resulting in the prior distribution, and the second equation is the filtering step (also known as *measurement update* in some literature) resulting in the posterior distribution. In

literature the *state estimate* is sometimes called the *(internal) belief* (meaning internal knowledge about the state), *state of knowledge* or *information state* [65]. The terms *posterior* and *prior* refer to the moment when the measurement is incorporated into the (internal) state estimate / belief. If not explicitly stated the term (internal) state estimate / belief denotes the posterior state distribution. The term *predicted state estimate* denotes the prior distribution.

In pseudocode the Bayesian filter can be written as Alg. 2.1 but in practice the algorithm cannot be directly implemented for continuous problems on a machine with a finite set of states.

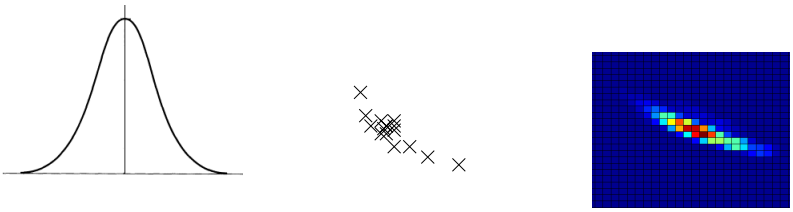
Algorithm 2.1: The Bayesian filtering algorithm. Adapted to our notation from [65].

input : $\mathbf{u}_t$, $\mathbf{y}_t$, $P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1})$
output : $P(\mathbf{x}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t})$
1 **forall** $\mathbf{x}_t$ **do**
  `// The prediction step`
2 $P(\mathbf{X}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) = \int_{\mathbf{x}_{t-1}} P(\mathbf{X}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) P(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1})$
  `// The filter step`
3 $P(\mathbf{X}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}) = \alpha P(\mathbf{y}_t|\mathbf{X}_t) P(\mathbf{X}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$
4 **end**
5 **return** $P(\mathbf{X}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t})$

The problem is that the integral in the prediction step is not solvable for arbitrary continuous distributions. Several approximate solutions exist. Some discretize the distribution in different ways, others approximate the distribution with a function for which the integral is solvable. However, even some approximative solutions have very high computational costs and are hardly computable efficiently. The state of the art Bayesian filter implementations are introduced in Section 2.2.2. In Section 2.2.4 the search space and its influences on the computational costs are highlighted.

2.2.2 Bayesian Filter Implementations

The integral in Eq. 2.12 is not solvable for continuous processes. Therefore an exact Bayesian filter solution is limited to discrete world problems. However, there are some ways to approximate the integral. The Kalman filter offers an analytical solution of the integral assuming Gaussian noise. Numerical approximations of the integrals, such as the particle filter or the Bayesian histogram filter are explained in the subsequent sections. Figure 2.8 outlines the differences in the representation of the compared Bayesian filter implementations. A brief and clear survey on Bayesian filtering techniques [35] is also recommended.



(a) The Kalman filter uses a Gaussian, fully represented by its expectation value μ and the covariance $\mathbf{P}$.

(b) The particle filter saves the particle coordinates as representation.

(c) The Bayesian Histogram filter assigns a probability value to each histogram bin.

Figure 2.8: The representations used by the Bayesian filter implementations.

Kalman Filter

The Kalman filter method is the result of the analytical solution of the integral in Eq. 2.12. The analytical solution is restricted to Gaussian sensor noise and linear system dynamics with Gaussian *system noise*. The term system noise describes the uncertainty in the state change, which can be caused by incomplete world knowledge and manufacturing tolerances. In practice the Kalman filter is surprisingly tolerant against violations of these restrictions. In engineering tasks, noise is often generously assumed to be Gaussian and linearity is perceived by local linear approximations of the process dynamics in enhanced versions of the Kalman filter. We will later see that the smaller the noise, the better this oversimplification works.

The Kalman filter can be derived from the Bayesian network equations (2.12 and 2.13). For a complete proof the reader is again referred to the standard literature [62, 65]. The overall concept of the derivation is stated here. As in the literature, we use the one dimensional example for the sake of simplicity. First a Gaussian prior distribution

$$P(x_{t-1}) := \alpha e^{\frac{-(x_0-\mu_0)^2}{2\sigma_0^2}}$$

is introduced with α as the normalization term to retrieve a probability density function from the Gaussian by setting the integral over the function to 1. Let μ_0 and σ_0^2 be the expectation value and the variance of the prior distribution. These values represent the initial knowledge about the system state. We further assume a linear Gaussian transition model

$$P(x_t|x_{t-1}, u_t) := \alpha e^{\frac{-(x_t-x_{t-1})^2}{2\sigma_x^2}}$$

with $\mathbf{u}_t = 0$ and a Gaussian sensor model distribution

$$P(y_t|x_t) := \alpha e^{\frac{-(y_t-x_t)^2}{2\sigma_y^2}}.$$

The prior distribution and the transition model are inserted into Eq. 2.12. The equation is again presented in Eq. 2.14 for the first time step of the Kalman iteration.

Inserting the explicit Gaussian distributions from above into the equation and combining the exponents and applying "the completing the square method" leads to the handy form in Eq. 2.15. The mathematical tricks used to get there can again be looked up in [62]. In short the argument of the Gaussian function is split up into a constant term, which becomes 1 after normalization with α and the residual term stays in the argument of the exponential function in 2.15.

$$P(x_t|y_{1:t-1}, u_{1:t}) = \int_{x_0} P(x_1|x_{t-1}, u_t)P(x_{t-1}|y_{1:t-1}, u_{1:t-1}) \tag{2.14}$$

$$= \int_{x_t} \alpha e^{-\frac{(x_1-\mu_{t-1})^2}{2(\sigma_{t-1}^2+\sigma_x^2)}} \tag{2.15}$$

The result of Eq. 2.15 is applied to Eq. 2.13 together with the Gaussian sensor model from above.

$$P(x_t|y_{1:t}, u_{1:t}) = \alpha P(y_t|x_t)P(x_t|y_{1:t-1}, u_{1:t}) \tag{2.16}$$

$$= \alpha \exp\left(\frac{1}{2}\frac{(x_t - \frac{(\sigma_{t-1}^2+\sigma_x^2)y_t+\sigma_y^2\mu_{t-1}}{\sigma_{t-1}^2+\sigma_x^2+\sigma_y^2})^2}{\frac{(\sigma_{t-1}^2+\sigma_x^2)\sigma_y^2}{\sigma_{t-1}^2+\sigma_x^2+\sigma_y^2}}\right) \tag{2.17}$$

The result after another row of mathematical tricks in Eq. 2.17 looks rather complicated. But by comparing the term with the Gaussian function description, it is clear that it is a Gaussian with expectation value

$$\mu_t = \frac{(\sigma_{t-1}^2 + \sigma_x^2)y_t + \sigma_y^2\mu_{t-1}}{\sigma_{t-1}^2 + \sigma_x^2 + \sigma_y^2}$$

and variance

$$\sigma_t^2 = \frac{(\sigma_{t-1}^2 + \sigma_x^2)\sigma_y^2}{\sigma_{t-1}^2 + \sigma_x^2 + \sigma_y^2}.$$

The derivation is complete since the posterior distribution with μ_t and σ_t^2 was derived by the prior distribution, using the sensor distribution and the

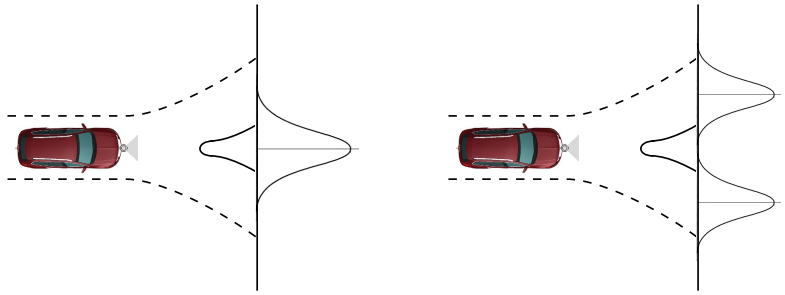
transition model. Thereby a complete iteration is completed. Furthermore the derivation outlines why the Kalman filter needs the basic assumption of Gaussian distributions and linearity. Without those assumptions the distributions in the algorithm would alter to non-Gaussian distributions in the multiplication during the filtering step or during the non-linear prediction. This would lead in the next iteration at the latest to the problem that the integral can no longer be solved analytically by the mathematical tricks used.

When using multiple dimensional space, the overall concept stays the same but the distributions become multivariate Gaussian distributions, the expectation value becomes a vector and the variance a covariance matrix.

Although in practice the Kalman filter achieves passable results when approximating distributions as Gaussian, enhanced versions of the Gaussian filter were researched and used widely in practice to cope with non-linear system dynamics. The *Extended Kalman Filter (EKF)* and the *Unscented Kalman Filter (UKF)* approximate non-linear system dynamics by different methods. While the EKF uses a Taylor approximation of the transition model for a local linearization around the current estimated state x_t the UKF uses sample points (called sigma points) in order to preserve the covariance. A short glimpse into these algorithms is given here, and for further details consult [65]. The linearization in EKF is done using the first term of the Taylor polynomial arround x_t. Thereby the transition model changes for each t. Using the first term of the Taylor polynomial is like drawing a line with the gradient of the system dynamics beginning at x_t. At diverging positions where small changes in x yield high changes in the transition model output the linearization leads to worse results. The variance in the internal estimate can be especially drastically underestimated. Nevertheless the EKF is often used in practice due to its simple implementation. In many cases it works, but the possible pitfalls are discussed later in Section 2.2.4.

The UKF was designed to avoid high errors in the covariance by using a stochastic linearization instead of the Taylor approximation. From the prior distribution some samples so-called *sigma points* are extracted. A sigma point is generated at the expectation value of the Gaussian prior PDF. In each dimension two further sigma points are determined, each at a certain distance in each direction of the dimension. The distance is arbitrarily chosen once in the beginning by introducing some bias parameters. Each sigma point is then individually fed into the transition function. A Gaussian is reconstructed from the mapped sigma points. The expectation value and the variance of the new Gaussian is determined by a weighted mean over the mapped sigma points. The weights are again determined by bias parameters.

However looking at Fig 2.9 it becomes clear that even the UKF cannot cope with highly non-linear systems. Therefore the Kalman filter is used too generously by engineers to cope with highly non-linear dynamics and multi-modal Gaussian distributions. A *Multiple-Hypothesis Kalman*



(a) **(b)**

Figure 2.9: A vehicle is approaching a Y-junction. (a) The Kalman filter will predict the position of the vehicle being on the traffic island. (b) The desired behavior of the Bayesian filter: The prediction should split into a multimodal distribution, accounting for the fact that the vehicle has to steer in order to avoid a collision with the traffic island. The figure is inspired by the bird and tree picture from p. 590 in [62].

Filter keeps track of multiple expected tracks and a weighted sum of the predictions is used to set the posterior state estimate. The weights of the Gaussian distributions are set by the likelihood that the sensed observation fits to the different predicted Gaussian estimates [62, 65]. Unfortunately the number of hypotheses grows exponentially with time. Some multiple-model approaches keep multiple hypotheses and therefore run multiple Kalman filters in parallel, but they have to keep track of the number of hypotheses by doing a fusion of several hypotheses. Fusing the posteriors to a low number of hypotheses will lead to high approximation errors in a highly non-linear system, but the fusion of some hypotheses is necessary in order to maintain a tractable number of hypotheses. Without fusion, a hypotheses tree has to be computed and the number of hypotheses may become incomputable in a finite time. Some approaches avoid keeping multiple hypotheses and use them only in the prediction step (cf. Fig. 2.10(a)). Either a weighting on the hypotheses can then be done or the most likeliest hypothesis will be used by a maximum selection [72]. A comprehensive overview of multiple-model tracking algorithms is given by [61] and a comparative study by [59]. Further information and readings on multi-model filters can be found in Part III of this thesis.

Another way to cope with multimodal distributions in a more "natural" way is the particle filter, which is discussed in the next section.

Particle Filter

Overview Particle filters use the Monte-Carlo-Method to sample probability distributions. So-called particles are generated over the entire state

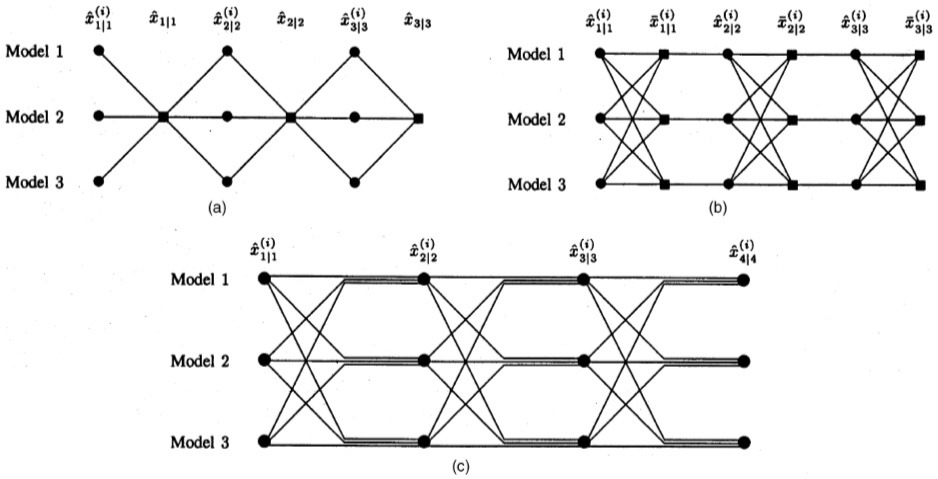


Figure 2.10: The figure shows different concepts how estimates from different filters can be fused in multi-model approaches. (Fig. 5 from [61])

space. In the initialization step, particles are created with a high probability in regions of the state space with high probability density. In regions with low probability density the probability for a particle generation is low or even near zero. The particle filter is a Bayesian filter implementation, so there is both a prediction and filter step. In the prediction step the particles are applied as input in the transition model and probabilistically distributed to the predicted states according to the model (note the difference to the sampling in the UKF, where the sigma points are directly mapped to *one* specific forward position). The result is the prior state estimate in a particle representation (cf. Fig. 2.8b).

In the filter step the new sensor input is fused to the prior estimate. Since it is difficult in the particle representation to perform a multiplication in a direct way a trick is used. Each particle in the prior distribution has a weight assigned to it. The weight is given by the sensor model. The higher the probability that the state at the particles position is true given the sensor input, the higher the assigned weight. Therefore the sensor model is only sampled at the particle's position. Highly multi-modal sensor models will therefore lead to problems. Next an *importance sampling* takes place. Particles with low weight (=importance) are erased. Particles with high weights will be cloned so that the number of particles stays constant over all time steps [65]. The result is the posterior distribution in particle representation. Then the iterative process repeats. The particle filter representation has several advantages, but also limitations. The limitations are discussed in the following paragraph.

Limitations The first limitation is not the fault of the particle filter itself, but the fault of successor algorithms. Most algorithms which use the tracking output as their input cannot cope with particle representations directly. A probability density function rather than a particle cloud is desired even for for visualization of the distribution. If the particle filter is used in discrete state spaces the density can be extracted by counting the particles in the bins. For continuous state spaces there are no hard and fast rules about how to create the density function from the particles. One idea is the Gaussian approximation, which generates a mean and variance by the particle samples generating a unimodal distribution. Another method is to artificially lay a histogram or density tree over the state space. Another idea uses each particle as center of a (Gaussian) kernel to rebuild the density function. Lastly, algorithms are computationally complex in comparison with the first one (which creates a unimodal distribution) [65]. On the other hand, the first one loses a lot of information from the internal representation to the output of the particle. The entire complex distribution is sampled to a Gaussian distribution losing all its characteristics except the mean and variance. In the end many current ADAS algorithms cannot cope with multimodal data at all, so that the simple Gaussian density output suffices for those systems in far too many cases (c. f. 2.2.4).

Discretization errors are another issue when dealing with low particle numbers per state space size in a dimension. This is relevant in high-dimensional problems, where the particles have to be distributed over a wide area in all dimensions. In that case the result of a particle filter may deviate from the unknown true Bayesian state distribution significantly after one or more time iterations. That means that a particle filter started several times from the same original state with exactly the same sensory inputs over time will produce different state distributions, depending on the seed of the random number generator used for particle forwarding. In particle filter terms this is called *sample variance*. The error increases with each iteration when no new sensor information is fed in or the sensor noise is very high. The variance of the distribution will be increasingly underestimated, whereby the estimate distribution approaches a single point. A strategy called *variance reduction* or *systematic resampling* is therefore used. The first strategy suspends the importance sampling in some time steps when the state changes slow down, or it even stops the integration of measurements when the state is known to be static for a while [65].

It is useful to think about the importance sampling as a roulette wheel approach to understand the *systematic resampling* or *low variance sampling*. The *systematic resampling* then chooses the particles systematically rather than picking them randomly from the set of particles.

A number of further discretization problems can occur if the number of particles is too low relative to the state space size. The *sampling bias* occurs in regions with a low number of particles. In the extreme case there is no

particle in the vicinity of the measurement y and only one particle a fair distance away. The importance weight of that particle is then much greater than all other weights. In the extreme case that there is only one particle in the entire state space the input y would be ignored completely [65].

The *particle deprivation problem* also arises when the number of particles is small relative to the state space size, but it can - with a smaller likelihood - also appear when the number of particles seems high enough. When running a particle filter long enough it may eventually appear that the importance sampling does not produce particles in an important region by chance. To avoid empty areas, often a small number of random particles are added to the complete state space. [65]

Despite all the limitations particle filters perform well and are used in practice in the automotive domain [44], [64], [30]. Particle filters are used less often than Kalman filters, but much more often than the approach used in this thesis.

Bayesian Histogram Filter

In this thesis a type of Bayesian Histogram Filter (BHF) was used. The BHF is a form of grid-filter. Chen [28] offers a comprehensive survey on Bayesian filtering and Bayesian grid-filters. Grid filters represent the probability density function by a finite set of samples, bins or regions. Another early form of the grid-filter is the Point-Mass Filter (PMF) also known as probability-grid filter.

Point-Mass Filter The PMF [45, 25] represents the estimate by probing the density functions at certain equidistant states with distance d. The points get weights (masses) according to the sensor input. The points are then sent through the transition function. The result is a transformed grid, which has to be resampled by interpolation to a new equidistant grid. The standard probability-mass-filter needs equidistant sample states, since the new posterior distribution is resampled from the density where the transformed grid states lay together with their probability mass. [18] Point mass filters are used for terrain navigation for aircrafts or underwater navigation, though the task can also be solved by particle filters with lower computational expenses and only slightly more inaccurate results [10, 19].

In contrast, in the Bayesian histogram filter approach the sample points are not transformed to other positions in the state space following the transition function, instead the sample points position is fixed and the probability inherent in the sample points is distributed to the other sample points according to the transition function. In other words: In BHFs there is a probability flow from one bin to the other, instead of sample points flowing, taking their probability with them, as is the case in the PMF or even in the particle filter.

Bayesian Histogram Filter The Bayesian Histogram filter as another grid-based filter samples the probability density function at N so-called grid cells (also known as bins or regions). The set of all grid cells cover the state space X. One specific grid cell k contains a certain probability mass $p_{k,t}$ (not to be confused with point-mass, cf. 2.2.2) which gives the probability that the true state $\mathbf{x}$ lies at the time step t within that region $\mathbf{x_k}$. The volume of the grid cell $|\mathbf{x}_{k,t}|$ may be fixed or vary over time. In either case, all grid cells have to be disjoint from each other.

We will see later, that it is useful to select a certain state $\hat{\mathbf{x}}_{k,t}$ within each grid cell and assign a special role to it. For example the density functions are usually probed only at one representative state in each grid cell. It is useful to choose the center of mass $\hat{\mathbf{x}}$ of the grid cell (cf. Eq. 2.25 [65]) in order to minimize errors. That special state (sometimes called node or sample point) $\hat{\mathbf{x}}_{k,t}$ is then the *representative* of the grid cells.

$$\hat{\mathbf{x}}_{k,t} = |\mathbf{x}_{k,t}|^{-1} \int_{\mathbf{x}_{k,t}} \mathbf{x}_t \, d\mathbf{x}_t \tag{2.18}$$

The density functions were probed at this representative $\hat{\mathbf{x}}_{k,t}$, which means that the sensory model $p(\mathbf{z}_t|\hat{\mathbf{x}}_{k,t})$ and the transition function $p(\hat{\mathbf{x}}_{k,t}|\hat{\mathbf{x}}_{i,t-1}, \mathbf{u}_t)$ are density function samples at $\hat{\mathbf{x}}$ instead of probabilities. Thus, integrating these density functions probes over the whole state space would not accumulate to 1.

$$\sum_k p(\hat{\mathbf{x}}_{k,t}|\hat{\mathbf{x}}_{i,t-1}, \mathbf{u}_t) \neq 1 \tag{2.19}$$

$$\sum_k p(\mathbf{z}_t|\hat{\mathbf{x}}_{k,t}) \neq 1 \tag{2.20}$$

The representative $\hat{\mathbf{x}}_{k,t}$ is also used to determine the system dynamics for the whole grid cell k. Meaning that each state within the borders of the grid cell x_k follows the state dynamics of the representative $\hat{\mathbf{x}}_{i,t}$. Of course this will lead to approximation errors, which are smaller for fine granularity grids and smooth non-linearities in the system dynamics. The sensor models are only sampled at the grid nodes $\hat{\mathbf{x}}_{k,t}$ (cf. Eq 2.20), which leads to discretization errors when the sensor characteristics are steep as well (Sec. 6.2 and 6.2).

When probing the probability density directly at the node position $\hat{\mathbf{x}}$ the Bayesian filtering equations look like Eqs. 2.21 and 2.22. These equations can be directly used when the grid nodes are equally spatially distributed. The normalization η sets the sum over all nodes to 1, ensuring that all computed values are probabilities after the filtering step.

$$p(\hat{\mathbf{x}}_{k,t}|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) = \sum_i p(\hat{\mathbf{x}}_{k,t}|\mathbf{u}_t, \hat{\mathbf{x}}_{i,t-1}) \cdot p(\hat{\mathbf{x}}_{i,t-1}|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t-1}) \quad (2.21)$$

$$p(\hat{\mathbf{x}}_{k,t}|\mathbf{z}_{1:t}, \mathbf{u}_{1:t}) = \eta p(\mathbf{z}_t|\hat{\mathbf{x}}_t) \cdot p(\hat{\mathbf{x}}_{k,t}|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) \quad (2.22)$$

An algorithmic notation of these formulas using the probability mass $p_{k,t}$ (and the predicted state estimate probability mass $p_{P_{k,t-1}}$) is given by Equation 2.23 and 2.24.

```
For all grid cells k do:
```

$$p_{P_{k,t-1}} = \sum_i p(\hat{\mathbf{x}}_{k,t}|\mathbf{u}_t, \hat{\mathbf{x}}_{i,t-1}) \cdot p_{i,t-1} \quad (2.23)$$

$$p_{k,t} = \eta p(\mathbf{z}_t|\hat{\mathbf{x}}_t) \cdot p_{P_{k,t-1}} \quad (2.24)$$

The distinction between probability density and mass is only necessary when the grid cell size is not equal for each grid cell. However, when the grid cell volume $|\mathbf{x}_{k,t}|$ differs, it is necessary to convert the densities into probability masses first (Eq. 2.25).

$$p_{k,t} = |\mathbf{x}_{k,t}| \cdot p(x_{k,t}) \quad (2.25)$$

In this thesis the Bayesian histogram filter is researched in a continuous world task. The discretization problems of the Bayesian histogram filter are comparable but differ in details from that of the particle filter. We discuss the discretization problems in Sec. 6.2 and the curse of dimensionality in Section 2.2.4. In chapters 3,4 and 5 the MDBHF (Merged Dimension Bayesian Histogram Filter) is described and in Sec. 7.2.2 the ICUBHF (Iterative Context Using BHF).

In the following section we explain another kind of grid filter, often confused with the BHF.

2.2.3 Obstacle Maps and the Bayesian Occupancy Filter

In the automotive domain the Bayesian Occupancy filter (BOF) is a recent field of research. The fact that the BOF as well as the BHF are grid-filters often leads to mix-ups and confusion. Therefore this section will explain the differences between these two kinds of grid filters. The main difference is the state space of the BOF. Instead of keeping track of the position of an object, the BOF is used to keep track of the individual positions. It states the probability that an individual position is occupied by an arbitrary object. Additionally, it assigns the velocity of a potential object at that position.

We will first discuss the general concept of static obstacle maps and then bridge to the BOF.

Static Obstacle Maps

The BOF consists of a grid over binary occupancy states $\{occupied, \neg occupied\}$ and additional variables for each grid cell. The concept of using binary occupancy states evolved from the *binary Bayes filter* approach. The binary Bayes filter is also a state-of-the-art approach used for *simultaneous localization and mapping (SLAM)* in robotics [12] and driving assistance [70]. Each grid cell consists of a probability giving the likelihood that the cell is occupied by a static or movable object.

The Bayesian Occupancy Filter

The original BOF formalism is complex and not easy to describe it in a short introduction. We explain the outline of BOF by using a simplified Bayesian Network based on the BOF version of [40] and [23]. They enhanced the BOF in order to use context information in the system dynamics. The notation was also changed to a form which conforms better to the Bayesian-filter notation.

The internal state $\mathbf{X}$ of the BOF consists of a vector for the occupancy of all N grid cells $\mathbf{O} = (O_1, O_2, .., O_N)$, where each O_i has two possible states $O_i = \{occupied, \neg occupied\}$. Each grid cell can also be assigned a number of non-binary variables, e.g. the velocity of the occupancy inside $\mathbf{V} = (V_1, V_2, .., V_N)$ with V_i as a two dimensional velocity vector discretized in cells per time-step. Consequently the internal state of the standard BOF is $\mathbf{X} = (\mathbf{O}, \mathbf{V})$.

The original standard BOF notation consists of a velocity distribution for each cell, but the velocity distribution was replaced by a single velocity value per field in later versions due to the high computation costs [27]. A sketch of the state vector X of the newer BOF version is shown in Fig. 2.11.

The state X can be further augmented by additional variables, in order to contain further information about the individual grid cell. For example the type of object occupying the cell can be estimated by a variable $G = \{pedestrian, vehicle\}$ [24].

In the transition step, the probability in the predecessor grid cells is forwarded into the current time step. The new velocity is calculated by the distance and time between the grid cells.

The difference between the BHF and the BOF is also becoming more clear when considering the dimensionality of the internal state variable in an abstract HMM view. In the BOF, the dimension of the occupancy state O for itself is already a N-dimensional variable with a binary state space in each dimension. In the end, the number of possible states is 2^N. A BHF observing a two dimensional position space contrast has a two dimensional position state variable, but each dimension has $\sqrt{N}$ possible states (the grid cells in each dimension). In other words the grid cells in the BOF are binary

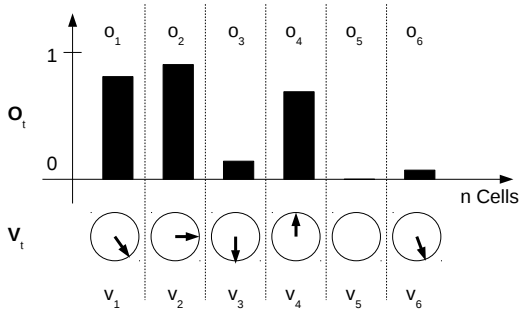


Figure 2.11: A one-dimensional outtake from a BOF. Each state consists of a probability of occupancy and a velocity. Velocities are represented as directions only in this figure. Occupancy probability of cell 5 is 0 and therefore no velocity is set for that cell by the BOF. The figure is simplified by neglecting the absolute velocity value.

state variables and the grid cells in the BHF represent states. The number of possible states in a two-dimensional Bayesian Histogram filter with N cells is $\sqrt{N} \cdot \sqrt{N}$. This does not mean that the BOF is more complex than the BHF. The number of possible states should not be confused with the complexity by the reader, since the BHF models a continuous distribution for each possible state, while the BOF models each cell independently as a binary state.

Due to that differences, the BHF and the BOF have different advantages for ADAS systems using their data as input. The problems of the BOF differ widely from the other Bayesian filter approaches. One advantage of the BOF is that the state is not object related. Each cell occupancy may belong to another object, and the BOF just collects sensory data without any association to objects. In multi-object environments the sensory evidence can be gathered and fused on a low level representation. This advantage is also a problem when it will be used for object tracking. Nearby grid cells are clustered by comparing distance, occupancy value, and velocity because objects have to be extracted from the BOF representation. The output is put into an object-related Bayesian histogram filter. In the literature the Kalman-filter was used together with a JPDA algorithm. [27]. This shows that the BOF is not a substitution of particle filters, Kalman filters or BHFs, but it may be used as a preprocessing algorithm to improve sensory input quality.

2.2.4 World Representation, Search-Space Reduction and Computational Costs

All the introduced Bayesian filter implementations can be applied to discretized or continuous processes in theory. However, all these methods struggle with the limited computational power and storage space in computer systems, when it comes to implementation of continuous systems. Bayesian networks can be used to model almost arbitrary complex model worlds with high-dimensional variables with continuous variable spaces, but a lot of complex concepts are doomed to always exist in the theoretical-world. When implementing the concepts for the real-world, Bayesian applications suffer not only from the well-known *curse of dimensionality*, but also from the continuous variable state space within each dimension and the need for a *search-space discretization*. The remaining finite state space should resample the most important characteristics of the original continuous state space. Which characteristics should be conserved depends highly on the application. In this section we first outline the two general search-space reduction methods, next how the search-space reduction was handled by real-world approaches in automotive tracking tasks and finally which state-space reductions were used in our MDBHF approach.

Search-Space Discretization

Each continuous stochastic variable stands for an arbitrarily formed *probability density function (PDF)*. For use in computers the PDFs have to be approximated by a number of bins, Gaussians, particles or other means. The basis behind each approximation is always the conversion from the unlimited complex state space to a finite number and range of variables representing the state space. The number of finite variables and their range is limited by computer memory and computation time requirements.

Curse of dimensionality

While the necessary variable range grows linearly with the size of the individual dimension, the number of dimensions increases the computational effort exponentially. The reason for the exponential growth of complexity is intuitively understandable when examining the discrete counterpart of the CPD, the CPT, in the Bayesian network. With each additional conditional dimension the number of entries in the CPT grows exponentially (cf. Sec. 2.1.1).

The key to computability is therefore to keep the probability density function in terms of the CPD small. This goal can be reached by exact or approximate dimension reducing approaches. An exact way is to find independent subspaces and divide them in independent variables. This is exactly done by the process of setting up the JPD of the Bayesian network. Recall

that independent stochastic variables are not connected by an edge in a well defined Bayesian network graph. As an approximate approach, it may also be useful or even necessary to assume independence between variables with existing but weak dependence. Note that the drawbacks of this harsh approximation may even be compensated by putting the freed resources into a more accurate state space discretization. The success of this approach may vary from case to case. In practice independence assumptions are often used somewhat generously. The reader has already learned about the Markov assumption, which is in most applications an approximation assuming that further unconsidered variables are independent to the observed system. As an extreme example, a sudden meteorite impact and its influence on the vehicle state may be ignored by all Bayesian vehicle tracking approaches, but stronger independence assumptions of all kinds also exist. Instead of ignoring dependencies between variables completely, a less generous approximation is the reduction of the variable range within a dimension. This is of course related to the search-space discretization. Non-essential dimensions can be represented in a coarser way. Instead of considering the whole distribution of a non-essential dimension, the dimension may be represented by the maximum or mean value of the distribution as a *representative value*. Note that the usage of a single representative value is a special case of a particle filter, in which only one particle instead of a number of particles per dimension is used. In [30] a combination of different representations is used, the velocity dimension is represented by particles and the position dimension by a grid.

A dimension reduction assumption often used in the automotive domain is the *flat world assumption*. The three dimensional position state space is mapped onto a plane two dimensional earth surface. The assumption is highly relevant when projecting camera images into a birds-eye-view. Applied to tracking the flat world assumption means that vehicles cannot jump and vehicles on crossover bridges above the ego-vehicle plane or in tunnels under the ego-vehicle plane are ignored. The reader notices that this assumption is not very strong, since most on-board sensor systems fail in detecting cars under the ground surface or on bridges above. Jumping vehicles play a minor role in everyday traffic, but the assumption gains relevance in hilly areas. Since we focus on on-board tracking, these cases can also be neglected, since current on-board sensors and detection algorithms are highly prone to slope changes and a lot of work is still necessary in that field of research.

Complexity of the PDF

Due to the curse of dimensionality, the complexity of the PDF has to be kept small when coping with high dimensional problems. Low dimensional problems allow complex PDF representations. The system engineer must

decide what is computable and he or she needs to check if the providable complexity fits to the complexity that is needed by the system. The needed complexity is discussed in this subsection. In the end it must be decided on a case-by-case basis, where and if the requirements overlap with the computational reality.

PDFs can exist as a single value, a Gaussian distribution or an arbitrary distribution. From the Gaussian representation to the arbitrary PDF representation the computational effort increases. The needed complexity increases with increasing uncertainty in the sensor or in the transition model.

When the sensors are very accurate, meaning that the sensor noise is very small, the usage of a probabilistic framework becomes exaggerated. We will nevertheless play this situation through. When the sensors become more and more accurate, the variance of the PDF becomes smaller and smaller, while the PDF converges into a dirac delta function $\delta(A)$ where A is the measurement y. With an exact measurement all information is available in the current time step so that a zero order Markov model can be assumed. The transition model becomes obsolete and the shape of the PDF remains a dirac forever.

For the sake of completeness it is mentioned here that the same effect would occur when a sensor with a given sensor noise greater than zero could deliver measurements in arbitrary small *update time intervals*. With fixed sensor noise even a simple mean computation would then suffice, under the assumption that the system cannot switch its states arbitrarily, but continuously. Such thought experiments help us understand better when more complex probabilistic representations become necessary. As already stated, the complexity of the PDF increases with increasing uncertainty in the sensor or in the transition model. The uncertainty in the transition model also increases with growing time intervals between the measurements and with smaller system stability. Some ADAS approaches use just the most probable value of the PDF as a single representative and ignore that the value may be not right in some cases (e.g. [20], the successor publication [21] generates different individual representatives by a Monte-Carlo method fed into the algorithm of [20]).

The most state-of-the-art approaches try to at least model the uncertainty with a Gaussian noise assumption, presumably without reflecting upon the validity of the assumption very much. Using the assumption, all PDFs are represented by Gaussians. As already stated in 2.2.2 the assumption works for smaller noise levels and linear or stable areas of the state space. Basically the EKF approach calculates the system dynamics at the expectation value of the state. A covariance is used to model the uncertainty of the estimation. The EKF approach gives an illusive security. The covariance is underestimated over time in non-linear unstable areas of the state space. Outliers in the PDF are ignored by EKFs and even by UKFs. It is possible that systems working with Gaussians will fail suddenly

in unexpected scenarios in rare situations. In general, such system behavior is always possible when the uncertainty in sensors or in the transition function is underestimated. This is based on the general tradeoff that all probabilistic systems are associated with. When too much noise is assumed, the system's state estimate is to vague and no actions can be derived by the system. For example, a system with the goal to avoid dangerous situations while driving may be an overcautious system. In extreme cases this system may not move at all since the world may be too unpredictable and comet impacts are possible at any time. However, a system assuming less noise is too self-confident in its estimation. Non-probabilistic systems are not better. They underestimate the noise of real systems in every case because all real sensors have a sensor noise greater than zero and deterministic systems assume zero noise.

Of course particle filters and grid filters also underlay the possibility of such system failures, but in practice the engineer's trust into Kalman filters is frequently exaggerated in comparison to the other filters. In non-linear systems with higher noise this trust is not reasonable and may lead to problems in autonomous vehicles. Ignoring this fact could lead to accidents and should therefore be considered by automotive manufacturers, who plan to build autonomous vehicles.

In addition to the sensor noise, the update time interval also plays a role, as is shown in the extreme case discussion above. When the prediction horizon increases, the transition models will become more and more non-linear. The benefit of using Bayesian filters capable of representing arbitrary PDFs increases with increasing prediction horizons. Looking back at the Y-junction situation in Fig 2.9 on page 21 this becomes very clear. Only with large update time intervals the prediction has to consider both movement alternatives simultaneously. In smaller prediction horizons it is sufficient that only one alternative is considered at one time step, since the measurement updates will iteratively correct the estimate to the right position while the prediction horizon only models small scale movements in the forward directions. This shows that only long term predictions with multi-modal distributions need a more sophisticated noise representation in this Y-situation.

All the discussed effects point out that the selection of the representation is driven by the assumptions of the quality and quantity of the sensor noise and the uncertainty in the system dynamics. They are the most fundamental questions any Bayesian filter user should consider. The next section will outline, which assumptions are made on a number of exemplary state-of-the-art approaches from the automotive domain.

Examples from the Automotive Domain

The consequences of the computational limitation on the state space representation can be discovered in existing automotive tracking approaches. Table 2.2.4 gives an overview of the different approaches.

Approach	*Search Space*	*Complexity of the PDF*
Barth [15, 14]	2D	Gaussian[1]
Althof [8]	1D[2]	Histogram
MDBHF	2D	Histogram

In [15, 14] a system to track the state of oncoming vehicles and predict their trajectory from the state was presented. A multiple model EKF is used. The multiple models are necessary since a Kalman-filter parametrized for tracking longitudinal movement reacts too slow in turning scenes according to the authors. The multiple models are therefore used not to represent complex PDFs exceeding the Gaussian state-of-the-art way, but to adapt the noise in the prediction model in an indirect way. The flat-world assumption is used and the state space is the two dimensional earth surface. The Kalman filter representation suffice since the application is limited to approaching objects. This causes the sensor noise to become smaller and smaller with decreasing distance to the approaching object. The final predictions are done by running the Kalman-filter in a loop that utilizes the prediction step and not the filter step, in later chapters we will introduce this as *prediction-only mode*. In the beginning predictions are rather weak but are iteratively getting better and better with the decreasing sensor noise.

In [9] or [8] stochastic reachable sets in non-linear systems are calculated with a Markov model. Collision probabilities of certain trajectory combinations are derived by the approach and two dimensional histograms are therefore laid on deterministic preselected trajectories. All trajectories are observed from a global non-on-board view. The deterministic trajectory is set by the street-pathway and the second dimension of the histogram is not independently modeled. Depending on the type of vehicle, segments on the right are weighted higher for bicycles, and for cars the middle is weighted higher. A probability flow over time from one histogram bin to the next histogram bin takes place only on the trajectory. All other probability values are derived by the probability on the trajectory. The reduction to one dimension is necessary, since the overlap of two or more reachable sets has to be calculated in order to receive the collision probability. Each trajectory of vehicle A has to be compared with each trajectory of vehicle B. The

[1] Barth is using multiple models, but not in order to represent the PDF in more detail. The coordinate frame is global.

[2] Althof is using a 2D-histogram, but the second dimension is generated generically. Therefore the visualizaton is in 2D, while the representation is 1D.

complexity of the algorithm rises with the number of histogram bins N with the number of trajectories M, and with the number of vehicles V leading to the sloppy O-Notation: $O((N \cdot M)^V)$. On the other hand, it is not guaranteed that these constraints are valid in all cases. For example, bicycles are often ridden anywhere on the street in urban areas. Lane changes cannot be modeled generically, which means that typically lane change trajectories have to be preset in advance, although the position where a lane change can take place is arbitrary. In wider intersection areas the real trajectories of the vehicles may have a high diversification.

Similar to the approaches above, our MDBHF approach, which is introduced in the next part of this thesis, tracks the position of an observed vehicle and derives the behavior by a model comparison technique. As with all discussed approaches, we use the flat-world-assumption and utilize a two dimensional histogram on the driving plane for the position estimate. At each position, the velocity dimension is represented by a mean value. This mean value is a representative value of the entire velocity dimension. The merging of a dimension to a single value is the new feature of the MDBHF in comparison to traditional BHF approaches. This simplification of the state-space is necessary, since the probability distribution disperses with increasing prediction horizons. The computability in real-time then becomes intractable. The MDBHF has in comparison to the cited related work the highest complexity of the position state-space and allows therefore multi-modal estimate distributions and a complex prediction model with individual predictions for each cell position. Further novelties of the MDBHF approach and the additional implemented features like the ego-movement compensation and the improved integration technique are discussed in the subsequent chapters.

In this section we learned about issues with selecting the state space of Bayesian models. We mentioned the effects of the curse of dimensionality and explained how different approaches deal with the computational limitations. We now focus on the Bayesian histogram filter for the automotive tracking task and in later chapters we will also use further techniques to improve computation time in the histogram filter.

Part II

Vehicle Position Prediction and Tracking with the Bayesian Histogram Filter

In the first part of this thesis the theoretical background of the Bayesian filter was discussed. These foundations are now used to track and predict the position of vehicles in road environments. Bayesian filtering gives the mathematically optimal solution for tracking tasks. The Bayesian filter itself cannot be implemented exactly, but it needs to be approximated by the Kalman filter or numeric solutions. In this part of the thesis we explain and discuss all relevant steps for vehicle tracking with our BHF solutions, the MDBHF and the ICUBHF. First, relevant parameters for adjusting the world representation of the BHF are discussed. Second, the influence of the prediction model is outlined. Third, the vehicle tracking task is explained in detail and illustrated by a vehicle simulation experiment. In the last chapter we discuss the modeling of the driving behavior.

Chapter 3

World Representation of the MDBHF

As discussed in section 2.2.4 the unlimited state-space of the real world has to be converted into a finite number of variables with high resolution representing this state space $\mathbf{X}$. The MDBHF and the common BHF approach achieve this by using grid cells $\mathbf{x_k}$ as introduced in section 2.2.2. The following section deals with the placement of the grid cells and the granularity of the grid cell distribution. Next, we evaluate how the PDF can be sampled and the influences on the position estimation quality. The last section covers the complexity of the representation of the velocity dimension.

3.1 The State-Space Domain and the Grid Cell Representation

The right choice of the state-space for a tracking task is highly task-dependent. Different kinds of state-spaces may fit even when in the field of vehicle tracking.

3.1.1 State-Space Domain

When localizing a vehicle with global sensors like the introductory example in section 2.1.2 the UTM position X is delivered by a satellite triangulation. When no further constraining assumption on the location of the vehicle can be given, the state space of the BHF has to cover the whole earth surface.

For the task of using on-board sensors of an ego-vehicle and tracking the position of the observed vehicle relative to the ego-vehicle a much smaller state-space needs to be covered. On the input side all sensors have a limited range and all state-of-the-art on-board sensors are limited to a range below approximately 100 m. And on the output side ADAS systems rarely need wider ranges for deriving driving actions. Evasive maneuvers in the case of

wrong-way drivers may constitute an exception here, but will be neglected due the lack of sensor and object detection capabilities. For the usual sensor setting it is also eligible to use a smaller state space in the lateral direction. The flat-world assumption is used for the vertical direction since vehicles mostly stay on the ground.

The state space of the real state is specified by $\mathbf{Z} = (Z_0 \ Z_1 \ Z_2)^T$, where z_0 is a certain position difference to the side, z_1 towards the vehicles nose and z_2 the height dimension (c.f. Fig. 3.1). The domain of all these probabilistic variables is $Dom(Z_i) =]-\infty, +\infty[$ when ignoring the curvature of the earth and the periodicity. The domain of the estimated state $x = (l_0 \ l_1 \ l_2 \ ||\mathbf{v}|| \ \omega(\mathbf{v}))^T \in \mathbf{X}$ is $[-l_{min}, +l_{max}]$ for the location variables l_0 and l_1. The height is not represented ($l_2 = 0$). The domain of the absolute vehicle value is thereby $Dom(||\mathbf{v}||) =]-\infty, +\infty[$ and the Domain for the velocity direction is $Dom(\omega(\mathbf{v})) =]-\Pi; +\Pi]$. All velocity and position variables are relative to the ego-vehicle's global position and velocity. To be even more accurate, we observe the position of the other vehicle's geometric center with respect to the ego-vehicle's center of the rear axis.

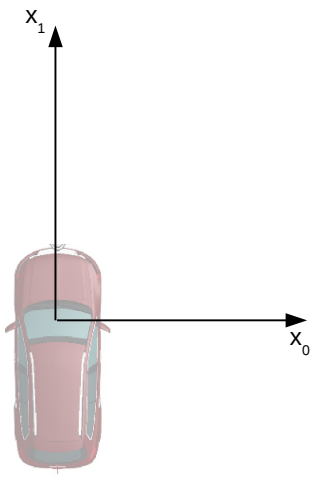


Figure 3.1: x_0 is the direction to the side, x_1 towards the vehicles nose and x_2 the height dimension.

As discussed above, the discrepancy between the modeled state space and the real state space can be neglected in our task in most cases. But there are some special cases that may produce problems. These problems and the ways to address them are discussed in the next sections.

3.1.2 The Boundary Condition Problem

The consequence of a finite state space is that boundaries have to exist. The world beyond the boundaries is not modeled by the MDBHF. But what happens when the observed object leaves the covered space or when parts of the PDF already lay beyond the boundary?

The following situations are possible:

- In the filter step, sampling errors occur when using a Gaussian Sensor Model. Since Gaussians assign probability values to infinite positions in all directions, there are always probabilities outside the covered space.
- In the prediction step, sampling errors occur when the transition function is Gaussian-like. When a vehicle is leaving the covered space the probabilities will gather at the boundary, unable to leave the covered space (c.f. Fig. 6.7). We will refer to this as *boundary jam*.

From other domains, such as splines or cellular automaton theory, such problems are well known and strategies to cope with them are used. However, not all proposed solutions make sense in a probabilistic representation.

- Periodic boundary conditions (leads to a toroidal world shape). Probabilities leaving the covered space will reenter on the opposing side.
- Ignore boundary problem. Do not use special rules at the boundaries.
- Persistent border values. Values at the border keep their value.
- Natural boundary condition. Set probabilities in the boundary to zero to absorb probabilities there.

The periodic boundary condition is clearly not suitable for our task. When ignoring the boundary problem the problems stated above will occur. The persistent boundary values method might not be the worst solution, but it will lead to the problem that the probability in the center will become smaller and smaller, while the probability accumulates in the boundary. When ignoring the probability values in the boundary during the normalization and during the output, that problem can be avoided. This is exactly the behavior of the natural boundary condition. Probability values in the boundary are treated as zero by ignoring them during normalization and during the prediction step. This is equivalent to an absorption of the probabilities in the boundary cells.

To ensure better and faster absorption, it is appropriate to use a boundary consisting of more than one row of boundary cells. An unlimited number of boundary cells would eliminate the errors, as would an unlimited covered area. In practice this is not possible, so the designer of a BHF should work

with a reasonable number of border node rows combined with a generously chosen covered area size. In our experiments three rows of boundary cells have proved to provide a good trade-off between the number of additional "useless" nodes and accuracy. Nonetheless, even with the absorption strategy of the natural boundary condition and a reasonable number of boundary rows the error will not be zero.

3.2 The Vehicle Position Represented by Grid Cells

The grid cells are the fundamental concept of the BHF. In section 2.2.2 we saw how the grid cells are used to represent the PDF. The grid cells position determines where the PDF is sampled, and the grid cell size determines the sampling accuracy.

We will tighten the notation here and subsequently explore the effects of different parameter choices affecting the cell position and grid cell sizes. A BHF consists of a set of N grid cells $\mathbf{x_k}$ with the index $k \in \{1..N\}$. Generally, a *grid cell* $\mathbf{x_k}$ samples the PDF at a certain volume of the state space $|x_{k,t}|$ in a given time step t. In our vehicle tracking task, each cell x_k contains the probability mass $p_{k,t}$ giving the probability that an observed vehicle's position lies within the grid cell volume $|x_{k,t}|$. Note that grid cells also represent velocity values. The details of the velocity representation are discussed in Sec. 3.3.

The *grid cell volume* $|x_{k,t}|$ can be set fix for all cells when choosing an regular grid, but it is also allowed to set different grid cell volumes for each cell. It is then important to distinguish between probability mass and probability density. Grid cell sizes are compensated by converting probability densities to probability masses before propagating the probability value during the prediction step (c.f. Eq. 2.25).

The freedom to vary the grid sizes allows the BHF designer to arbitrary set the *cell position*s. Not only is a regular tessellation possible, but so are all kinds of non-regular Voronoi or Dirichlet tilings. Arbitrary mixes are also possible. For example, there could be a grid consisting of cells with a regular tessellation in front of our vehicle, with an additional cell on the moon representing the small probability that the vehicle in front of our vehicle departed to the moon in the next time step. Of course, this example is not relevant to the vehicle tracking application. However, another type of tessellation would make sense, such as small grid cell volumes in the near-field of the ego-vehicle and bigger grid cells in more distant regions of the covered domain. Two reasons are speaking in favor of such an approach: First, most ADAS applications will demand more exact data on vehicles in the near-field since a collision is more likely and imminent. Second, the information delivered by the on-board vehicle detection sensors in more distant regions is not as exact as in the closer regions. For example,

a camera that detects vehicles in the surrounding will not output the exact position of distant vehicles.

In our MDBHF approach the grid cells are arranged in a regular tessellation with squares. Hence, the grid cells k are ordered in rows and columns where l_{k_0} denominates the columns in the x_0 direction and l_{k1} denominates the rows in the x_1 direction. The representative $\hat{x}_{k,t}$ (c.f. Sec. 2.2.2) is set on the center of mass of the grid cell, which assumes constant density, in the geometric center of the square. The representative state of the grid node in the bottom left corner of the grid has the position (l_{min_0}, l_{min_1}) and the upper right has the position (l_{max_0}, l_{max_1}). This regular tessellation allows a homogeneous behavior of the grid in all distances and positions for evaluation purposes. A variation of the grid cells can be implemented when demands on computation time and accuracy are determined by the application. The position $(l_{k_0}, l_{k_1})^T$ of the k-th grid cell can be easily computed by Eq. 3.1, where DIV is the integer division and MOD is the modulo operator, through the regular tessellation in the MDBHF. The distance between the grid cells is set by d_0 in the horizontal direction, d_1 in the vertical direction and $d := d_0 = d_1$ due to the squared nature of the cells. The grid cell volume is $|x_{k,t}| = d^2$. The on-board sensor of the ego-vehicle is set to the origin of the coordinate system. An offset value $\text{offset}_{x_1} \geqq 0$ assures us that the boundary is shifted towards the area behind the ego-vehicle.

$$\begin{pmatrix} l_{k_0} \\ l_{k_1} \end{pmatrix} = \begin{pmatrix} ((k\ DIV (max_0 - min_0)) - (max_0 - min_0)\ DIV\ 2) \cdot d_0 \\ (k\ MOD (max_1 - min_1) \cdot d_1 - offset_{x_1} \end{pmatrix} \tag{3.1}$$

In the next section the influence of the grid cell volume $|x_{k,t}|$, and thereby the distance between two neighboring nodes d on the tracking quality, is assessed.

3.2.1 Grid Cell Size and its Influence on Tracking Precision

The MDBHF is a numeric approximation of the Bayesian filter for non-linear and non-Gaussian systems. The quality of the approximation depends on an appropriate choice of several parameters. An example of one of these parameters is the grid granularity $g = \frac{1}{d}$. This is the inverse of the distance between the grid nodes d and thereby depends on the shape and size of the integration areas (c.f. Sec. 3.2). For instance, a BHF with infinitely small integration areas would be an exact solution of the Bayesian filter, though the need for an infinite number of grid nodes makes this approach impossible to compute. This chapter will outline the impact of parameter changes to the tracking result.

The finer the granularity and the smaller d, the more accurate the sampling of the PDF. The right choice of the grid cell size and position depends

highly on the task. In a vehicle tracking task for ADAS the parameter d has to fulfill specific demands. The parameter needs to be small enough so that the MDBHF is able to:

1. Cover the vehicle dynamics.
2. Derive warnings or evasive maneuvers.
3. Avoid discretization errors.

Since the number of nodes needed to receive the same covered space rises quadratically with the granularity in a 2D-space (meaning in a sloppy-O-notation: N = $O(g^2)$), it is a hard task to set the granularity to values fulfilling this demands and cope with the computational power of current hardware. When looking back to the satellite example it becomes clear that it cannot be solved with a BHF with a d=0.5m resolution on the whole earth surface. The grid needs to be much coarser, with the following consequences. First, it would be impossible to cover the vehicle dynamics. Second, evasive maneuvers are not possible since the ADAS algorithm on the ego-vehicle does not have any accurate information with lane-level precision. When assuming that the sensor accuracy of the satellite triangulation would provide the same accuracy on the observed vehicle position as on-board sensors it is also obvious that the granularity of the BHF would be too coarse to represent the PDF - the whole distribution would collapse into one grid cell. Note that the velocity dimension (Sec. 3.3) is even more prone to discretization errors in the position space, as detailed in later chapters. Low velocities lead to high relative velocity errors.

With an increasing grid cell size d, the approximation errors in the BHF become increasingly large. We evaluated the effects of d on the tracking result in a simulated situation in Sec. 6.1.1. The reader may look at Fig 6.2 and the following figures now, which show the tracking performance in dependence of the grid cell size d. Corresponding to Sec. 2.2.2, where the particle filter limitations based on [65] are discussed, we will discuss the limitations of BHFs and MDBHFs analytically in Sec. 6.2.

The grid cell size d is determined by the number of grid cells. And the number of grid cells is limited by the computational costs. In the next section we discuss how to reduce the computational costs in order to increase the accuracy of the BHF or MDBHF.

3.2.2 Computational Costs due to High Node Numbers

The computation time rises quadratically with the number of cells $O(N^2)$. While the filter updates each node n of the N nodes with the new sensor input in linear time resulting in a linear computation time for the filter step, the prediction step has to compute the integral in the Bayesian filtering by a sum over all preceding cells in Eq. 2.23. The probability flow from each

cell to each other cell has to be determined ($O(N^2)$) in order to receive the prior estimate distribution.

In practice there are methods to reduce the computation time by reducing the number of cells that need to be updated in a certain time step. We can consider the method of using boundaries at the covered area as an already discussed method, which allows us to ignore states/positions outside of the covered area. Of course, this is a very strict method, dividing the state space into an important area and an unimportant and ignored rest. It is also possible to perform a gradual transition between more important and less important areas. As in the particle filter approach, the PDF is then sampled in a better resolution in areas with high importance. High importance may be the result of a high probability density as in the particle filter approach, or also set by the ADAS application demands. For example, the area next to the ego-vehicle is more important than distant areas. Instead of a regular tessellation, a *density tree* can be used to cover the area with a different sampling quality. In important regions the distance between the nodes d is set to $\frac{d}{2^i}$ with $i \in \mathbb{N}$, while in rather non-relevant areas the sampling size is reduced by $i \in \mathbb{Z}^-$. In many cases these density trees will be static representations with fix i over time. Otherwise, new cell objects have to be created during runtime and tree structures need to be updated.

In order to accelerate the computation we use two further methods that work in a dynamic manner: *selective updating* (c.f. [65]) and a technique we call *selective propagation*. *Selective updating* is used in the filter and in the prediction step to ignore cells with a propability mass $< p_{min}$. Since the filter step works in linear time, it is not worth applying selective updating in the filter step, but in the prediction step significant speedups can be achieved. In our experiments a speedup of the factor 10 was generated with $p_{min} = 0.0001$ when using a grid size of $N = 88000$ grid cells. The benefit varies strongly with the size of the grid, the grid cell distance and the model noise and can be much greater when the grid covers huge unimportant areas. In Eq. 2.23 on page 26 cells i are ignored when $p_{i,t-1} < p_{min}$. The errors produced are marginal and are corrected by the next sensor input in the filter step. It is only with extremely converging dynamics that this may cause a problem since the small neglected probabilities accumulate into a small number of nodes and again become important, instead of diverging further into smaller and smaller probabilities, and therefore into even more unimportant activity ranges.

Selective propagation is a further method to save computation time. The idea of selective propagation is that even cells with high activation $p_{i,t-1}$ propagate most probabilities to a relative small number of cells and only a small amount of probability mass to the other cells. For this, the probability flow is calculated for each cell i when Eq. 3.2 is fulfilled. The equation shows the probability flow from node i to node k from time step $t-1$ to t according to the transition model. More accurately, it is more of an activa-

tion flow instead of a probability flow, since the activation is not normalized to probabilities here. In the BHF, the normalization to probabilities takes place only after the filter step.

$$flow_{k,i} = \begin{cases} p(\hat{x}_{k,t}|u_{t-1}, \hat{x}_{i,t-1}) \cdot p_{i,t-1} & p(\hat{x}_{k,t}|u_{t-1}, \hat{x}_{i,t-1}) \geq p_{pruning} \\ 0 & otherwise \end{cases} \tag{3.2}$$

In order to avoid the $O(N^2)$ computation time to check the above condition for each target cell k, the monotonous and smooth gradient of the transition function can be exploited. We utilized the "snail-search" algorithm, which checks the Moore neighborhood around the expectation value of the transition function. The range of the Moore neighborhood is increased until the transition probability in all new cells is smaller than $p_{pruning}$. The computation of further flows emerging from cell i is skipped for this time step. Of course, this method only works with smooth and monotonous transition functions. In return it avoids a full inspection of the state space domain.

When dealing with problems near the computational limits, where a higher number of grid points for a more accurate sampling is not possible, a variation of the sampling may help. The next section deals with different sampling strategies used in order to increase accuracy by reducing the discretization error with a constant number of cells.

3.2.3 Integrating Probability Mass in a Grid Cell

In BHFs the PDF is sampled at the grid cells. Each grid cell is an integration interval. Errors in the integration occur through approximation errors due to large grid cell volumes or by inaccurate function approximation within the cells. The prediction step is especially prone to such function approximation or discretization errors produced by that sampling. This is because the error propagates into the especially error-prone velocity dimension.

In standard BHFs the PDF is probed in each cell k once at the representative state of the cell $\hat{\mathbf{x}}_k$ (c.f. Sec. 2.2.2). In the position dimension the representative is the cell's center of mass $\hat{\mathbf{x}}_k = (l_{k_0}, l_{k_1})$. This standard integration technique is called piecewise constant function approximation. We will refer to it as *constant piecewise integration*. Instead of using only one representative, each cell can also be probed at several points. We have chosen the corner points c_0,...,c_3 in Eq. 3.3 to do a more sophisticated integration (referred to as *piecewise linear integration*).

$$\begin{aligned} \mathbf{c_0} &:= (l_{k_0} - d_0, l_{k_1} - d_1) \\ \mathbf{c_1} &:= (l_{k_0} - d_0, l_{k_1} + d_1) \\ \mathbf{c_2} &:= (l_{k_0} + d_0, l_{k_1} - d_1) \\ \mathbf{c_3} &:= (l_{k_0} + d_0, l_{k_1} + d_1) \end{aligned} \tag{3.3}$$

The prediction equation in the MDBHF is therefore replacing Eq. 2.23 with Eq. 3.4 in the position space.

$$p_{P_{k,t-1}} = \sum_i p(c_0, c_1, c_2, c_3 | u_t, \hat{x}_{i,t-1}) \cdot p_{i,t-1} \tag{3.4}$$

$$\begin{aligned} p(c_0, c_1, c_2, c_3 | u_t, \hat{x}_{i,t-1}) = (&p(c_0 | u_t, \hat{x}_{i,t-1}) + p(c_1 | u_t, \hat{x}_{i,t-1}) \\ &+ p(c_2 | u_t, \hat{x}_{i,t-1}) + p(c_3 | u_t, \hat{x}_{i,t-1}))/4. \end{aligned} \tag{3.5}$$

The selection of the corner points as probing points has the benefit that the function value can be saved in a look-up list and be reused for the neighboring cell in the "snail-search" algorithm (c.f. 3.2.2). Only one additional row and column of probe points in comparison to the standard implementation are needed.

Equation 3.5 shows how the values at the corner points are combined in order to receive the probability value for the cell. The equation is the result of the solution of the integral over the volume below the integration area interpolated by two triangular plane shapes through the points c_0,c_1,c_2 respective c_0,c_3,c_2.

The additional probe points are only used for probing the transition model, since the velocity dimension is derived here by the start and end of the flow (c.f. Sec. 3.3). The velocity dimension is highly prone to errors (c.f. 6.2). Since the filter step is more robust, the sensor model is probed at the representatives in the center of mass as in the default setting.

The improvements achieved by the additional probe points are seen in the experimental vehicle tracking results in Fig. 6.4 in Sec. 6.1.2.

In the previous sections we introduced how vehicle positions are represented by grid cells in the BHF, we investigated the influence of the grid cell size on the tracking results and we introduced techniques to cope with the trade-off between computation time and tracking accuracy. The representation of the velocity dimension was mentioned but not explicitly stated. We deal with the velocity dimension in the next section.

3.3 Velocity dimension reduction

The BHF not only tracks the vehicle position by position sensor inputs, it also derives a velocity dimension in order to be able to use the vehicle dynamics in the vehicle tracking. Representing the PDF of the position dimension already requires high computational effort (c.f. Sec. 2.2.4). Fully representing the velocity dimension as a grid in addition to the position would increase the computation time to $O(N^4)$. The velocity dimension of the PDF is therefore reduced in the MDBHF (Merged Dimension BHF) to a single representative value.

3.3.1 Derive Velocity from Position Estimates

The MDBHF is designed to work with on-board sensors, which deliver the position of the detected object (e.g. a vehicle detection via camera, stereo-camera, lidar or radar, c.f. Sec 5.1). Using the velocity data from a velocity sensor device such as radar can improve the tracking but is not mandatory for the MDBHF. Instead, the velocity is derived by the position differences over the time within the probabilistic representation. In order to reduce discretization errors, the algorithm was slightly improved for the piecewise linear integration. However, first we will look at the solution for constant piecewise integration.

Velocity Derivation with Piecewise Constant Integration

The velocity is implicitly given by the probability flow in Eq. 3.2. The position estimate flows from one state (cell) to the other state (cell) within a time step ΔT. The probability flow from a cell i to k represents the probability that the vehicle's position is within cell i and that a vehicle at that position departs to cell k during the time step. In Bayesian network terms, the vehicle velocity is a hidden variable, which depends on the current estimated position and the previous estimated position.

The velocity $v_{k,i}$ (Eq. 3.6) is determined by the grid cell positions l_i and l_k.

$$\mathbf{v}_{k,i} = \frac{\mathbf{l}_k - \mathbf{l}_i}{\Delta T} \tag{3.6}$$

The number of all velocity flows that enter the grid cell k are then merged into a single representative velocity for the probability mass in cell k. A pure BHF solution has to model all possible flows (velocities) in the grid cell. The velocities assigned to a grid cell are the result of all incoming flows from all other grid cells. The computation time for this representation would be far too high, therefore the velocity PDF is reduced by averaging one representative per grid cell. This is the topic of the next subsection.

3.3.2 How to Keep the Important Information by Merging the Velocity Dimension.

The goal of averaging of all incoming velocities to a single representative is to keep as much relevant information in the representative velocity as possible. Therefore the averaging is done separately for the absolute value $||\mathbf{v}||$ (Eq. 3.8) and the direction $\omega(\mathbf{v})$ (Eq. 3.7) of the velocity, in order to keep the essence of the information. When doing a simple vector addition instead of the separate averaging, two opposing flows with high velocities

would be summarized to a zero velocity. The nature of separate averaging fits better for the problem.

$$\omega(\mathbf{v}_k)_t = atan2\left(\frac{\sum_{i=1}^{N} \mathbf{v}_{k,i} \cdot flow_{k,i,t-1}}{\sum_{i=1}^{N} flow_{k,i,t-1}}, x_1\right) \tag{3.7}$$

$$||\mathbf{v}_k|| = \frac{\sum_{i=1}^{N} ||\mathbf{v}_{k,i}|| \cdot flow_{k,i,t-1}}{\sum_{i=1}^{N} flow_{k,i,t-1}} \tag{3.8}$$

The resulting velocity in polar coordinates $(||\mathbf{v}_k||, \omega(\mathbf{v}_k))$ can be transferred again into Euclidean space (Eq. 3.9).

$$\mathbf{v}_k = (||\mathbf{v}_k|| \cdot \sin(\omega(\mathbf{v}_k), ||\mathbf{v}_k|| \cdot \cos(\omega(\mathbf{v}_k)))^T \tag{3.9}$$

Instead of the averaging strategy, a winner-takes-all method also is possible. The averaging method helps to smooth the resulting PDF and therefore reduce discretization errors, the winner takes all method enables us to keep trajectories of the most probable behavior. This is especially relevant in prediction only loops (c.f. Sec. 4.3).

$$i_{max} = argmax_{i \in \{1..N\}} \left(flow_{k,i,t-1}\right) \tag{3.10}$$

$$\omega(\mathbf{v}_k)_t = atan2\left(flow_{k,i_{max},t-1}, x_1\right) \tag{3.11}$$

$$||\mathbf{v}_k|| = ||\mathbf{v_{k,i_{max}}}|| \cdot flow_{k,i_{max},t-1} \tag{3.12}$$

When the winner-takes-all strategy is used, the Eq. 3.11 and 3.12 substitute for 3.7 and 3.8.

Velocity Derivation with Piecewise Linear Integration

The piecewise linear integration has been used to improve the sampling of the transition model. As an additional benefit, we can further decrease the discretization errors in the velocity dimensions. Velocities are usually determined by the difference between the start location and end location of a probability flow 3.6. The locations are limited to the center of mass of the grid cells.

The additional sampling points in the corners of the cell allow us to specify the end location in a more continuous way by averaging. Note that due to the Markov assumption this is of course not possible for the start locations without blowing up the state space representation.

With the transition function sampled at the corners of the cell we have an indication into which subarea of the cell most probability is flowing. Taking into account this additional information it is better to shift the end location used for the velocity estimate towards the corners with the higher flow instead of choosing the center of mass.

This is realized by a weighted sum over the corners (Eq. 3.13) with the resulting location $\hat{\mathbf{l}}_k$ inserted into the velocity estimate equation 3.14.

$$\begin{aligned}\hat{\mathbf{l}}_k =&((p(c_0|u_t, \hat{x}_{i,t-1}) \cdot \mathbf{c_0} + p(c_1|u_t, \hat{x}_{i,t-1}) \cdot \mathbf{c_1} \\ &+ p(c_2|u_t, \hat{x}_{i,t-1}) \cdot \mathbf{c_2} + p(c_3|u_t, \hat{x}_{i,t-1})) \cdot \mathbf{c_3}) \\ &/(4(p(c_0, c_1, c_2, c_3|u_t, \hat{x}_{i,t-1}))\end{aligned} \tag{3.13}$$

$$\mathbf{v}_{k,i} = \frac{\hat{\mathbf{l}}_k - \mathbf{l}_i}{\Delta T} \tag{3.14}$$

Our experimental evaluations in Sec. 6.1.2 show significant improvements due to a combination of both techniques (Fig. 6.4 and 6.5). The reasons for these improvements are the better sampling of the transition function for the position estimate and the improved velocity derivation.

3.3.3 Potential Errors due to the Velocity Representation

The merging of the velocity dimension to one representative is, of course, a strong approximation. Therefore, we can give example PDFs in which strong approximation errors will occur.

A worst case scenario is a position estimate which splits at a certain point (e.g. a vehicle at a Y-junction) and then comes in conflict by entering a crossing from opposing directions. The PDF in the velocity direction dimension will have two distinct peaks. On the one hand, the absolute velocity will be preserved at the clash of the probability peaks, but on the other hand the velocity direction dimension will collapse. The collapse symptoms will vary depending on the velocity merging technique used in Sec. 3.3.2. The averaging technique leads to probability flows which depart from the clash in an orthogonal direction. The winner-takes-all strategy partly preserves the direction. In other words, some flows will survive the clash, but with a uniform split at the Y-junction both clashing peaks in the PDF have the same amount of probability. Small numerical differences determine which peak wins.

Such behavior is not wanted when using a filtering algorithm. Luckily, such worst-case behavior is limited to worst-case scenarios. Most of the worst-case scenarios will not apply to our vehicle tracking task. Even the scenario above is only possible with very wide prediction intervals, since in most cases the filtering step will collapse the PDF into one peak long before a multi-modal distribution reaches a junction point again. However, in prediction-only loops such cases may occur.

But even without the filtering step the behavior of the velocity merging techniques is rather of good nature. In diverging situations the approximation errors are minor, and even in some converging situations, such as

merging Y-junctions, because clashing probability flows will in most cases not clash with exactly opposing velocities and mirrored probability masses. In that case they will act much more reasonably, as when two exactly opposing peaks in the distribution hit each other. Imagine the scenario above with a merging Y-junction instead of an orthogonal crossing. The direction estimate will, according to the merging technique average to the correct direction. Above all, when the probability masses of both peaks differ, the proposed merging techniques will succeed.

Considering the computation time savings, this is a behavior we gladly accept. After dealing with the representation of our filter approach (MDBHF), we will now focus on the anticipation of vehicle positions in the next chapter. In a separate error analysis (Chp. 6.2) we will experimentally evaluate errors caused by the representation.

Chapter 4

Position Prediction

Anticipating the future state and behavior of car drivers is a challenging task for the designers of autonomous vehicles and ADAS. Current state-of-the-art ADAS systems for luxury class vehicles are capable of driving autonomously in highway scenarios, combining distance-keeping and lane-keeping technology. This technology uses modern sensor equipment including cameras and radar. From an engineer's perspective this task is relatively easy to manage. Lane detection already works in highway scenarios and each radar echo in front of the ego-vehicle within a lane is highly likely a car, motorbike or truck following the route of the highway. The rule-set and number of possible surprising situations in highway scenes is relatively small in comparison with inner city scenarios. In the case of an unforeseen event the car hands control back to the human driver.

The current goal of the automotive manufacturers is to use existing low cost sensor systems without expensive LIDAR solutions for these systems, but these sensors are prone to errors and only strong assumptions allow their usage. Such assumptions are possible in highway scenarios, for example. It is known what lane markings look like and how lanes are curved. It is not necessary to interpret the radar sensor into object hypotheses. The vehicle does not know objects at all, but uses raw data (radar reflections) and keeps its distance from them. Some preprocessing and filtering is needed at the data level only. Such simple models of the real world are not sufficient in inner city scenarios. A nearly unlimited amount of possible situations with exceptions and unforeseen events exist in such scenarios. To cope with them, a vehicle needs some form of comprehension of objects and their movement properties. This comprehension, which humans know and apply, is enabled by better world representations. In this chapter we deal with the knowledge about the movement properties of vehicles. In Chapter 7 we will build on this knowledge and introduce a representation which enables the MDBHFs to incorporate knowledge about vehicle movement given hypothesized driver intentions and, vice versa, driver intentions (behavior) can be recognized by

observing the vehicle behavior over time. This approach is called ICUBHF (Iterative Context using BHF).

The fact that world knowledge (system dynamics) can be used in Bayesian filtering via transition models is the big advantage in comparison to averaging approaches. The transition model is used to transfer the estimate from the past to the present.In comparison, moving averages work like low-pass filters, and therefore the averaged state lags behind the ground-truth.

The transition model can predict the state into the present or the future by using probabilistic *prediction models* already introduced as *transition models* in previous chapters. The prediction can take place as part of a vehicle tracking algorithm, using Eq. 2.23 and Eq. 2.24 in turns in order to predict the previous information collected by the sensor measurements into the next time step to combine them with the latest sensor measurement. Furthermore the prediction can be used in a *prediction-only-mode* using Eq. 2.23 iteratively without the filtering equation. In this way the reachable area of the observed vehicle can be calculated probabilistically [3]. However, one statement should be kept in mind when talking about predictions: all predictions are based upon assumptions on the system dynamics. Or, in our case, the predictions are based on assumptions on the behavior of the driver-car system. Any deviations from the assumptions will result in worse results.

We therefore first specify reasonable assumptions for prediction models in the next section, before we continue to incorporate these prediction models into our MDBHF.

4.1 Prediction Models

The prediction model $p(\hat{\mathbf{x}}_{k,t}|\mathbf{u}_t, \hat{\mathbf{x}}_{i,t-1})$ transforms the estimate into the next time step. In other words, it provides the rules how the probability has to flow from one cell $\mathbf{x}_{i,t-1}$ in time step $t-1$ to the other cell $\mathbf{x}_{k,t}$ in time step t. These rules are determined by the system dynamics and the system inputs, or more specifically on the automobile dynamics and the behavior of the driver $\mathbf{u}_t$. This probability flow can be treated like a neuronal activation passing the information about the object position from one neuron (cell) to the other neuron. We will look at the probabilistic flow first and then illustrate how the prediction model can be formulated by the kinematic model. In the last subsection we will apply the prediction model to the (MD)BHF.

All this together leads to the implementation of the prediction step, which consists of three sub-steps for each cell. We will refer to them in the following subsections:

1. Sample prediction model at all neighboring cells.

2. Create departing flow.

3. Accumulate incoming probability flow to new probability mass.

4.1.1 Probability Flow

The equations for the probability flow 3.2 are restated here for the sake of clarity. The probability flow from node i to k is defined by Eq. 4.1 (step 2 of the prediction step) and, by summing the incoming flow in k, the prior estimate is determined. The prior estimate distribution (Eq. 2.23) can therefore be rewritten as a sum over the incoming probability flow (Eq. 4.2, step 3 of the prediction step). Please note that the prior probability receives the index $t-1$ since the information inherent in the prior PDF is based completely on the previous time step. But the reader should notice that the time it represents is already t, since the prior estimate is the predicted state estimate.

$$flow_{k,i,t-1} = p(\hat{x}_{k,t}|u_t, \hat{x}_{i,t-1}) \cdot p_{i,t-1} \tag{4.1}$$

$$p_{P_{k,t-1}} = \sum_i flow_{k,i,t-1} \tag{4.2}$$

The probabilistic flow from Eq. 4.1 gives the probability moving from one cell i to another cell k from one time step to the other. The flow should, of course, match the dynamics of how the states of the real system can change. For this, the vehicle kinematics need to be modeled first, which is the topic of the next subsection. The modeling has to be done once in order to sample the model as the first substep of the prediction step.

4.1.2 Modeling the Vehicle Kinematics

Human drivers intuitively know intuitively how vehicles move. Technical systems either have to learn the kinematic physics or need an implemented physical model to decide which vehicle trajectories are possible and which are not. As an example, every car driver knows that another car cannot depart from its position orthogonal to its wheel plane in normal driving situations. This kind of world knowledge is also used by tracking systems. Therefore, we have implemented a probabilistic kinematic model which is based on the bicycle model (Eq. 4.6) widely used in robotics [63, 65]. The bicycle model can be used to simplify the well-known Ackermann steering into a model using a single imaginative wheel on each vehicle axis (c.f. Fig 4.1).

The bicycle model assumes that the wheels remain vertically oriented to the ground and that each wheel touches the ground in a single point. In this case two constraints apply:

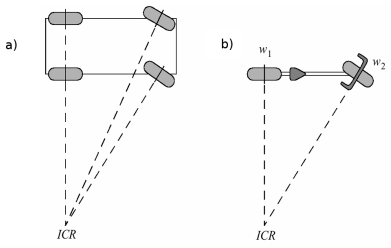
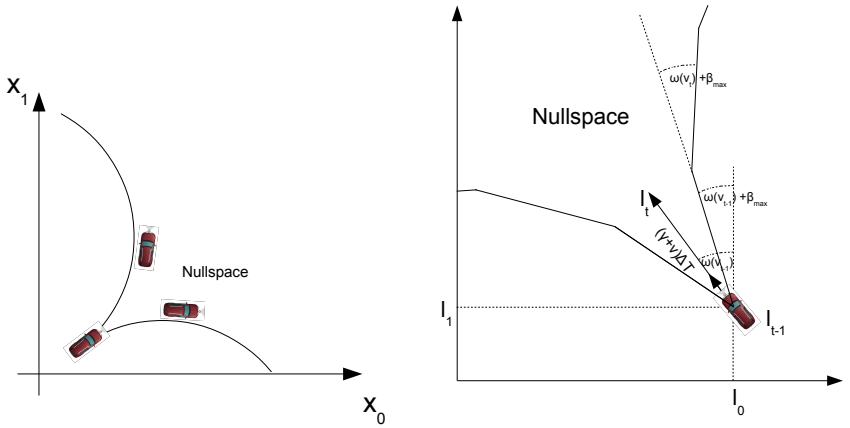


Figure 4.1: The bicycle model. [63]



(a) The kinematic model describes the reachable area (nullspace).

(b) The kinematic model using discrete simulation steps ΔT.

Figure 4.2: The reachable area in the continuous world and the discrete time approximation.

1. The rolling constraint, which states that the wheel must roll when motion takes place

2. The sliding constraint, which states that the wheel must not slide orthogonal to the wheel plane

While the first constraint is not particular relevant to our application, the second constraint limits the reachable area by the limited steering angle $-\beta_{max} < \beta < \beta_{max}$ in a limited time span ΔT. These constraints apply for each wheel and can be written as equations. Together they state a system of linear equations. The nullspace of this system of equations is the reachable area of the vehicle. That means the system of equations is solvable for certain vectors $\mathbf{x}_t$ of the state space. More details can be read in [3, 63]. A brief graphical description is seen in Fig. 4.2a and in the discrete time view in Fig. 4.2b.

4.1.3 The Probabilistic Driver Behavior

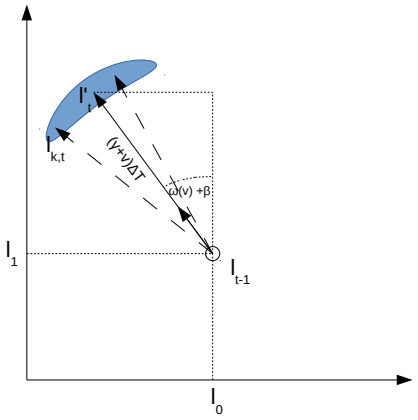
In the previous subsection the kinematic model was deterministically stated with strict limits for the driving behavior. In this subsection we introduce the driver behavior as a probabilistic variable in order to transform the deterministic kinematic model into a probabilistic kinematic model in the next subsection.

The probabilistic prediction model $p(\hat{\mathbf{x}}_{k,t}|\mathbf{u}_t, \hat{\mathbf{x}}_{i,t-1})$ depends on the system state and the system input. In other words, it depends on the vehicle state and the driver behavior.

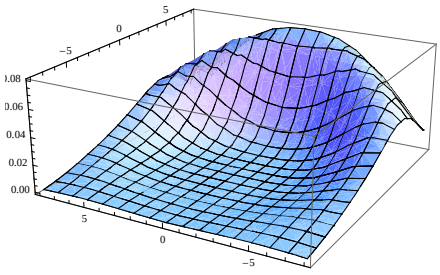
The state $\mathbf{x}_{t-1}$ is thereby a certain state from the state space domain $\mathbf{X}$ from the last time step. It contains the position $l_{0_t} l_{1_t}$, the absolute velocity $||\mathbf{v}_t||$ and direction $\omega(\mathbf{v}_t)$.

$$\mathbf{x}_t = \begin{pmatrix} l_{0_t} \\ l_{1_t} \\ ||\mathbf{v_t}|| \\ \omega(\mathbf{v}_t) \end{pmatrix} \tag{4.3}$$

The previous state fully defines the position expectation value of the state transition l'_t (Fig. 4.3a). A probabilistic model needs to account for the unknown driver behavior, and therefore needs to model deviations from the expectation value. Such deviating state transitions occur when the driver changes the velocity or steers.



(a) The transition model is derived from a probabilistic interpretation of the kinematic model.



(b) A two-dimensional Gaussian distribution in polar coordinates. E.g. the derived probabilistic prediction function in a 3D view.

Figure 4.3: The prediction function $p(\mathbf{x_t}|\mathbf{U}_t, 0)=$ in the continuous world and the discrete time approximation for a fix $\mathbf{x_{t-1}} = \mathbf{0}$ and a variable system input $\mathbf{u_t}$.

The behavior of the driver is defined by the probabilistic variable $\mathbf{u}_t$

(Eq. 4.4.) with the steering angle β and the acceleration γ.

$$\mathbf{u}_t = \begin{pmatrix} \beta \\ \gamma \end{pmatrix} \tag{4.4}$$

The domain of the acceleration is $Dom(\gamma) =]-\infty, +\infty[$ and the domain of the steering angle is also $Dom(\beta) =]-\infty, +\infty[$ instead of $]-\pi, +\pi[$. This allows us to use Gaussian distributions with unlimited domains in each dimension of the input. When using the circular boundaries instead, a von Mises distribution or a circular normal distribution [17] should be used for β.

The missing knowledge in the steering and acceleration behavior is modeled as a Gaussian distribution in our approach. Gaussian distributions $\phi(x; \mu, \sigma^2)$ are denoted with three parameters, as the probability value derived from the Gaussian distribution with expectation value μ and standard deviation σ sampled at value x. The steering is therefore denoted by $\beta := \phi(x, \mu_\beta = 0, \sigma_\beta^2)$ and the velocity change by $\gamma := \phi(x, \mu_\gamma = 0, \sigma_\gamma^2)$. The expectation values are set to zero, since a steering angle of 0 and an acceleration of 0 is assumed to be most probable. The resulting input space $\mathbf{U}$ is a two-dimensional Gaussian distribution in polar coordinates (Fig. 4.3b).

The usage of a Gaussian distribution has several advantages. The Gaussian distribution with its monotonous shape, allows the computational acceleration specified in Sec. 3.2.2. However, in the MDBHF the distributions can be chosen freely, and therefore arbitrary distributions are possible which reflect the behavior of the drivers in a better manner. The distributions may be learned dynamically or set by observations of real drivers.

We have chosen the Gaussian distribution as an a priori assumption for our models and it is possible to improve them later. The models reflect the fact that the steering wheel of a vehicle is, in most cases, in a neutral position and only in rare cases is in extreme positions. The Gaussian distribution is also a good choice since we do not know the maximum steering angle β_{max} of the individual observed vehicle. In addition, the assumptions that vehicles tend to keep their velocity is according to the vehicle physics and the traffic conditions with speed limitations. Deviations from this behavior, for example at traffic lights or curves, are discussed in the behavior prediction part (Part III) and in Chapter 7.

In the previous subsection we saw that the kinematic constraints dependent on the maximum steering angle, but the steering angle used does, in fact, depend on the driver. In this section we modeled driver's steering and acceleration behavior as a probabilistic variable $\mathbf{u}$. In the next section we exploit this model to define the probabilistic prediction model.

4.1.4 From the Kinematic Model to the Prediction model

By defining the distribution of the driver behavior in the last section, which accounts for the uncertainty in the directional and velocity changes of the observed vehicle, we can now define the probabilistic prediction model as a function. This function can then be sampled in the first sub-step of the prediction algorithm in order to generate the probability flow in the next step.

The probabilistic prediction model $p(\hat{\mathbf{x}}_{k,t}|\mathbf{u}_t, \hat{\mathbf{x}}_{i,t-1})$ for a certain system input $\mathbf{u_{t-1}}$ is derived by the prediction function f_P normalized over the prediction function for all probabilistic vehicle movement alternatives $\mathbf{u}_t = (\beta, \gamma)^T$ in Eq. 4.5. The normalization is necessary in order to achieve a probabilistic density function with function values between 0 and 1. Otherwise, the departing flow from the cell would not match the probability mass in the cell.

$$p(\mathbf{x}_t|\mathbf{u_t}, \mathbf{x}_{t-1}) := \frac{f_P(\mathbf{u}_t, \mathbf{x}_t, \mathbf{x}_{t-1})}{\sum_{\mathbf{x}_t^* \in \mathbf{X}_t} f_P(\mathbf{u}_t, \mathbf{x}_t^*, \mathbf{x}_{t-1})} \tag{4.5}$$

The state variable $\mathbf{x_{t-1}}$ and the input variable $\mathbf{u_t}$ together determine the probabilistic reachability of the states $\mathbf{x}$ as stated in Eq. 4.6. This equation can be directly derived by the kinematic constraints. Figure 4.3a helps us understand the effect.

$$\begin{pmatrix} l_{0_t} \\ l_{1_t} \\ ||\mathbf{v_t}|| \\ \omega(\mathbf{v}_t) \end{pmatrix} = \begin{pmatrix} l_{0_{t-1}} + cos(\beta + \omega(\mathbf{v}_{t-1})) \cdot (||\mathbf{v_{t-1}}|| + \gamma) \cdot \Delta T \\ l_{1_{t-1}} + sin(\beta + \omega(\mathbf{v}_{t-1})) \cdot (||\mathbf{v_{t-1}}|| + \gamma) \cdot \Delta T \\ ||\mathbf{v_{t-1}}|| + \gamma \\ \omega(\mathbf{v}_{t-1}) + \beta \end{pmatrix} \tag{4.6}$$

In contrast to the deterministic kinematic model, the prediction model uses β and γ as part of the probabilistic input variable $\mathbf{u}_t = (\beta, \gamma)^T$. The reader intuitively understands the correctness of the equation when substituting $\mathbf{u} := \mathbf{0}$. This is the state transition modeling the expectation value in the normal distributed $\mathbf{u}$. The expectation value of the transition will transfer the probability from $\mathbf{l}_{t-1}$ to $\mathbf{l}'_t$ as illustrated in Fig. 4.3a. In other words, a vehicle with a non-steering and non-accelerating driver will move from position $\mathbf{l}_{t-1}$ to $\mathbf{l}'_t$. The absolute velocity value and direction stay the same (third and fourth line of Eq. 4.6).

The probabilistic velocity change and steering angle $\mathbf{u}_t$ has the effect that the probability is not just flowing into the expectation value, but is also distributed to different subsequent states. The anticipated velocity and position in time step t of a vehicle with given position $\mathbf{l}_{t-1}$ and velocity $\mathbf{v}_{t-1}$ in the previous step is given by Eq. 4.6. The normalization in Eq. 4.5 causes the overall probability value propagated from the node i to all following nodes k in Eq. 4.1 is exactly the probability value in node i. In the last

substep of the prediction step all flowing probability can be summed up in node i again.

In the BHF approach Eq. 4.6 is sampled at the grid cells. By knowing the sample positions, and thereby the distances and angles between the starting cell and the sample points, we can postulate Eq. 4.7 as a generic prediction model for each grid cell i.

Remember that the sample positions are given, since the grid cells specify at which positions the prediction model should be sampled (c.f. Sec. 3.2.3).

$$\begin{aligned} f_P(\mathbf{v}_{i,t-1}, \mathbf{l}_k, \mathbf{l}_i) := \quad & \alpha(\omega(\mathbf{l}_k - \mathbf{l}_i); \omega(\mathbf{v}_{i,t-1}), \sigma_\beta^2) \cdot \\ & \gamma(||\mathbf{l}_k - \mathbf{l}_i||/\Delta T; ||\mathbf{v}_{i,t-1}||, \sigma_\gamma^2) \\ + \quad & \alpha(\omega(\mathbf{l}_k - \mathbf{l}_i) + \pi; \omega(\mathbf{v}_{i,t-1}), \sigma_\beta^2) \cdot \\ & \gamma(-||\mathbf{l}_k - \mathbf{l}_i||/\Delta T; ||\mathbf{v}_{i,t-1}||, \sigma_\gamma^2) \end{aligned} \tag{4.7}$$

The uncertainty due to speed changes is combined by a multiplication with the uncertainty due to the direction change. The expectation value can be set to the direction respective to the velocity in the previous time step. For a given velocity $\mathbf{v}$, each position difference between the two sample points $\mathbf{l_k} - \mathbf{l_i}$ is assigned a transition probability. The attentive reader will have noticed the second line of the equation. This line accounts for the possibility of driving backwards. The line becomes effective at very small velocities and/or large prediction horizons ΔT.

More information about prediction models can be found in [3] and [65]. [65] use a slightly different model. Instead of setting the new velocity direction $\omega(v_t)$ directly by the location difference $\omega(\mathbf{l}_k - \mathbf{l}_i)$, the direction is recomputed by rotating the velocity with $\omega(v_t) - \omega(v_{t-1})$. The underlying assumption is that the steering angle stays constant instead of preferring non steering. While this difference is important for the MDBHF, the effect becomes obsolete later when using the ICUBHF, since in the ICUBHF the steering angle is specified by the driving behavior (c.f. chapter 7).

In this section we have learned how the prediction step in the MDBHF works. We derived a probabilistic prediction model from knowledge about vehicle dynamics. We also mentioned that driver behavior is an additional element of uncertainty, which can be modeled as well. As general rule of thumb, it can be stated that the better the vehicle kinematic and driver behavior are modeled, the better the estimated states transitions will correspond with the real state transition an actual system will undergo. The influences of these model assumptions on the estimate are illustrated upon in the next section.

4.2 Conclusion on Model Assumptions and their Influence on the Estimate.

As human drivers do, also the technical systems have to rely on assumptions in order to compensate for uncertainties in the available sensors. Most of the time, the internal models are good enough, so that humans do not even notice that the brain relies on them. Neither, do we notice that what we see is the internal model world, our estimate of the reality rather than the reality itself. Let's be more concrete here and look at typical examples of driving situations most driver will be familiar with. In highway situations at night the velocities of very slow cars are often overestimated. The prediction model of the human driver assumes that cars drive faster than trucks. At the same time, at night our vision is less reliable about the velocity of other vehicles, and therefore the sensor input is trusted less than the prediction model. When our vehicle is approaching the car much faster from behind than expected, we are often immensely surprised. With the approaching vehicle, the sensor inputs gain in trust and the prediction model is revealed to be wrong. The fact that the velocity of trucks with the same speed is estimated correctly and without any surprise situation backs the theory that the reason is the wrong prediction model.

Another more trivial example is the underestimation of the acceleration of quickly-accelerating cars. When deciding to enter an intersection some drivers are confident that certain perceived vehicles will speed up with only very limited acceleration. When the assumption is proved wrong, the resulting situation causes surprise to the drivers of both cars. The driver of the outbound vehicle is surprised by the other vehicle approaching so fast, and the driver of the initially accelerating vehicle might be surprised that the other vehicle entered the intersection, since his prediction model assumed that other vehicles should give way in this situation. From the Bayesian perspective, it is clear that much annoyance in the traffic domain is the result of wrong internal models of traffic participants. In this context, 'wrong' simply means that the model does not match the real situation.

Since humans rely on models (based on assumptions), it is fair to allow that to machines. Machines need assumptions as well, since their sensors give them no complete world knowledge. If all sensors were accurate, comprehensive assumptions would not be needed. For example, if the machine knows what the other drivers want to do, assumptions on their behavior are no longer needed. In general, how many assumptions are necessary and how well the world needs to be modeled depends on the quality of the sensors. The MDBHF and ICUBHF have been developed for cheap state-of-the-art sensors with high uncertainty (c.f. 5.1). For a tracking system it is therefore necessary to have good models. In contrast, for accurate sensors or sensors with small update time steps ΔT a simple Gaussian distribution as

sensor model or transition model is often more than enough (c.f. 2.2.4). But in medium time range predictions, such prediction models would result in a smooth probability estimate. The PDF of the position estimate would easily cover the width of the entire road. ADAS algorithms need a more precise estimate as input. The variance of the estimate PDF depends on the models. The better the assumptions used in the models are, the more precise is the resulting estimate.

In the previous section we introduced the prediction model. The assumptions made, were that the driver steers with a normal distribution with σ_β and accelerates with a normal distribution σ_γ. Using small values for σ_β and σ_γ the PDF of the estimate will only fade slowly. The stronger the assumption, meaning the smaller the values for σ_β and σ_γ, the slower the information from the previous time steps is lost, at the cost that the model might be too strong and that the real state deviates from the strong assumption. This is exactly the same problem that the human driver has in the examples above.

On the other hand, when the assumptions are too weak, which implies high values for σ_β and σ_γ, the information from the past time steps is quickly lost. In extreme cases an automated vehicle is not allowed to enter an intersection from a give-way road at any time since an opposing vehicle may suddenly appear out of the dark at any point in time.

The parameters of the prediction model have to be chosen carefully by the designer and need to correspond with the grid cell distance: the grid cell distance needs to be small enough to sample the PDFs without producing many errors. A theoretical analysis is done in Chp. 6. As a final note, the thought experiments in this section were done for Gaussian models, but they easily generalize to other formed models.

Prediction models are used in Bayesian filter approaches to model the state transition. As we already know the Bayesian filter is a two step algorithm using in turns a prediction step and a filter step. In the next section we will discuss the possibility of using an iterative prediction without filter step.

4.3 The Pure Prediction Loop

The pure prediction loop is a method that can be used for iterative predictions over a long time interval ΔT instead of using a single prediction step with small ΔT as an alternative method. It can, for example, be used to project the current estimate into the future, in order to determine crashing probabilities and evasion maneuvers by a comparison of the own predicted trajectory with the predicted estimate of the other traffic participants. Both methods have advantages and disadvantages. A single prediction over a long time interval has the advantage of faster computability and a smaller amount

of discretization errors. One reason for this is the greater *distance to grid distance ratio* Λ (c.f. definition in 6.2.2). The other reason is that the sampling after the prediction step has to be applied only once (Sampling errors are discussed in Sec. 6.2.1). The advantage of the iterative prediction in a pure prediction loop is that prediction models can be used that are not initially built for long predictions. For example, as we will see in later chapters when inferring the lane course to improve the prediction, the behavior of the prediction functions differs locally. That local behavior might be ignored by long single prediction steps spanning a great distance.

The iterative pendant to a single long prediction step, the MDBHF in pure prediction loop, works like this: the sub-steps of the prediction steps are iteratively applied in Eq. 4.1 and 4.2. $P_{p_{k,t-1}}$ from Eq. 4.2 becomes the new $P_{i,t-1}$ in Eq. 4.1, when advancing one time step. The filter step, which is explained in 5.3 is omitted.

Due to the flattening probability distribution over time, pure prediction loops are costly to compute. This is because the probability masses climb over the selective updating limit in more and more cells from time step to time step. In practical considerations an online computation quickly becomes impossible with current sequential computation techniques.

4.4 Compensating the Velocity of the Ego-Vehicle

In our MDBHF and ICUBHF approaches we track the relative position between the observed vehicle and the ego-vehicle. All positions and movements are observed in the *ego-centered-coordinate-frame* (ECCF). The advantages and disadvantages of such reference frames are discussed in the first subsection. The next subsection discusses the additional implementation effort for compensating the ego-velocity most ECCFs need. In the subsequent subsection we give instructions on how to integrate the ego-compensation into the MDBHF. In the last section we discuss the consequences of a missing ego-movement compensation.

4.4.1 Advantages of Ego-Centered-Coordinate-Frames

The use of an Ego-Centered-Coordinate-Frame (ECCF) has benefits for vehicle tracking. First, the sensors are mounted on the ego-vehicle and therefore all measurements are relative to the ego-vehicle. On the other hand, when using a global coordinate system the ego-vehicle has to be located in the global reference frame first, in order to place the measurements in the global frame. A tracking algorithm working in the global reference system needs, at the very least, the ego-position and the ego-direction as additional inputs. But usual sensors like GPS do not accurately provide this data. thus, the ECCF is the more natural frame of reference to use. The ECCF

based tracking needs the ego-yaw rate (or steering angle with vehicle configuration) and the ego-velocity as input parameters. Both are available and rather accurate in automobiles. While the ego-velocity is available on the Controller Area Network (CAN)-Bus, the yaw rate is provided by an Inertial Measurement Unit (IMU). This means that on the input side, all necessary input variables are available in many currently developed vehicles.

On the output side, ECCF based tracking supplies a representation which can be used directly by ADAS algorithms to generate emergent action. These actions can be easily derived from the ego-centered estimate PDF by sensor-to-motor mappings. A well known example is the Braitenberg vehicle [22]. A Braitenberg vehicle maps the sensor inputs directly into the motor output in order to create an emergent light following or light avoiding behavior (Fig. 4.4). The behavior that a Braitenberg vehicle executes, would need complex algorithms in a representation using a global reference system. The usage of ECCF makes it possible to build comparable ADAS algorithms with intelligent behavior that emerges from the representation itself, instead of needing a pre-programmed behavior routine for each situation or special case. Thus, also for adding ADAS capabilities on top of the tracking algorithm, an ECCF is the more natural choice.

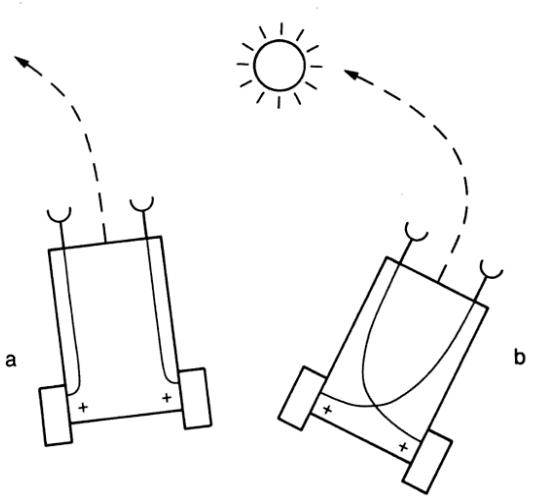


Figure 4.4: The right light sensor of the Braitenberg vehicle *a* gets a higher signal and speeds up the right wheel. The left light sensor of the Braitenberg vehicle *b* gets a higher signal and speeds up the right wheel. [22]

The disadvantage of ECCF is the higher implementation effort. The effort is higher, since an ECCF is not an inertial frame of reference, meaning that Newton's laws of motion cannot be used in the prediction models without considering the ego-motion. However, assuming that the ego-yaw rate is constant, it is a rotating reference frame since the ego-vehicle, like

every Ackerman vehicle, drives around an instantaneous center of rotation (ICR) defined by the wheel axis [63]. (An ego-yaw rate of 0 can be treated as a special case or, in order to avoid a special case, we disallow zero and set zero values to very small values - small enough to appear as zero for our application. We prefer to disallow zero and treat straight movement as a circle movement with infinite curve radius). With a rotating reference frame the influence of the ego-movement has to be compensated in each time step in order to transform the probability from one time step to the next time step. The theoretical background for the ego-compensation is the topic of the next subsection.

4.4.2 Rotating Reference Systems and their Implementation in the BHF

In a rotating reference frame all relative movements $\mathbf{v}_l$ consist of a velocity component caused by the global velocity (we denote it as absolute velocity) $\mathbf{v}_{abs}$ and a velocity component caused by the movement of the ego-vehicle $\mathbf{v}_{Ego}$ (cf. Eq. 4.8).

$$\mathbf{v}_{l,i} = \mathbf{v}_{abs,i} - \mathbf{v}_{Ego,i} \tag{4.8}$$

The ego-velocity vector depends on its position on the rotating plane. Therefore each cell has its specific ego-velocity vector. The ego-velocity can be denoted as a function of the cell position and the ICR-position, which is again defined by the yaw-rate or the steering angle and ego-vehicle velocity.

$\mathbf{v}_{abs}$ is the movement a stationary observer in a global frame of reference would see. $\mathbf{v}_l$ is the movement a moving observer would see. To picture what happens with positions and velocities observed by an observer on a non-linearly transferred reference frame you can imagine a transformed coordinate frame and a point with a fix velocity vector in the global reference system. The point is straight-moving in the global reference frame, but the movement will be bent in the transformed reference frame. Fig. 4.5 shows the general principle of how an observer moving on a circle will see a straight moving vehicle.

When using Bayesian filters, there are two general options about how to account for the ego-movement during the time transition:

1. Introduce another step, which rotates the representation after the prediction step
2. Consider the turning within the prediction function

The first option introduces a separate step, which is independent from the prediction step. The advantage is that only a simple rotation of the

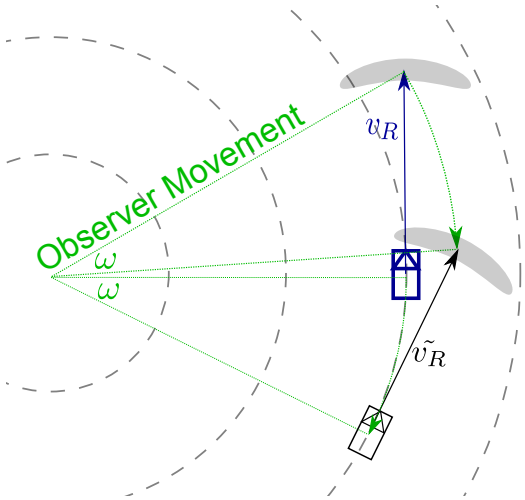


Figure 4.5: A moving observer sees a vehicle at the position of the blue vehicle, and in the next time step at the position of the black vehicle (rotated by the negative ego-yaw rate $-\omega$). During the transformation the vehicle is assumed to be stationary. The observed vehicle has a velocity $\mathbf{v}_R$ and points at the expectation value of the next time step. The velocity vector is also transformed by $-\omega$ to the new velocity $\tilde{\mathbf{v}_R}$. The new expectation value shows the predicted position of the observed vehicle in the next time step. The observer, who is by convention arbitrarily but fix positioned in the transferred location space and which is oriented parallel to the dashed circle, is seeing the movement of the observed vehicle as a circle movement illustrated as the green arrow. This means in the end, that in the eyes of a left turning observer objects are deviated on a circular trajectory to the right.

representation needs to be done, but the drawback is that another sampling is needed. In addition to the sampling in the grid cells at the end of the prediction step, the rotated estimate PDF has to be sampled again at the grid cells (cf. Fig. 4.6).

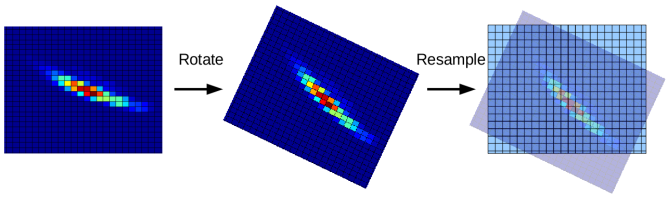
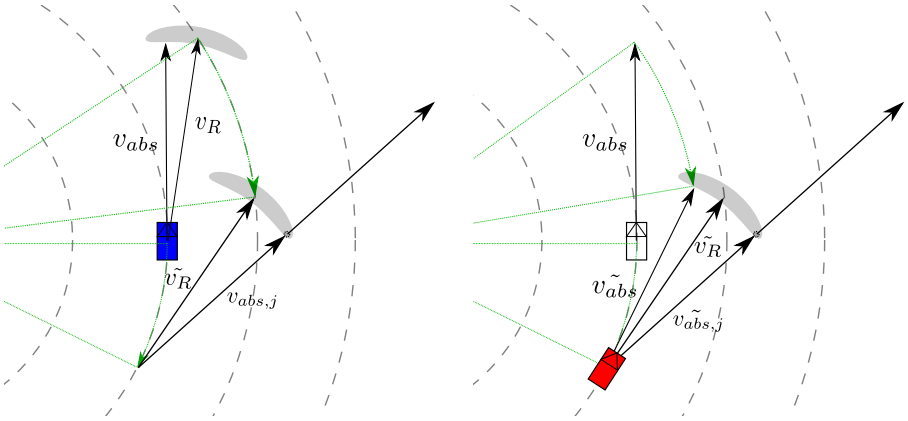


Figure 4.6: A simple rotation of a grid requires a resampling.

The second option is more complicated to implement. We decided to incorporate the turning within the prediction function and ignore the implementation overhead in order to avoid stronger discretization errors due to the second sampling. The concept of velocities in different reference systems needs to be implemented within the prediction step, while the first concept allows us to omit this by applying the rotation on the relative positions and velocity. The second concept needs the following velocities: The relative velocity $\mathbf{v}_l$, the absolute velocity $\mathbf{v}_{abs}$ and, as already discussed, the ego-velocity $\mathbf{v}_{Ego}$. Additionally, it needs the rotated velocities, which will be marked by a tilde, e.g. the rotated absolute velocity $\tilde{\mathbf{v}_{abs}}$. In later chapters we introduce the concept of attractors. The attractors based on context information also need to be rotated for use in the prediction function, depending on the method used. We mention them here in order to avoid a double explanation of the ego-velocity compensation. The rotated absolute velocity using attractors will be denoted by $\tilde{\mathbf{v}_R}$, the velocity proposed by the attractor as $\tilde{\mathbf{v}_A}$.

Note that for the sake of clarity we omitted the cell indices. Each grid cell has its own velocities, so the velocities usually have, as an additional index, i.e., the number of the cell i. Only the ground truth velocities used in other chapters do not need a cell index since there is only one true velocity.

There are at least two ways to incorporate the ego-velocity compensation in the prediction step. One method is to calculate the transition function in the coordinate system from the actual time step, depicted by the blue color in Fig. 4.5. The other method uses the coordinate frame from the next time step. Both methods are approximate solutions. All transformations on the velocities are executed on the expectation value, while the transition function itself is not transformed, because the computational expense would be too high. The second method is used in our BHF approach due to a slightly better implementability in our individual implementation, but the first method is mentioned here first, since it uses the coordinate transforma-



(a) Method 1. **(b)** Method 2.

Figure 4.7: Comparison of the two methods.

tions in a more intuitive order.

Method 1

The first method is illustrated in Fig. 4.7a. The individual steps of the algorithms are:

1. Calculate prediction expectation values with attractors from *context* and starting position (blue).
2. Rotate start position and end position of the vector $v_R \longrightarrow \tilde{v_R}$.
3. Use the new prediction function described by the velocity expectation $\tilde{v_R}$ to distribute the probability flow.

The attractor function defines $v_R = f_A(v_A, v_{abs})$ with $v_{abs} = v_l + v_{Ego}$ and v_A denotes the velocity determined by the attractor algorithm $v_A = ATTR(context_{t-1})$. Since attractors have not yet been introduced, the reader may interpret the attractor function f_A as the identity function on $v_R = f_A(v_{abs}) := v_{abs}$ in order to ignore the attractor influence.

Method 2

The second method is used in our BHF approach. It is illustrated in Fig. 4.7b and consists of the following steps:

1. Use rotated starting point and rotated lane context $context_{t-1} \longrightarrow \widetilde{context}_{t-1}$.
2. Calculate transition function with the attractor derived from predicted $\widetilde{context}$ and the red starting point.

3. Use the new prediction function described by the velocity expectation $\tilde{v_R}$ to distribute probability flow.

The attractor function defines $\tilde{v_R} = f_A(\tilde{v_A}, \tilde{v_{abs}})$ with $\tilde{v_{abs}} = \tilde{v_l} + v_{Ego}$ and $\tilde{v_A}$ is the velocity determined by the attractor algorithm generated by the rotated lane context $\tilde{v_A} = ATTR(\tilde{context}_{t-1})$. For this method the reader may for now look at the attractor function f_A as the identity function on $\tilde{v_R} = f_A(\tilde{v_{abs}}) := \tilde{v_{abs}}$ in order to ignore the attractor influence.

4.4.3 Integration into the BHF via the Prediction Model

The theoretical background of the ego-motion compensation in rotated coordinate systems was discussed in the previous subsections. Now we incorporate the results in the prediction step of the BHF in order to compensate the ego-movement when predicting the position of the observed vehicle.

The type signature of the prediction function (Eq. 4.7) is therefore extended by two additional parameters: The ego-vehicle influence at node i on the observed velocity $\mathbf{v}_{ego,i}$ and the yaw rate of the ego-vehicle ω. The context is also needed as input when the use of attractors is enabled (cf. Chp. 7). The attentive reader will also notice that the probabilistic transition function (Eq. 4.5) now depends on the additional ego-movement variables, in addition to the existing observed driver action variables β and γ, and therefore the input vector $\mathbf{u}$ increases to $\mathbf{u} = (\beta, \gamma, \mathbf{v}_{Ego}, \omega)^T$. We restate Eq. 4.7 here as Eq. 4.9 for the readers convenience.

$$
\begin{aligned}
f_P(\mathbf{v}_{i,t-1}, \mathbf{l}_k, \mathbf{l}_i) := \quad & \alpha(\omega(\mathbf{l}_k - \mathbf{l}_i); \omega(\mathbf{v}_{i,t-1}), \sigma_\beta^2) \\
& \cdot\gamma(||\mathbf{l}_k - \mathbf{l}_i||/\Delta T; ||\mathbf{v}_{i,t-1}||, \sigma_\gamma^2) \\
+ \quad & \alpha(\omega(\mathbf{l}_k - \mathbf{l}_i) + \pi; \omega(\mathbf{v}_{i,t-1}), \sigma_\beta^2) \\
& \cdot\gamma(-||\mathbf{l}_k - \mathbf{l}_i||/\Delta T; ||\mathbf{v}_{i,t-1}||, \sigma_\gamma^2)
\end{aligned} \tag{4.9}
$$

$\tilde{f_P}$ (Eq. 4.10) with ego-movement compensation now replaces f_P (Eq. 4.9). The content of the function is explained in Alg. 4.1, since $\tilde{f_P}$ is no longer representable in equation form.

$$
\tilde{f_P}(\mathbf{v}_l, \mathbf{l}_k, \mathbf{l}_i, \mathbf{v}_{ego,i}, \omega_{t-1}, context_{t-1}) \tag{4.10}
$$

By implementing this function the BHF is ready for ego-motion compensation. In the next two subsections we will first briefly discuss the BHF behavior without ego-compensation and then show the benefits in a Carmaker simulation tracking task.

Algorithm 4.1: The new prediction function $\tilde{f_P}$ in time step $t-1$.

input : $(\mathbf{v}_l, \mathbf{l}_k, \mathbf{l}_i, \mathbf{v}_{ego,i}, \omega_{t-1}, context_{t-1})$
global input: ΔT
output : The transition activity from node i to node k
`// First step. Rotate position and context.`
$\tilde{\mathbf{l}_i} := predictPoint(\mathbf{l}_i, \omega_{t-1}, \mathbf{v}_{ego,i}, \Delta T)$
$\widetilde{context} \leftarrow$ predictContext$(context_{t-1})$
`// Second step. Translate to absolute velocities and feed into prediction function.`
$\tilde{\mathbf{v}_{abs}} \leftarrow \tilde{\mathbf{v}_l} + \mathbf{v}_{Ego,i}$
if UseAttractors **then**
  $\tilde{\mathbf{v}_A} \leftarrow$ ATTR$(\widetilde{context})$
  $\tilde{\mathbf{v}_R} \leftarrow f_A(\tilde{\mathbf{v}_A}, \tilde{\mathbf{v}_{abs}})$
else
  $\tilde{\mathbf{v}_R} \leftarrow \tilde{\mathbf{v}_{abs}}$
end
return $f_P(\tilde{\mathbf{v}_R}, \mathbf{l}_k, \tilde{\mathbf{l}_i})$

4.4.4 The Ego-Movement without Ego-Movement-Compensation

The BHF can also be used in a transferred reference frame without ego-movement compensation. The resulting errors are small, as long as the velocity changes and the yaw rate are small. That means that in highway situations with high curve radii and without harsh accelerating or decelerating actions by the driver, the effects of the ego-motion on the tracking result are small.

As already mentioned, the biggest challenges for ADAS lie in inner city scenarios. In inner city situations the ego-vehicle will turn with steeper radii than in highway situations. This is why new tracking techniques should cope well with ego-movement when they are planned to work in real world inner city scenarios.

The explicit compensation of the ego-movement makes it possible to use a *constant absolute velocity assumption* instead of a *constant relative velocity assumption* on the observed vehicles. The first means that the prediction model assumes that the absolute velocity of the observed vehicle stays constant without driver input ($\alpha = 0 \wedge \omega = 0$), and the second assumes the same for the relative velocity. Only for the sake of completeness do we mention that the latter model may also have benefits. Note that both assumptions guarantee that the absolute velocity will stay the same when the ego-driver input is zero (meaning that the yaw rate and the ego-velocity is not changed: $\Delta\omega = 0 \wedge \Delta v_{Ego} = 0$) and therefore no velocity needs to be compensated. Early versions of our MDBHF approach (cf. [5]) used

a constant relative velocity assumption. It delivered good results in large curves, as used in the TORCS racing simulation. In a wide and steady curve the relative movement of the observed vehicle stays the same while the absolute movement changes. This has the benefit of the estimated PDF staying in the lane in large smooth curved roads instead of trying to model a straight movement with constant absolute velocity, which would leave the curve.

In the newer experiments the attractor algorithm is used to achieve a lane-following behavior. Curves in Carmaker or in the real world are often much sharper than the curves used in racing tracks. The errors generated by the constant relative velocity assumptions are therefore higher than the benefits. There is no need to use a fixed relative velocity assumption instead of a fixed absolute velocity assumption when using attractors.

In curves with small curve radii the attractor algorithm clearly outperforms the positive side effects of the constant relative velocity assumption. In Sec. 6.3 we analyze the errors caused by a missing ego-movement compensation. The results clearly show that the ego-movement compensation is necessary for inner city tracking tasks.

Chapter 5

Position Tracking

Advanced driver assistant systems need to know the position of vehicles around the ego-vehicle in order to derive actions or driver warnings in dangerous situations. The typical data flow from the overall system input to the overall system output can be separated into the following modules:

1. The sensors and the sensor post-processing
2. Object/vehicle detection
3. Data association
4. Object/vehicle tracking
5. Action derivation
6. Output to the vehicle or driver.

This classification is valid for ADAS which use the concept of object hypothesis. We have already mentioned that ADAS tasks that allow the use of strong assumptions on the situation, e.g. in highway scenarios, may manage tasks without the concept of objects (cf. introduction of Chp. 4).

This chapter deals with the fourth module, the object tracking task. The core of the object tracking module is the filtering of sensor information from the sensor and object detection channel. As previously discussed, the Bayesian filter consists of a prediction step and a filter step. The prediction step was described in detail in the last chapter, and we now focus on the filter step. However, before the new sensor information can be integrated in, it has to pass through the previous modules, which we outline below.

There exists a vast number of sensors which can be or which are already used for ADAS. They can be categorized into stationary or on-board sensors. There are big projects which attempt to monitor dangerous areas like intersections [48], and systems which monitor traffic in order to optimize the

traffic flow [57], mostly with stationary sensors attached to the infrastructure. The quality of the needed sensors and the budget for high-precision sensors depend highly on the task. It is costly to equip infrastructure with sensors. Maintaining is often more expensive than the costs of the technical parts themselves and the needed standardization demands high costs for lobbying. The system should work with the vehicles of all manufacturers and, depending on the technology used in the end, some car manufacturers will have advantages in the beginning. Many car manufacturers focus therefore on on-board sensor systems for ADAS.

Those who use on-board sensor applications have two philosophies. The first is related to finances. Relatively inaccurate low-cost sensors, which are already integrated into top-of-the-range models, are used by many car manufacturers for ADAS solutions. This is because the current consumer wants ADAS solutions without much additional charge and the dictate of the car designers is to disallow any superstructures on the vehicles. The second philosophy is to use as much technology as possible. The most expensive sensors or sensors that need superstructures on the cars, as well as world-knowledge by map databases in order to get a detailed image of the surrounding world. The most famous example of this approach is Google. The inaccuracy in the world knowledge is so far reduced that even prototypes of vehicles which drive autonomously in city-center and rural conditions are possible. The results of current ADAS systems using low-cost sensors are far from such a world-knowledge. Perhaps the car-manufacturers will also use map-data more intensively in the future in order to reduce the inaccuracy in the world knowledge, with Google and others acting as map providers for them. How such map data can be used to improve tracking is part of Chp. 7.

Depending on which sensors are used an individual sensor post-processing and object detection algorithm is used. The detection of a vehicle on a camera image differs from the vehicle extraction of a radar echo. Often, especially in the radar sensors used by the car manufacturers the sensor post-processing and the object detection are done by the sensor as a black-box component. For camera sensors, these algorithms are separate and independent from the camera manufacturers. One of the many reasons for this is that features extracted from camera sensors are more diverse. The ADAS systems not only try to extract cars from the camera image, but also lane markings, the ego-lane [47] or the free driving space [46]. The properties of an object extracting algorithm used in our real-world evaluation are explained in Sec. 10.4.1. The BOF approach mentioned previously (cf. Sec. 2.2.3), can also be used on the sensor post-processing level to fuse sensor information on a low-level representation before extracting features from the BOF-map in order to detect objects. All algorithms that extract objects, whether they are working on camera images, radar or the clustering algorithm working on BOF-maps [27], can act as a virtual sensor for

the object-tracking-module. Unfortunately, the object hypotheses delivered by the object detection module are often not PDFs, but simple positions. In addition, the error distributions are often unknown and therefore it is also difficult to rebuild PDFs from the position of the object hypothesis and the error distribution. Much can be improved in this field by sensor manufacturers and by the developers of virtual sensors. Too much information is getting lost in the first two modules with the state-of-the-art solutions. This information in the form of PDFs instead of single positions could be used meaningfully by non-Gaussian Bayesian filter implementations. The reason that virtual sensors providing PDFs are not widespread in use is a vicious circle. It is the consequence of the fact that most state-of-the-art tracking algorithms can only cope with Gaussian input, and therefore the engineers demand sensors to deliver single positions or, at best, Gaussian PDFs as an input for tracking. This is similar to the self-delusion that exact positions are better than PDFs, that is later mentioned in Sec. 6.2.3. The lack of virtual sensors providing PDFs also downscales the positive effects when using non-Gaussian Bayesian filter implementations.

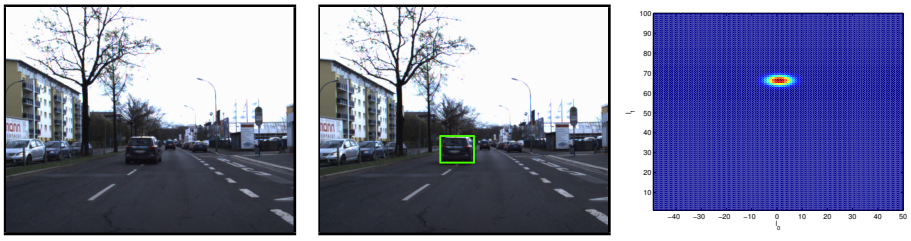
In the following section we show basic ideas as to how low-cost sensors like radar and camera can be modeled in the BHF. These models are used in our simulations and in real-world-scenarios in later chapters. Subsequently, we explain the vehicle tracking in the MDBHF using the sensor models together with the prediction models. We show that the usage of non-Gaussian sensor and non-Gaussian prediction models can deliver better results in a tracking task.

5.1 Modeling Sensors

Sensor models (also known as *measurement models*) reflect the knowledge about the errors a sensor has when sensing a vehicle at a certain position. We first explain the theoretical concept of sensor models with a row of simple examples. Then we show how we modeled sensor models for ADAS in order to use the sensor information for the tracking task in the next section.

5.1.1 Measurement Models and Inverse Measurement Models

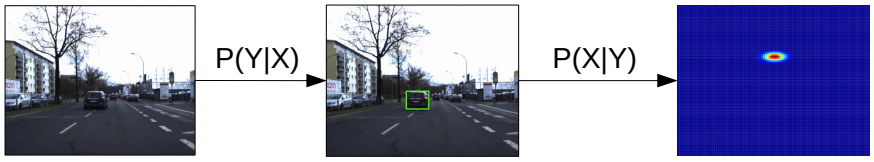
The measurement model (or sensor model) describes the sensor measurement depending on the vehicle state in the real world. It is a CPD since sensors always deliver a noisy and incomplete image of the world, e.g. the camera sensor which images the 3D world onto a 2D image. In Bayesian filters we use an *inverse sensor model* to map the 2D image back to a 3D position. Of course, like the sensor model, it is also fraught with uncertainty and therefore a CPD (cf. overview in Fig. 5.1).



(a) Real world.

(b) Sensor annotates vehicle in camera image coordinates.

(c) The vehicle position in the model world.



(d) The measurement model $P(Y|X)$ determines were the sensor detection will be given the real position. The inverse measurement model $P(X|Y)$ reasons the vehicle position that caused the measurement.

Figure 5.1: The real world is sampled by the camera sensor. Within the camera image a virtual sensor detects the vehicle position in camera image coordinates. The real position can be estimated using the inverse measurement model.

The word 'inverse' is used in the inverse measurement model, because it reasons from the effects to the causes [65]. In Bayesian filtering the measurement model and the inverse measurement model can be substituted for each other by using the Bayes' theorem. In some situations it is easier to postulate the inverse measurement model instead of modeling the forward measurement model. Further information on sensor models used for range sensors like laser scanners can be looked up in [65].

For a better understanding of inverse and forward sensor models we use a minimal working example in this chapter. We introduce a light barrier array as a sensor. Each of the six single light barriers Y={1,..6} detects whether there is an object in the line between the light transmitter and the photoelectric receiver at corresponding position X={a,b,c,d,e,f}. This is probably one of the simplest designs to determine the position of an object. Additionally, we assume that the object size is exactly the size of the distance between two sensors, so that it is always detected by exactly one light barrier in the array. Without loss of generality we assume that it is a stationary sensor instead of an on-board sensor. This does not change the basic understanding.

In the first example, light barrier four detects an object, therefore the CPD created by the sensor measurement $P(X|y=4)$ is $P(x=d|y=4) = 1 \wedge P(x=X \setminus \{d\}|y=4) = 0$ as illustrated in Fig. 5.2.

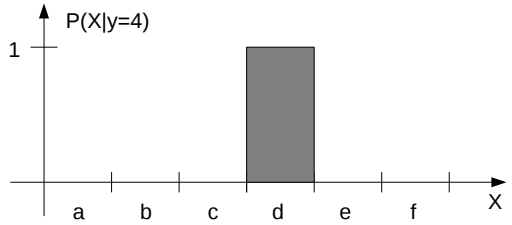


Figure 5.2: Sensor y=4 has a detection, meaning the object needs to be at position d.

In the first example we assumed that all light barrier arrays are properly mounted and cannot be shifted to the side. The direct assignment of 1 indicates that this can also be solved with deterministic functions. We assume in the second experiment that the same light barrier array is not mounted on the floor and can be shifted to each side. The reason for that shifting is unclear and not important, perhaps it was shifted by our lab dog Kelly. We know from previous observations that there is a 20 percent chance that it shifted by exactly the margin between two light barriers to the left or with the same chance to the right, so the neighbor light barrier will sense an object instead of the correct light barrier. This can be perfectly modeled

by a probabilistic sensor model, and the corresponding CPD is illustrated in Fig. 5.3. Note that the light barriers are mounted and fixed on a structure and only the structure as a whole can move. Therefore the CPD is the same for the whole input dimension y.

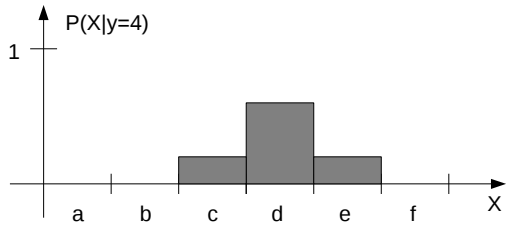


Figure 5.3: The CPD for y=4. There is a 60% chance that the object is at position d. Note that the full CPD is a function with a two dimensional input space assigning each (x, y)-tuple a probability value. We set the y dimension fixed to 4 and illustrate the probability of an object occurrence over the x direction in this graph. For each sensed position y the CPD assigns a probability distribution over x.

Note that the inverse measurement model is equal to the measurement model in this example (cf. Fig. 5.4), since $P(Y|X) = P(X|Y)P(Y)/P(X)$ and the object can appear in the whole array with the same probability ($P(X)$ is a uniform distribution) and each light barrier is as sensitive as any other barrier ($P(Y)$ is a uniform distribution).

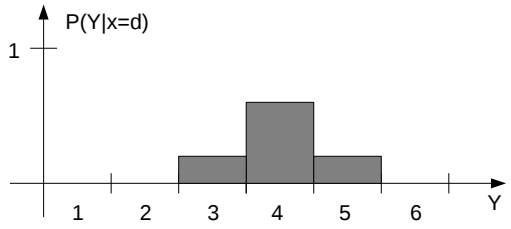


Figure 5.4: The forward sensor model for x=d. When the object is at position d there is a 60% chance that the light barrier 4 will be activated and a 20% chance that light barriers 3 and 5 will be activated.

Let's first focus on how we model sensors used in the ADAS domain. Later we will use the above minimum working example again in order to explain the problem of multi-object tracking and the data-association problem.

5.1.2 Radar and Camera Sensors

In most ADAS approaches the sensor model is modeled as Gaussian noise in the position space. We already used more sophisticated prediction models in this thesis. In this subsection we will introduce two more sophisticated sensor models and use them in the simulated and real-world evaluations. In the simulations the model parameters of the sensor model match exactly the simulated sensor characteristics. We are able to set the measurement model this way because the noise model is known under simulation conditions.

In the real-world scenes the measurement model parameters are chosen as a rough estimate since, as already mentioned above, object detection algorithms or virtual real world sensors do not provide sensor noise distributions. Such distributions can be determined for specific sensors under specific conditions by excessive empiric evaluations. This is not reasonable here since object detection algorithms are under constant development and therefore the measurement model undergoes changes. Additionally, the noise performance of the virtual sensors depends on the situation, the weather, the brightness, and parameters of the detection algorithm. Therefore, the parameters of our measurement models are manually adjusted to the situation.

In our experiments we mainly made use of one of two different sensor types, the radar sensor or the camera sensor. Both are low cost sensors used for ADAS tasks. The origin of the reference system, in which the state X is measured, is set in each case to the sensor position in order to avoid conversion between the sensor and the ego-vehicle reference system (cf. Sec. 3.2).

Radar Sensor

The radar sensor is rather good at estimating distances. It emits electromagnetic waves and measures the echoes created by obstacles with help of a radar antenna. Using the Doppler effect each position measurement is also assigned a velocity value. The drawback of this sensor type is that ghost echos can occur from reflections. The noise in the angular dimension is rather high. The advantages in the ADAS domain are the sensing capabilities through fog and rain.

Although the velocity measurement can be easily used by our MDBHF approach to assign a value to the velocity dimension, we decided not to incorporate this information, but to derive the velocity by the position dimension, because we did not have access to real-world radar sensors with such velocity information. From Bayesian filter theory we know that a BHF proved to work without this easing will also work with better results with a measured velocity.

Equation 5.1 shows our measurement model for the radar sensor. Using the Bayesian formula and the assumption that the a priori distributions P(X)

and P(Y) are equal we can directly state the inverse measurement model in Eq. 5.2. They are equivalent here since the Gaussian terms α and γ, which denote the expectation value parameter and the sample point parameter, can be swapped.

$$P(\mathbf{y}|\mathbf{x}) := \quad \eta \cdot \alpha(\omega(\mathbf{y}); \omega(\mathbf{x}), \sigma^2_{angle}) \cdot \gamma(||\mathbf{y}||; ||\mathbf{x}||, \sigma^2_{distance}(||\mathbf{x}||)) \quad (5.1)$$

$$P(\mathbf{x}_k|\mathbf{y}) := \quad \eta \cdot \alpha(\omega(\mathbf{l}_k); \omega(\mathbf{y}), \sigma^2_{angle}) \cdot \gamma(||\mathbf{l}_k||; ||\mathbf{y}||, \sigma^2_{distance}(||\mathbf{y}||)) \quad (5.2)$$

η is the normalization constant in order to receive a proper PDF. σ^2_{angle} is the noise in the angular dimension and $\sigma^2_{distance}$ the noise in the distance measurement. The remaining variables are already known: $\mathbf{l}_k$ is the position of the representative of the cell $\mathbf{x}_k$ and $\mathbf{y}$ is the position of the sensor measurement.

The relative uncertainty in the measured distance depends on the distance of the object. The uncertainty in the noise is modeled over-proportional to the distance. It is therefore implemented as a function depending on the distance $||x||$, or for practical considerations, $||y||$. Since the real distance is not known by the sensor model, we use the simplification $||\mathbf{x}|| \approx ||\mathbf{y}||$ and use the measured distance $||y||$ for the noise function instead of using the more correct but unknown $||\mathbf{x}||$.

The noise $\sigma^2_{distance}$ (Eq. 5.3) increases by 2% with increasing distance (value used at Honda Research Institute Europe (HRI-EU)).

$$\sigma^2_{distance}(||\mathbf{x}||) := ||\mathbf{x}|| \cdot 0.02 \quad (5.3)$$

Camera Detection Sensor

The object detection working on camera images is also more precise in closer locations, since more object features are more clearly visible. We can reuse the radar model Eq. 5.2 in order to postulate the camera model. The distance noise $\sigma^2_{distance}$ needs to be adapted to the specifics of cameras.

In the camera sensor model the noise variance in the camera depth dimension also depends on the distance, but the noise in the distance is determined by the pixel resolution in the camera depth dimension. With increasing depth the camera resolution decreases, becoming zero in the vanishing point. At HRI-EU the following model is used: The camera resolution in the depth dimension (Eq. 5.4) depends on the pixel width in meters d_{pixel}, the focal length d_{focal} and the camera baseline $d_{baseline}$.

$$\sigma^2_{distance}(||\mathbf{x}||) := \frac{1}{2} \frac{d_{pixel}}{d_{focal} d_{baseline}} \cdot ||\mathbf{x}||^2 \quad (5.4)$$

The factor $\frac{1}{2}$ is used since the deviations are possible to the same extend in each direction and therefore split by the factor. The provided parameter values are $d_{pixel} = 1,10 \cdot 10^{-5}$ m, $d_{baseline} = 0.3$ m, $d_{focal} = 0.0120$ m.

As a result the relative measurement error increases quadratically in the camera model in comparison to the linear increase in the radar model. The changing noise value highly impacts the influence of the measurement on the estimate. When tracking vehicles near the ego-vehicle the noise in the sensor measurement is low, the estimate sharp and therefore the relatively wide prediction model comparably unimportant for the tracking. With further distances the measurement model is very wide and gives less information on the tracked vehicle, and the estimate, and therefore the prediction is weighted higher by the Bayesian filter. In Sec. 5.2.2 we will briefly discuss this effect. First, however we have a look at how to fuse the position measurement into the position estimate.

5.2 Fusing Sensor Information

With proper, probabilistic sensor models in hand, we can now proceed with fusion the sensor measurements into the state estimates in the MDBHF. In 2.2.2 we learned that the Bayesian filtering consists of a prediction step and a filter step. While in the prediction step the estimate distribution is distributed according to the assumptions on the movement, in the filter step the sensor measurement of the new time step is combined with the predicted estimate.

The first subsection deals with the filter step itself, and the second calls to mind how recursive filtering without prediction would look like.

5.2.1 The Filter Step of the MDBHF

The filter step of the MDBHF does not deviate from the filter step of a usual BHF (Eq. 2.24). Filtering takes place in the position dimensions only. The velocity is derived by the position and not measured by a sensor, and there is therefore no fusion necessary between a predicted velocity and a hypothetically measured velocity.

Eq. 2.24 is restated here for the sake of clarity as Eq. 5.5. The variable z is renamed to y to comply with our MDBHF notation.

```
For all grid cells k do:
```

$$p_{k,t} = \eta p(y_t|\hat{x}_t) \cdot p_{P_{k,t-1}} \tag{5.5}$$

In contrast to the rather complicated prediction step, the filtering step is easy to implement. Eq. 5.5 shows how a sensor input can be fused into the estimate by multiplying the value of the measurement model $(y_t|\hat{x}_t)$ sampled at cell k and the existing probability value $p_{P_{k,t-1}}$ (which represents the predicted estimate) in cell k. The result must again be normalized to a probability density function.

The simple multiplication executed over all grid cells in order to fuse the sensor input with the predicted estimate is only possible due to the independence assumption (independence was briefly discussed in Chp. 2. When X_P and Y are independent the following is a correct equation: $P(X|Y, X_P) = P(Y) \cdot P(X_P)$). The application of the filter step has a correcting influence on the estimate, which is made explicit in the following subsection.

5.2.2 Position Estimation of Static Objects and the Balance between Measurement and Prediction

The filter step in the MDBHF can be understood best when estimating the position of static objects. In 4.3 we applied the prediction step in a loop. For the tracking of static objects we can also apply the filter step in a loop, since the prediction step is only necessary in order to model moving objects.

Hence the filter step on its own only works correctly for static objects. The underlying assumption is in this case that the measurement of the current time step is generated by exactly the same ground-truth state $\mathbf{x}$ as in the previous time step. The multiplication in the filter step provides a combination of the knowledge of both sources of information, the estimate from the previous time step $\mathbf{X}_{t-1}$ and the measurement $\mathbf{Y}_t$. When using Gaussian distributions, it can be clearly seen in the Kalman-filter equation 2.17 that the multiplication generates a weighted averaging. For the reader's convenience the one-dimensional Kalman equation is restated here as Eq. 5.6.

$$P(x_t|y_{1:t}, u_{1:t}) = \alpha \exp\left(\frac{1}{2}\frac{(x_t - \frac{(\sigma_{t-1}^2+\sigma_x^2)y_t+\sigma_y^2\mu_{t-1}}{\sigma_{t-1}^2+\sigma_x^2+\sigma_y^2})^2}{\frac{(\sigma_{t-1}^2+\sigma_x^2)\sigma_y^2}{\sigma_{t-1}^2+\sigma_x^2+\sigma_y^2}}\right) \tag{5.6}$$

$\sigma_{t-1}^2 + \sigma_x^2$ can be aggregated in the equation, and is then the variance of the predicted estimate ${\sigma_{Pt-1}}^2$. The term 'predicted estimate' is not correct for a filter only step, so that we will call it 'prior estimate' in this subsection. The new expectation value of the estimate after the filter step is the output of a weighted average over the prior estimate and the measurement

$$\mu_t = \frac{{\sigma_{Pt-1}}^2 y_t + \sigma_y^2 \mu_{t-1}}{{\sigma_P}_{t-1}^2 + \sigma_y^2} \tag{5.7}$$

and the variance of the new estimate is

$$\sigma_t^2 = \frac{{\sigma_{Pt-1}}^2 \sigma_y^2}{{\sigma_{Pt-1}}^2 + \sigma_y^2} \tag{5.8}$$

The weights of the weighted averaging in Eq. 5.7 are the variance of the predicted estimate ${\sigma_{Pt-1}}^2$ and the variance of the measurement σ_y^2.

The variance is inversely proportional to the information the distribution contains. A high variance in the predicted estimate means that the prior estimate contributes less information to the new estimate and therefore the measurement is weighted higher in the averaging.

The resulting distribution contains more information on the estimate and has therefore a variance smaller than the smallest variance of the combined variables (cf. Eq. 5.8). By repeatedly combining the sensor measurement with the estimate, the information on the object becomes higher and higher and therefore the uncertainty in the form of the variance in the estimate becomes smaller and smaller.

This concept also applies to non-Gaussian variables or when combining sensor measurements from different sensors in the same time step. However, the stochastic variables need to be statistically independent in order to be combined. Only fully independent variables create the information gain postulated in Eq. 5.8. The more information overlapping of the to be combined variables occurs the lower is the information gain. When both are totally dependent the information gain is zero. In [34] suggestions on the combination of dependent Gaussian distributions are made.

The above findings help us to understand what will happen in our MDBHF when using the radar sensor or camera sensor introduced above. The sensors are less accurate at higher distances. That means that with increasing distance between the ego-vehicle and the observed vehicle, the uncertainty in the sensor rises and the prior estimate is increasingly trusted in comparison to the sensor input. In contrast to the static object state assumption used in this subsection, the vehicles in our experiments move and therefore a combination of the filter step and the prediction step is needed for vehicle tracking. However, the prediction step induces information loss by adding the motor noise in each iteration. At a greater distance the information gain of the filter step becomes smaller in relation to the information loss of the prediction step, and the overall information in the estimate decreases and levels out at a lower value. This behavior is absolutely consistent with our human perception of uncertainty and information. The following performance evaluation will evaluate this statements.

In the next section we introduce the concept of tracking as a combination of the prediction and filtering step.

5.3 Bayesian Tracking with the MDBHF

Having defined the probabilistic prediction model and the probabilistic sensor model, all components are now available and can be put together. Figure 5.5 shows the complete Bayesian network created by the introduction of the stochastic variables and the CPDs introduced in the last chapters.

To summarize: The vehicle location $\mathbf{l}$ as part of the state vector $\mathbf{X}$ gen-

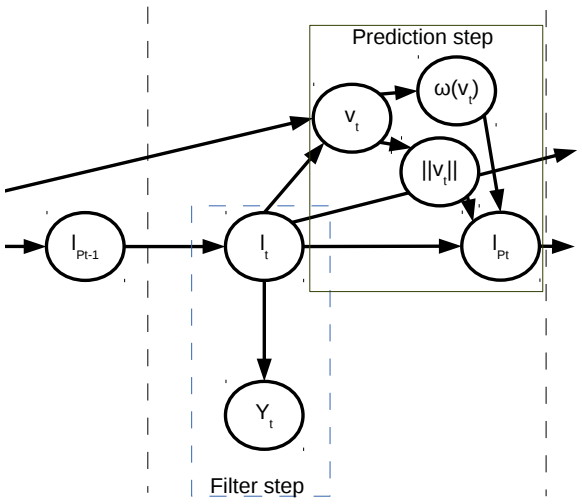


Figure 5.5: The figure shows the MDBHF as an Bayesian Network Graph. The prediction step and the filter step is highlighted. The individual time slice is marked by the vertical dashed lines.

erates the sensor measurement. This dependency is represented by the measurement model CPD. Remember that altogether the state vector consists of the variables $\mathbf{x} = (l_0\ l_1\ 0\ ||\mathbf{v}||\ \omega(\mathbf{v}))^T \in \mathbf{X}$ (cf. Sec. 3.1.1). The velocity $\mathbf{v}$ is derived by the position difference. It is merged into a direction and an absolute value in each grid cell (cf. Sec. 3.3). Both are, together with the ego-movement parameter for ego-movement compensation the inputs for the prediction model CPD (cf. Chp. 4). The result is the predicted position $\mathbf{l}_{P_t}$. The partition of the estimate into an estimated position $\mathbf{l}_t$ and a predicted estimate $\mathbf{l}_{P_t}$ position is artificial and not necessary at first glance, but has advantages as shown in later chapters.

5.3.1 The MDBHF Performance versus the EKF

Does the MDBHF perform better due to the more sophisticated models enabled by the distributed cell representation? Or will the extended Kalman filter (EKF) still outperform the MDBHF due to the disretization errors of the MDBHF or other non-concerned issues? These questions were addressed in one of our first evaluations (published in [5]).

Scenario Description

The evaluation scenario was created by The Open Racing Car Simulator (TORCS) [2]. The observed vehicle O is overtaking the ego-vehicle E in a

2-lane highway scene. After overtaking, vehicle O reduces speed and cuts into the lane of vehicle E. They then both drive then with the same speed and finally enter a right curve. Fig. 5.6 shows the scenario when vehicle O is beginning to cut into the right lane. The detailed scenario description is: The ego-vehicle E uses the MDBHF to track the position of vehicle O during the overtaking. Car E accelerates to a constant speed of 80 km/h in the right lane of a simulated road. The overtaking car O starts next to E in the left lane and overtakes car E while accelerating to 100 km/h. It stays in that lane for 8 timesteps = 4.0 seconds, which corresponds to 100 meters. It then, switches lanes ahead of car E for 4 timesteps = 2.0 seconds, while reducing speed to 80 km/h, and finally continues in the right lane ahead of car E for another further 8 timesteps until it enters a right curve. The overtaking trajectory of car O was intentionally controlled in a rather abrupt fashion to evaluate to what extent the tested filters can deal with different directional changes. In the screenshot Fig. 5.6 the estimate distribution appears to be rather wide, but the reader should notice that a log-scale color coding was used. However, when looking at the update interval $\Delta T = 500\ ms$ it is clear that a lot of uncertainty is inherent in the estimate, due to the limited amount of measurement updates. The sensors were modeled to mimic actual ones, even when ΔT was chosen rather high due to constraints of the TORCS simulator. The radar sensor uses a standard deviation in the angular dimension of $\sigma^2_{angle} = 0.218\ rad$. The distance between the grid cells is set to $d = 0.5\ m$

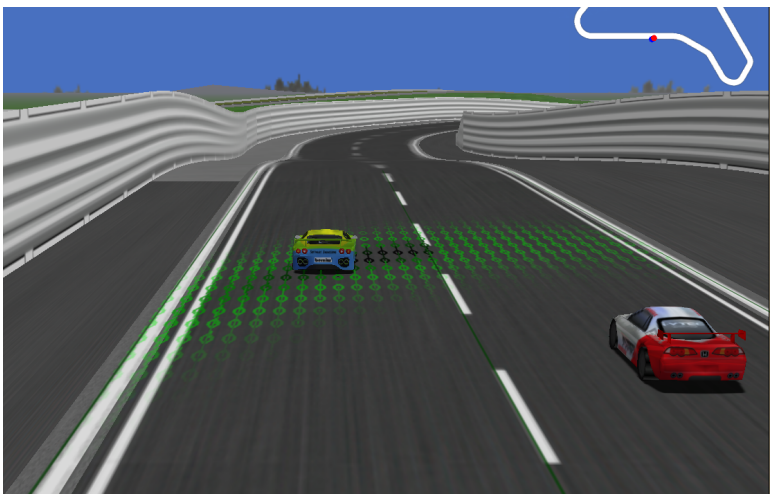


Figure 5.6: A screenshot from the test scenario. The yellow car O is overtaking the red ego-vehicle E, heading for the curve. The active grid cells representatives are drawn in perspective on the street.

Performance Evaluation

In order to measure the tracking quality we use as *metrics* the Euclidean-distance $dist(E(P(X_t)), x_{real})$ between the expectation value of the estimate $E(P(X_t))$ and the ground truth position x_{real} and the probability assigned to the ground truth position by the estimate denoted by $P(X = x_{real})$. In later chapters we will introduce and discuss a wide variety of more sophisticated quality measures. Note: $E(P(X_t))$ is found by a probability weighted averaging of the positions of all grid cells. The parameters are evaluated in 50 runs with the camera measurement model and camera noise, and in 50 further runs with the radar measurement model and radar noise, each with an identical trajectory.

Fig. 5.7 shows the trajectory of the ground truth positions x_{real}, the expectation value $E(P(X_{0_t}))$ and variance $\sqrt{VAR(P(X_{0_t}))}$ in the x_0 direction of the MDBHF estimate and the EKF estimate. In the figure and in Table 5.1 it can be seen that the MDBHF outperforms the Kalman filter in all but one stage of the scenario. It is only in the lane changing state, and then only when using the radar sensor, that the Kalman filter performs better due to a faster reaction to the lane change. The MDBHF estimate is slightly delayed upon the lane change but yields a lower deviation of its expectations. This is because the trust in the sensor is set too low by the radar model of the MDBHF and modeled with lower variance by the Kalman filter sensor model. The camera sensor settings in the MDBHF and the Kalman filter are generally far more comparable and in all cases the distance metric indicates a better performance of the MDBHF (cf. Table 5.1).

Table 5.1: Average Euclidean distances *dist* between real position x_{real} and expectation value of the position $E(P(X_t))$ and average standard deviations $\sigma = \sqrt{VAR(P(X_t))}$ of the expected positions during the different stages of the overtaking maneuver

Stage	**Driving by**		**Changing Lane**		**In front**	
Radar Data	dist	σ	dist	σ	dist	σ
MDBHF	0.49	1.51	1.81	2.08	0.62	2.24
EKF	0.60	2.83	0.84	4.57	0.72	4.50
Camera Data	dist	σ	dist	σ	dist	σ
MDBHF	0.18	0.86	0.72	1.36	0.33	1.59
EKF	0.22	0.83	1.06	1.37	0.66	1.60

The $E(P(X_t))$ interpretation (cf. Fig. 5.8 and Table 5.2) backs the conclusions drawn from the distance measure. This means, that at least with these parameter settings and this situation, that the MDBHF can outperform the EKF in filtering tasks. It can be used for tracking the position of a vehicle equal to the EKF.

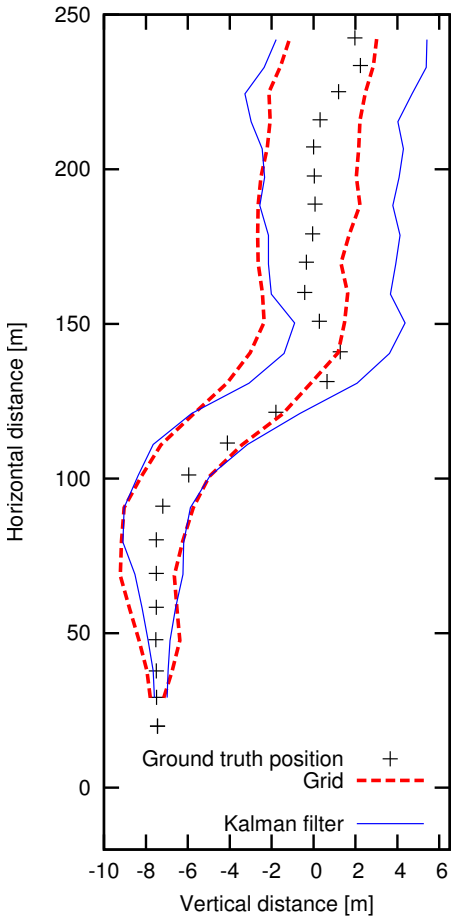


Figure 5.7: Mean and standard deviations of the expectation values of the posterior distributions of the filters are compared with the actual overtaking vehicle position, plotting corridors of one standard deviation from the mean for each filter approach. Compared are the MDBHF and the EKF.

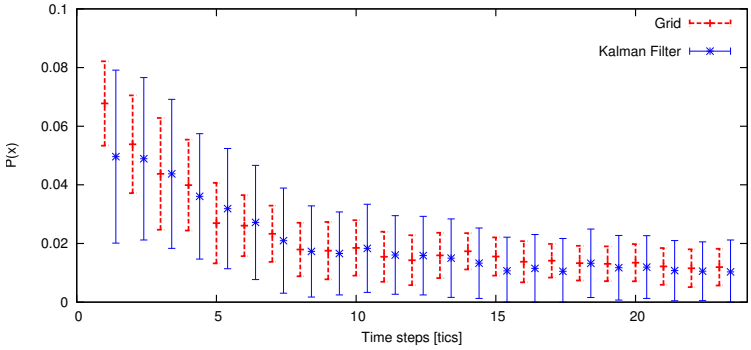


Figure 5.8: The probability $P(x)$, and standard deviation, for the likelihood of the true state x during the overtaking situation with radar sensor information.

Table 5.2: The average probability $P(X = x_{real})$ at the real position x_{real} in the different stages of the overtaking maneuver.

Stage	**Driving by**	**Changing Lane**	**In front**
Radar Data			
MDBHF	0.0374	0.0168	0.0140
EKF	0.0235	0.0163	0.0121
Camera Data			
MDBHF	0.0744	0.0246	0.0151
EKF	0.0331	0.0174	0.0104

Conclusion

Of course, it is always difficult to produce a fair comparison between an optimized own approach and another existing approach as a benchmark. We tried to be fair in the comparison and optimized the Kalman filter as far as possible. The models of the Kalman filter are limited to Gaussian noise, so we formed and rotated the models in a way that best matched the uncertainty in the sensors and the motor noise. For this reason we preferred to use an EKF, with a sensor model depending on the measured angle to the observed vehicle. More details on this modeling can be read in [5].

Before going from the position tracking to the driving behavior tracking in the next chapter, we offer a short glimpse on multi-object tracking in the next subsection.

5.3.2 Multi-Object-Tracking and the Data-Association Problem

In real situations there is usually not just one vehicle to track, but multiple vehicles. Therefore the object detection algorithm detects a number of M vehicles at once. Bayesian filters can only handle one object hypothesis, so M Bayesian filters are needed. A pre-selection must be done, which decides which measurement is routed to the input of which Bayesian filter. The problem of finding a correct pre-selection is called the *Data-Association-Problem*.

For better comprehension we will again use the sensor array as a minimal example. Lets assume that light barriers 2 and 4 both detect light absorption. Then, assuming that both light barriers are similarly trustworthy, the CPD of the sensor consists of two peaks, each with 0.5 change due to the normalization of the sum to 1 (cf. Fig. 5.9).

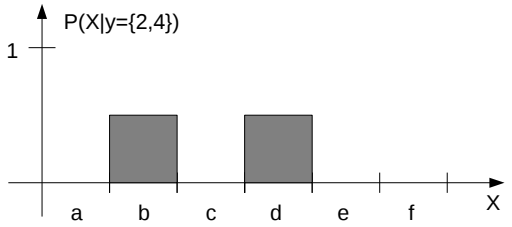


Figure 5.9: Light sensors 2 and 4 are both activated with the knowledge or the underlying assumption that only one object can or does trigger the activation.

The measurement model in the multi-object tracking has to take into consideration the fact that a measurement can be the result of different objects $P(Y|X_{o_1}, X_{o_2}, .., X_{o_M})$. The inverse measurement model is hard to

postulate, since the detection cannot be associated to exactly one input. Hence, it is not a function.

Depending on the sensor type a detection algorithm has different properties when mapping the detection exactly to a vehicle: The color of the vehicle, the size, direction or velocity of the detection and the proximity to other detections. Unfortunately, a detection can be temporarily lost by, for example, obstruction or weaknesses of the detection algorithm. In some cases the vehicle will mistakenly be assigned to a new object after the loss. For this reason, the detection algorithm also needs a kind of tracking, which can be executed by a BOF or by a backward information flow from the vehicle tracking top-down to the object detection algorithm. Such information is used, for example, by the joint probabilistic data association (JDPA) algorithm [27]. A survey of data association techniques can be found in [32].

The association from a detection to an object hypothesis will never be accurate, a problem which has to be solved with probabilistic models. In most concepts the detections have influence on the object hypothesis which is weighted with the probability of causality. Each object hypothesis depends therefore on all detections to a certain degree. We will not get into the mathematical details here and refer to the cited literature for further readings. In the following we assume that the data association is done correctly by the detection module which delivers the object position of a single object hypothesis as a virtual sensor.

As a last point we want to graphically outline the overall concept again: how ADAS deal with ambiguous detections. Either the multiple peaks in the detection are decomposed into multiple object hypothesis when transferring information from the detection step to the tracking step (cf. Fig. 5.10), or all information is first gathered in a representation within the detection step, e.g. in a BOF (cf. Fig. 5.11) in order to do the data association by a clustering on that gathered representation.

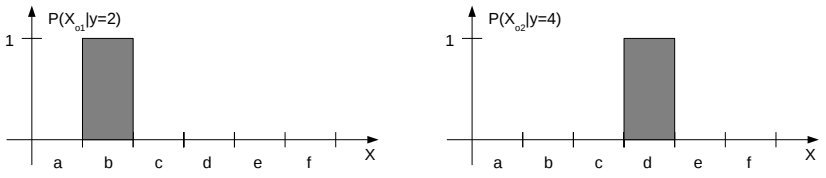


Figure 5.10: The detections are associated to two different objects o_1 and o_2. The input is fed into two different Bayesian filters.

In conclusion, each Bayesian filter has, by definition, the "problem" that it can only deal with single objects, so when using BHFs an own BHF is needed for each object hypothesis. Since multi-object tracking is a problem that holds true for all Bayesian filters, it is not especially relevant for the

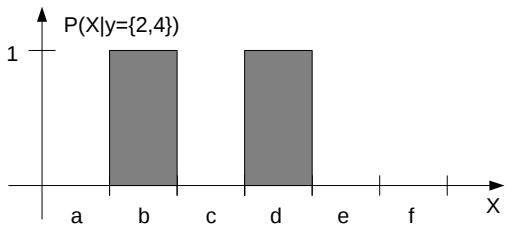


Figure 5.11: Light sensors 2 and 4 are both activated and the sensor input is fed directly into a BOF without an object assignment. Positions b and d seem to be occupied in the BOF representation.

scope of this work, but the problem is related to the task of finding out the correct behavior model of a certain driver, which we discuss in Part III.

In the Chp. 7 we will start to model the behavior of the observed driver in order to later estimate the correct behavior. In contrast to the multi-object tracking, where the BHF acts like all Bayesian filters without any specific advantages, the distributed grid cell representation of the BHF has several advantages when it comes to behavior modeling. But first we will have a small excursus into error analysis of the MDBHF.

Chapter 6

BHF Error Analysis

The MDBHF is now fully introduced. As was shown, the MDBHF essentially approximates the environmental state and the progress over time by means of an enhanced version of the BHF. Filters of this kind inevitably cause estimation and prediction errors due to the necessary compact representations. We already discussed erroneous model assumptions in Sec. 4.2 and errors due to the sparce velocity dimension in the MDBHF in Sec. 3.3.3 during its introduction. Now we focus on discretization errors of the BHF and other errors that are typical due to the grid representation. Additionally we evaluate the MDBHF behavior on systematic sensor errors and the error when running the filter without compensation of the ego-motion.

6.1 Tracking in a Simulated Example Situation with Varying MDBHF Parameters

In the following subsections we evaluate the performance in the tracking in a simulated scene (cf. Fig. 6.1). The scene is identical with the one published in [5].

The MDBHF's task is to track surrounding vehicles observed from a moving ego-vehicle (ego-velocity $v_{ego} = 90km/h$). The test scenario is a highway scene created with the TORCS simulator. It consists of five phases in which an observed vehicle driving in front of the ego-vehicle is observed. The measurements were updated every $\Delta T = 0.2s$ (In contrast to $\Delta T = 0.5s$ in [5]). In the first phase, the observed vehicle changes from the left lane to the right lane, where the ego-vehicle is and stays in that lane (approximately time step 0 to 30). In the last part of the first phase the observed vehicle is doing an abrupt correction of its driving direction in order to maintain a path along the lane center of the right lane. In phase 2 (time step 30 to 50) the observed vehicle drives in front of the ego-vehicle on a straight highway segment. In time step 50 the observed vehicle enters a curve to the right (begin of phase 3). In time step 57 the ego-vehicle also enters the

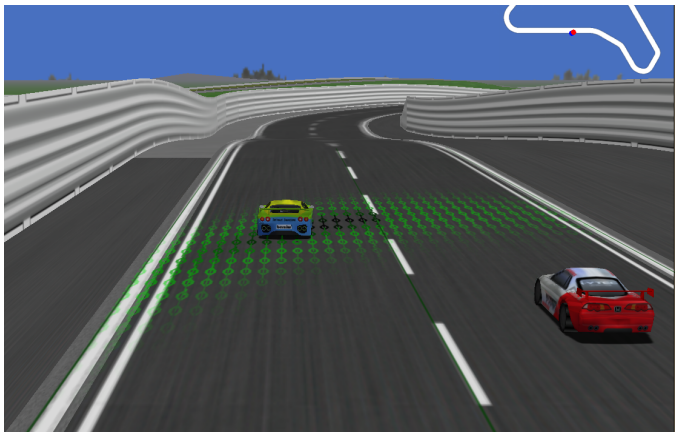


Figure 6.1: The scenario created with the TORCS Simulator. The yellow vehicle is overtaking the red ego-vehicle. The grid nodes are projected into the image. The line within the nodes shows the estimated velocity direction. (Parameters other than those stated below are used.)

curve (begin of phase 4). In time step 80 the ego-vehicle leaves the curve. All highway segments have circular or straight shape, no clothoids are used. This fact makes it more difficult for any filter algorithm to keep track of the observed vehicle. In the following subsection we vary the parameters of the filter and evaluate the tracking performance.

6.1.1 Effects of the Grid Cell Size

In Sec. 3.2.1 the grid cell size was introduced. In this subsection we want to evaluate the tracking error depending on the grid cell size d.

In the shown example (Fig. 6.2) with $d = 0.25m$ the average estimated state equals the ground-truth state of the observed vehicle. This indicates that the filter works correctly. The error of the average estimated state is at all times below $\frac{d}{2}$ (cf. Fig. 6.3 and Fig. 6.4. In this figures the error is analyzed for the direction towards the side x_0 and towards the ego-vehicle nose x_1 as described in Fig. 3.1). The errors rise with increasing d. For $d = 1.5m$ the standard deviation increases significantly, indicating that the tracking becomes more and more unstable. The grid is now too coarse to cover the vehicle dynamics. For $d = 2.5m$ discretization errors become clearly visible in the position estimate. The probabilities are gathering in a few grid cells and the discrete grid cell positions are in some cases not suited to represent the position state with this low number of cells. When the ground truth position lies by chance on a position that is easy to represent by the grid, the errors and the standard deviation will be low (e.g. $d = 3.0m$ at time 60 to 80). For $d = 3.5$ or $d = 3.75$ the ground-truth position is not followed anymore in a reliable manner. For $d = 4.0$ approximation errors

exceed the working range of our approach and no position estimate was possible anymore.

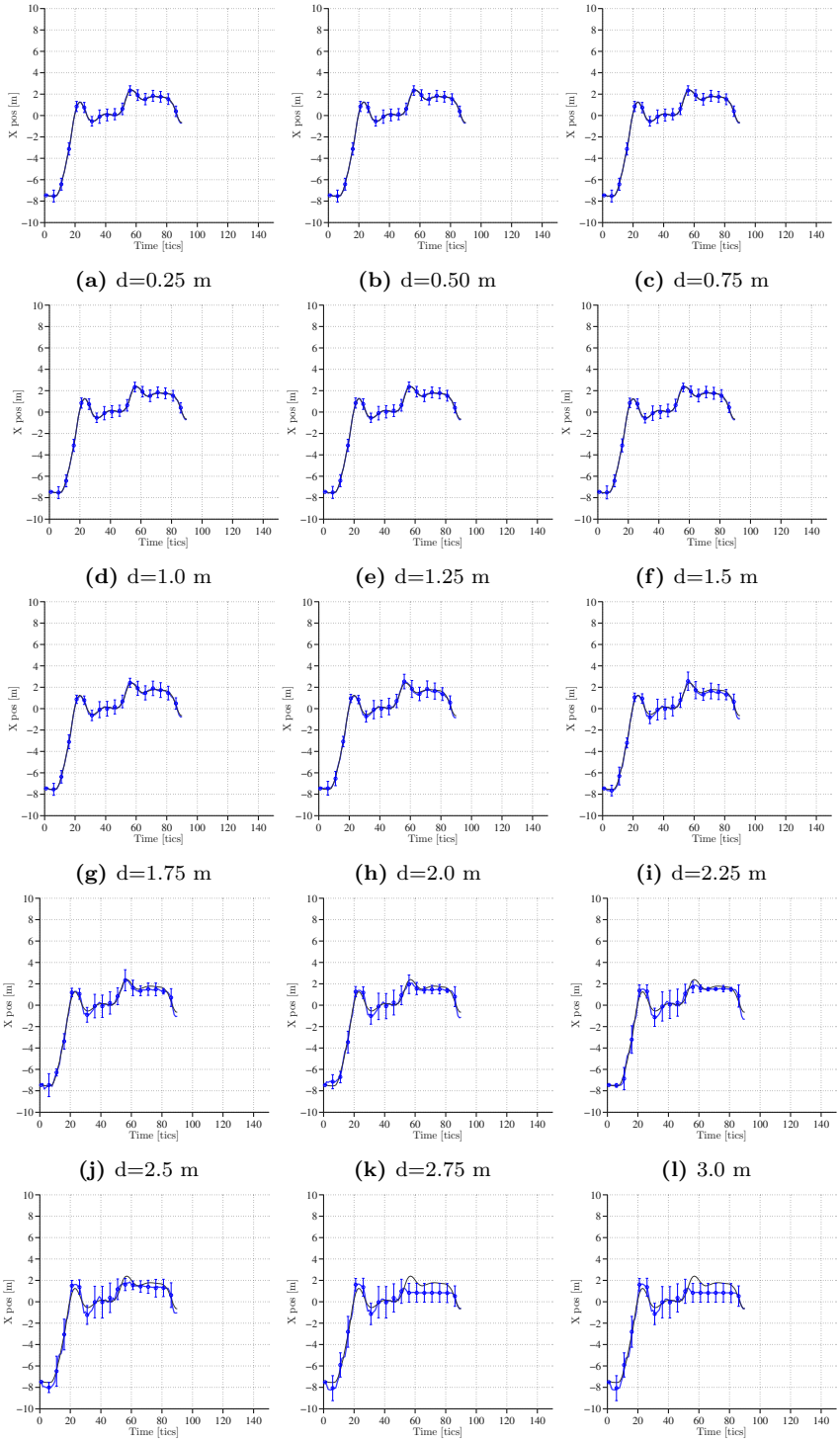
6.1.2 A Comparison of the Integration Methods

In Sec. 3.2.3 two different integration techniques in order to sample the PDF were introduced. Here, we analyze the effects in the TORCs simulation scenario from above.

The experiments are executed with different granularity d and different standard deviation in the transition model σ_β. A different standard deviation in the transition model does not only influence the anticipated behavior (e.g. high σ_β anticipates direction changes better while small σ_β values deliver smoother results when tracking during a straight driving phase (cf. Sec. 4.1)), it also influences the discretization errors. The reason is simple, a high σ_β leads to wide and smooth transition functions while a small value leads to steep transition functions. Steeper functions are more error prone to discretization and need a finer sampling. A theoretical excursus is done in Sec. 6.2.

The evaluation shows that with a fine grid granularity and smooth transition models there is no significant benefit of the better integration method. But the coarser the granularity and the steeper the transition function, the more important are more sophisticated integration methods. Figure 6.4 compares the evaluation runs using the piece-wise linear integration with the piece-wise constant integration. It is clearly visible that the tracking with the simple integration method performs worse when the prediction variance is small ($\sigma_\beta = 0.07071$). With $d = 1.00$ or greater the sampling of the prediction function with the piecewise linear integration clearly outperforms the piecewise constant integration. With $d = 2.0$ the track is totally lost without the improved integration technique.

We evaluate the integration methods with the same settings but $\sigma_\beta = 0.04$ instead of $\sigma_\gamma = 0.07071$. The differences in the tracking between the different integration techniques are smaller within the same granularity. There are two reasons for that. First, as already mentioned, the discretization error is smaller for smoother functions. Second, due to the Bayesian fusion, the influence of the prediction step becomes smaller in comparison to the filtering step if the standard deviation of the prediction function is higher. Both effects are extensively discussed in later chapters. As a result, the error becomes obvious at coarser granularity ($d = 2.0$) in comparison to $d = 1.0$, when using the higher standard deviation $\sigma_\beta = 0.04$ (compare Fig. 6.4 with Fig. 6.5).



(a) d=0.25 m **(b)** d=0.50 m **(c)** d=0.75 m

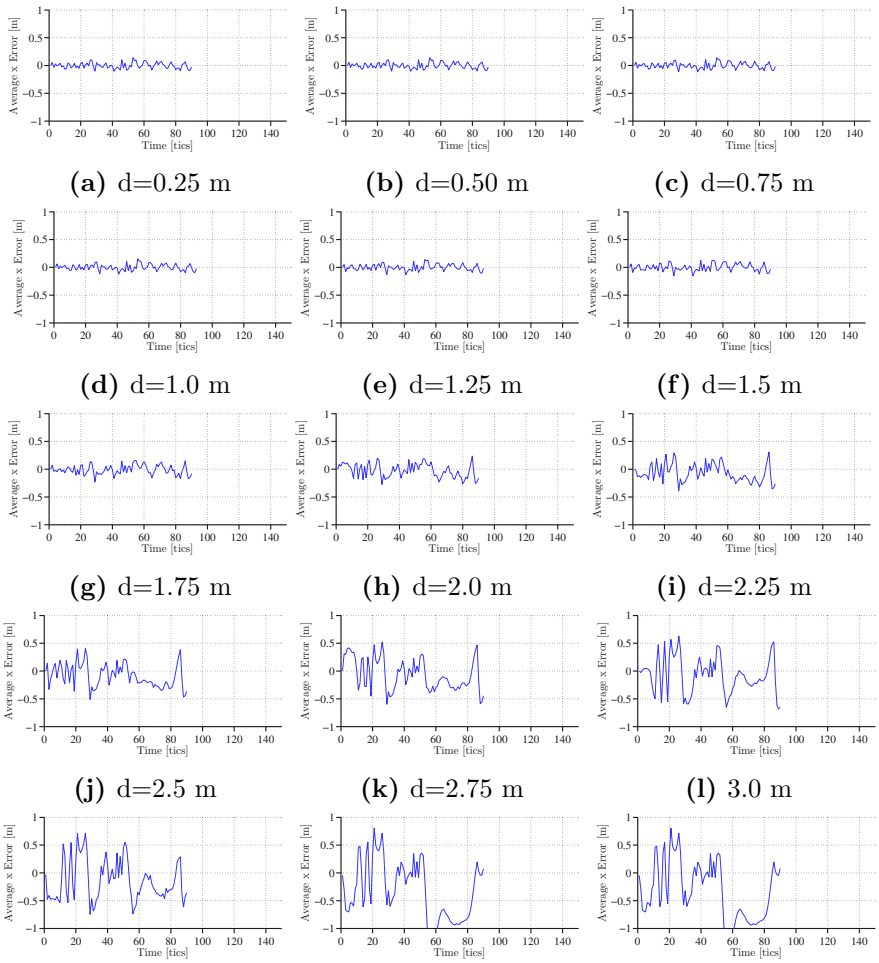
(d) d=1.0 m **(e)** d=1.25 m **(f)** d=1.5 m

(g) d=1.75 m **(h)** d=2.0 m **(i)** d=2.25 m

(j) d=2.5 m **(k)** d=2.75 m **(l)** 3.0 m

(m) d=3.25 m **(n)** d=3.5 m **(o)** d=3.75 m

Figure 6.2: Tracking of the x-position with different grid cell distances d. The blue line is the estimated position averaged over 100 runs. The black line is the ground-truth position. d is incremented in 0.25m steps. With d=4.0m the probability collapses and no tracking is possible.



(a) d=0.25 m **(b)** d=0.50 m **(c)** d=0.75 m

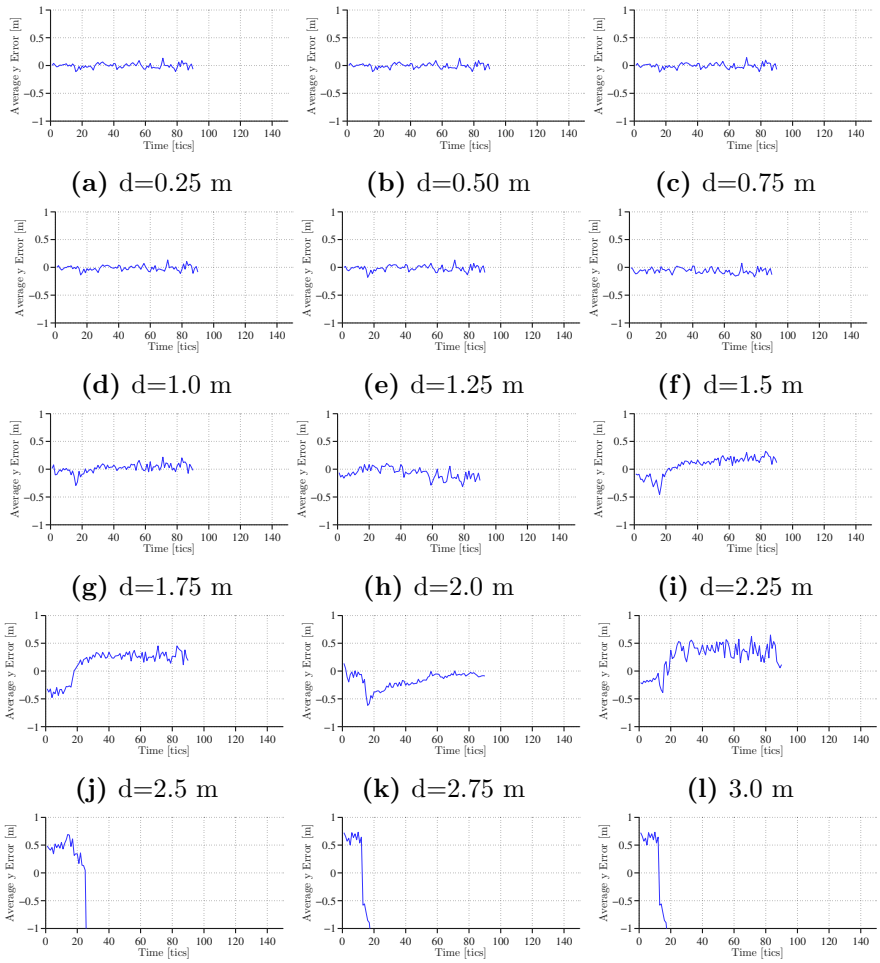
(d) d=1.0 m **(e)** d=1.25 m **(f)** d=1.5 m

(g) d=1.75 m **(h)** d=2.0 m **(i)** d=2.25 m

(j) d=2.5 m **(k)** d=2.75 m **(l)** 3.0 m

(m) d=3.25 m **(n)** d=3.5 m **(o)** d=3.75 m

Figure 6.3: The average error of the x_0-position with grid cell distances d. d is incremented in 0.25m steps. With d=4.0m the probability collapses and no tracking is possible.



(a) d=0.25 m **(b)** d=0.50 m **(c)** d=0.75 m

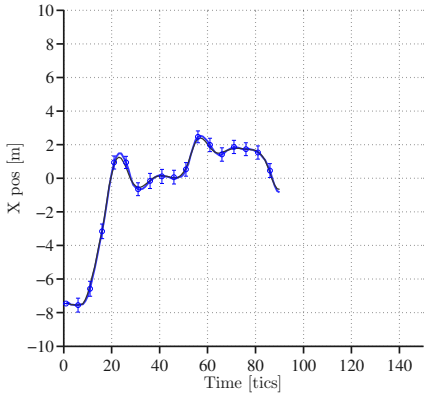
(d) d=1.0 m **(e)** d=1.25 m **(f)** d=1.5 m

(g) d=1.75 m **(h)** d=2.0 m **(i)** d=2.25 m

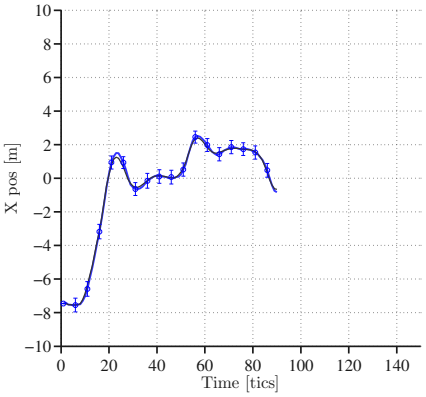
(j) d=2.5 m **(k)** d=2.75 m **(l)** 3.0 m

(m) d=3.25 m **(n)** d=3.5 m **(o)** d=3.75 m

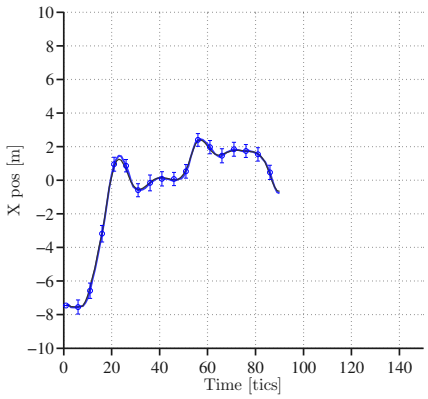
Figure 6.4: The average error of the x_1-position with grid cell distances d. d is incremented in 0.25m steps. With d=4.0m the probability collapses and no tracking is possible.



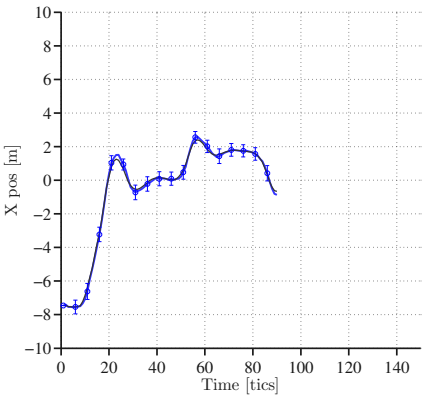
(a) Piecewise linear integration (d=0.25)



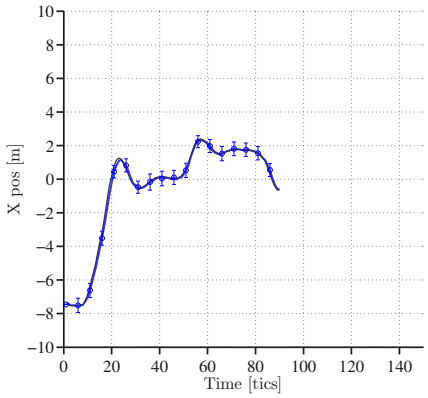
(b) Piecewise constant integration (d=0.25)



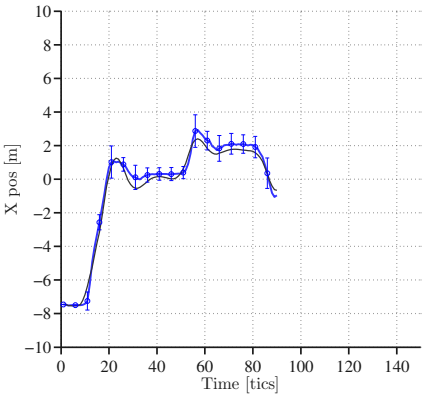
(c) Piecewise linear integration (d=0.50)



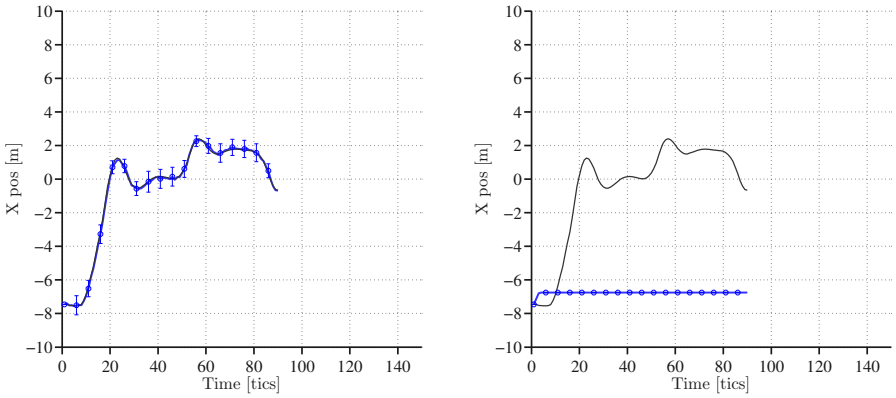
(d) Piecewise constant integration (d=0.50)



(e) Piecewise linear integration (d=1.00)



(f) Piecewise constant integration (d=1.00)



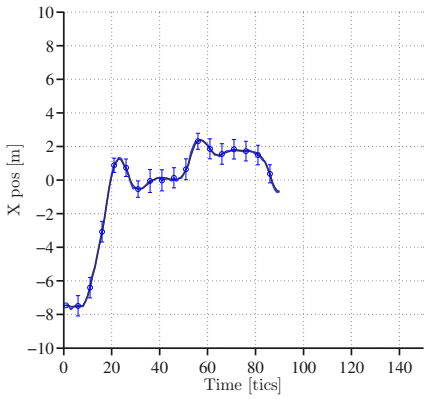
(a) Piecewise linear integration (d=1.50)

(b) Piecewise constant integration (d=1.50)

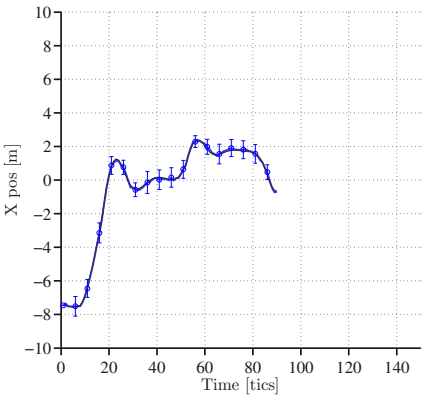
Figure 6.4: Comparison of the two integration methods with a prediction function standard deviation of $\sigma_\beta = 0.07071$

Further results from this evaluation

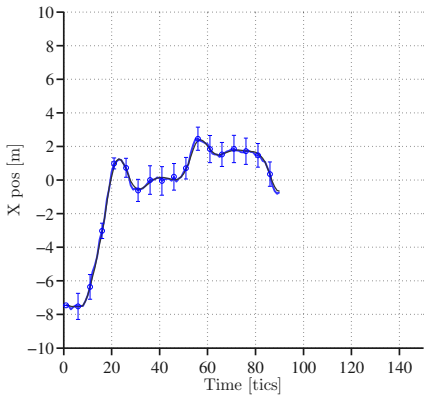
Another interesting effect, but irrelevant for the integration comparison is seen already for $d = 0.25$. Abrupt direction changes are not tracked optimally for $\sigma_\beta = 0.07071$. The average estimate corrects the direction too late in time step 22 and deviates from the ground truth position. But in the time interval with the smooth and straight movement, the $\sigma_\beta = 0.0.07071$ runs (cf. Fig. 6.4) outperform the runs with $\sigma_\beta = 0.4$ (cf. Fig. 6.5), which is clearly visible by the smaller deviations of the individual runs. This is exactly the result a Bayesian filter is expected to provide, since a higher σ_β filters the sensor input less strongly.



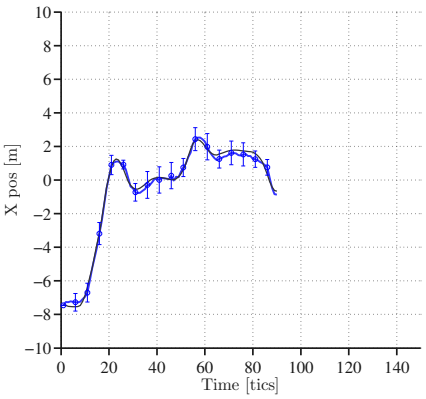
(a) Piecewise linear integration (d=1.50)



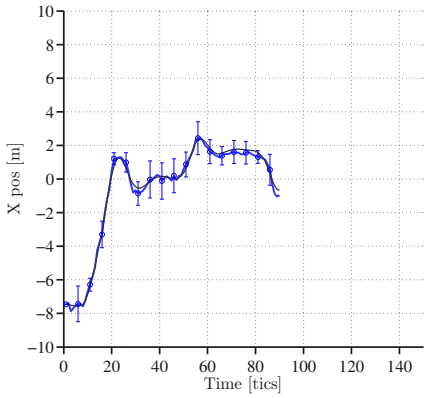
(b) Piecewise constant integration (d=1.50)



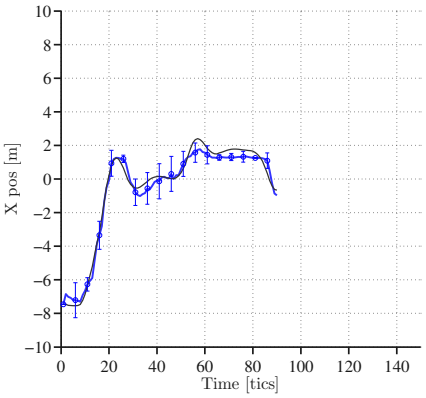
(c) Piecewise linear integration (d=2.00)



(d) Piecewise constant integration (d=2.00)



(e) Piecewise linear integration (d=2.50)



(f) Piecewise constant integration (d=2.50)

Figure 6.5: Comparison of the two integration methods with a prediction function standard deviation of $\sigma_\beta = 0.4$

6.2 Analytical BHF Error Evaluation and Extreme Case Studies

As already discussed in section 2.2.4 the unlimited state-space of the real-world has to be converted into a finite number of variables. Therefore all (continuous) PDFs have to be sampled at discrete points. A brief glimpse on these effects was already presented by the evaluation in Figs. 6.2 and 6.4 with a preliminary discussion in Sec. 3.2.1. Now we focus more on the theoretical background of the different types of BHF errors. First an analytical inspection is done and second an extreme case evaluation.

6.2.1 Sampling Errors

For one dimensional grids the sampling errors are well-defined by different techniques. The Nyquist-Shannon sampling theorem for example tells us that a signal with limited frequencies $< f_{max}$ needs a sampling of at least $2 \cdot f_{max}$ in order to reconstruct a signal exactly [56]. The frequency is here considered over time and implies that the observed function consists of periodic oscillations. While the time can be easily interpreted as spatial dimension to transfer the theorem into our field it is hard to assume that the PDFs are of periodic nature. In the end the value for an error analysis of the BHF is weak. High frequencies correspond with high changes of the function value and its derivatives. In other words, the higher the frequencies the "less smooth" is a function. The number of necessary sampling points will be higher for a "less smooth" function.

A more sophisticated error analysis can be done for one-dimensional functions using the theory about integration errors. The one-dimensional integration pendant to the piecewise-constant integration used here is referred to in the literature as the rectangle method. The trapezoidal rule is the one dimensional pendant to the piecewise linear integration and the special case of the Newton-Cotes integration using 2 sampling points per integration interval $[a, b]$. The integration interval is equivalent to the one dimensional grid cell volume. The approximate integration is done by Eq. 6.1.

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \frac{b-a}{2}[f(a) + f(b)] \tag{6.1}$$

The integration error in an individual grid cell is defined as the difference between the numerical integral over the approximating trapezoid function $p(x)$ and the integral over the real function $f(x)$ (cf. Eq. 6.2). Note: In contrast to the literature, the overall integral over all grid cells is not relevant for our application since we know that f is a PDF and therefore its overall integral is 1.

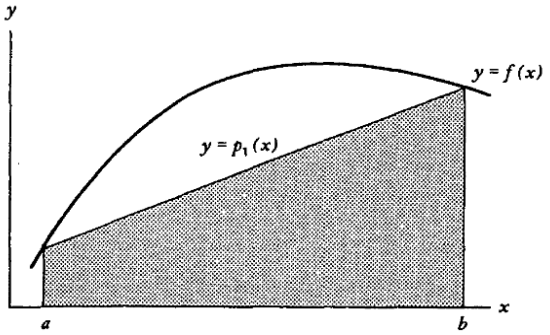


Figure 6.6: The first order Newton-Cotes integration. [11]

$$E(f) = \int_a^b f(x)dx - \int_a^b p(x)dx \tag{6.2}$$

The resulting error for trapezoidal integration intervals is stated in Eq. 6.3 (cf. proof in [11]).

$$E(f) = -\frac{(b-a)^3}{12} \cdot f''(\eta) \quad \eta \in [a,b] \tag{6.3}$$

The error depends on the second derivative of the function. PDFs with low $|f''|$ values are less error prone than PDFs with high $|f''|$ values.

Somewhere within the interval there is an η for which the second derivative becomes highest. In the PDFs used by us the exact solution of the integral and thereby η and the second derivative at its position is not known. Therefore this equation can be used only to give the order of the error term. Tables 6.1 and 6.2 show the integration errors for the trapezoidal integration over a Gaussian PDF for the integration interval around $\eta = 0$, were the f'' has its absolute maximum. As illustrated in Fig. 6.6 and in Eq. 6.3 the negative $f''(\eta = 0)$ leads in our evaluation to a positive error, which signals an underestimation of the probability density at the area around η.

Table 6.1: The maximum integration error when sampling a Gaussian with $\sigma = 1.0$. The accurate integrated density is 0.398.

Interval size	0.25	0.50	0.75	1.0	1.5	2.0
Density	0.386	0.352	0.301	0.242	0.129	0.054
Error	0.012	0.046	0.097	0.156	0.269	0.344

The order of the error can be further decreased by using higher order Newton-Cotes equations. E.g. when using the Simpson rule the error depends on the fourth derivative of the PDF. But the number of sample

Table 6.2: The maximum integration error when sampling a Gaussian with $\sigma = 0.5$. The accurate integrated density is 0.798.

Interval size	0.25	0.50	0.75	1.0	1.5	2.0
Density	0.704	0.483	0.259	0.108	0.009	0.000
Error	0.093	0.313	0.539	0.690	0.789	0.797

points increases with each order and the method becomes finally unstable due to negative sample weights. The literature advises that it is then better to reduce the cell size instead or to use more sophisticated methods like the Gauss-Legendre quadrature instead. These more sophisticated methods need varying sample point positions within the cells.

Neither the Newton-Cotes equations nor the Gauss-Legendre quadrature can be generalized to multivariate inputs. Although Multivariate integration is possible for a subset of functions, e.g. for periodic integrands [66]. That also means in a strict interpretation that the error term cannot be generalized to our two dimensional estimate PDF. Independent from that, the integration technique itself can be specified as a Monte-Carlo method with non-random fix sample points. With the particle filter (Sec. 2.2.2) we already mentioned a Monte-Carlo approach used for integrating the PDF. The advantage of our technique is the practical improvement in cases with critical cell distance d (cf. Fig. 6.4) at a very small computation overhead.

When looking at equation 6.3, it is clear that for a constant error $E(f)$ the designer of a BHF should keep in mind that the cell distance d should be reduced with rising second derivation of the PDF. That means that steeper prediction models, e.g. models with smaller σ_β and σ_γ need a smaller grid distance d. And even more so, the steepness of the sensor model influences the error and needs an adaption of d, since it is sampled with the constant piecewise function. In this subsection it becomes clear that there is no exact rule to determine the error in the two dimensional MDBHF. However, in the next subsection we will give rules of thumbs on the right parameter setting.

6.2.2 Error Types

Analogous to the extreme case analysis for particle and Kalman filters in [65] we will now examine the errors of the BHF and MDBHF on the basis of extreme parameter settings.

Boundary Problems

Usually the space covered by the BHF should be chosen in a way that the borders are far away from the relevant area. Otherwise the sensor sampling and the prediction sampling are distorted. As an extreme example imagine a prediction function with its expectation value outside of the covered space.

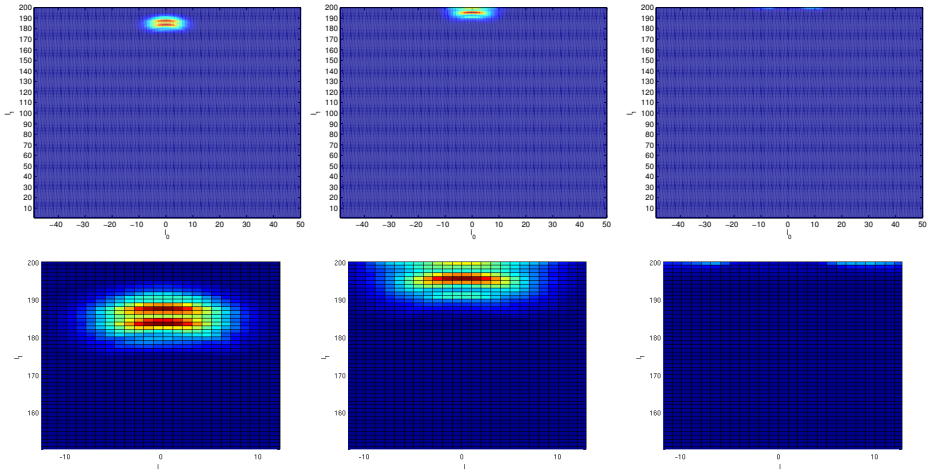


Figure 6.7: The graphs show the estimate in a series of time frames. The axes denote the cells ($d = 0.25$). In the second frame the Boundary effects become visible as a beginning boundary jam. In the third graph the probabilities begin to drift sidewards in the top row. The second row shows the zoomed versions of the images.

Since no sample points are near the expectation value the sampled density in that area is zero. There will be some parts of the prediction function with low probability densities inside the covered space. Due to the normalization the whole probability mass will flow into these areas inside the covered space. In result, a jam of probability values occurs next to the border. The direction of the probability flow is diverted to unlikely directions, which lie within the covered space as only left option.

We have created an artificial situation in which a vehicle approaches the border with $||v_{Observed}|| = 22m/s$ while the ego-vehicle stays exactly behind the observed vehicle with $||v_{Ego}|| = 0$. In order to highlight the error no boundary cells are used. The PDF of the estimate is illustrated in subsequent time-steps in Fig. 6.7.

Small Velocity to Grid Distance Ratio

This subsection targets the problem of small absolute velocity to grid distance ratios. Or to be more exact, the problem of small (flow) distance to grid distance ratios $\Lambda = \frac{\Delta T||\mathbf{v}||}{d}$. The problem was already mentioned in Sec. 3.2.1 and the effects became clear in the evaluation in Sec. 6.1.1. The absolute velocity error depends on the grid distance d with fixed time step size ΔT. With smaller Λ the relative error in the velocity dimension increases, while the absolute velocity stays constant. That means that low velocities are more prone to discretization errors than high velocities.

This becomes clear with an example: We observe a vehicle driving

straight into l_1 direction with a velocity of $||\mathbf{v}_{observed}|| = 5m/s$. The ego-vehicle follows with the same velocity $||\mathbf{v}_{ego}|| = 5m/s$. As a result the relative position of the observed vehicle does not change over time. The sensor error was set to zero in order to avoid disturbing effects induced by noise. Notice that this means not that the sensor model assumes zero noise, too. While tracking the following will happen: Since we are dealing with probability distributions, it is totally clear that some probabilities will flow from one cell to the other. They will not flow into a single cell. When a probability mass is flowing from one grid cell to the neighboring cell in l_0 direction, that flow represents movement vertical to the ego-vehicle nose. With a cell distance $d = 0.5m$ and $\Delta T = 0.1s$ this movement has a velocity of $5m/s$ into l_0 direction. In the chosen example the ground truth velocity was already $5m/s$. The velocity increases according to Pythagoras to $7.071m/s$ as the minimum value. And a probability flow hitting the cell above the cell in $l1$ direction, is resulting in the velocity $10m/s$. In a fine grid this flows would be assigned a very low probability value, since there are other sampling points between the starting cell and the destination cell, which produces the $10m/s$ flow. If there are no other cells in between, however, the probability flow with $10m/s$ is weighted too high. This kind of discretization errors produces a high error in the relative velocity and leads to an chaotic velocity behavior in the PDF. This is depicted in Fig. 6.8.

It is clearly seen that the absolute velocity is too small in comparison to the grid cell size, since the PDF has a very small width. The small width is produced by prediction functions with small extent. The possible (absolute) velocity change produces again a distribution with small width. Due to the coarse sampling, the sampled prediction function is even steeper than the real prediction function (cf. Sec. 6.2.1. One sampling point is due to the relative velocity of zero directly at the expectation value = maximum value of the prediction function. In the range of $f'' < 0$ of transition function, the next neighbors are underestimated.) Probability masses with relative high speeds depart from the sensor position in each direction. High velocities to the side gain influence and the estimate starts to lurch from the left to the right around the sensor input.

In Fig. 6.9 the correcting effect of the sensor input is not yet incorporated. It shows the predicted estimate. The discretization effect with the departing flows is seen there even better.

The example above was an extreme case with $\Lambda = 1$. Of course even smaller ratios are possible. The value was chosen since the effects are clearly visible directly in the estimate PDF, even for the untrained eye. But even with a higher velocity of $v = 11m/s$ and $Lamda = 2.2$ the effect influences the estimate clearly. It shows that at these velocities the sampling needs to use at least a cell distance $d = 0.25$ or smaller. A strategy to reduce that error could be to increase the time step size ΔT dynamically like in the sample variance problem of the particle filter (cf. Sec. 2.2.2). Alterna-

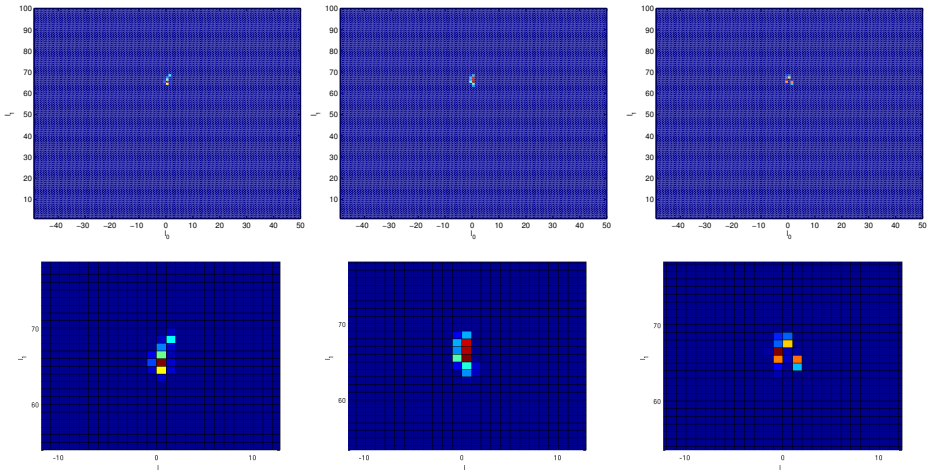


Figure 6.8: The graphs show the estimate in a series of time frames. The axes denote the cells ($d = 0.5$). The velocity is too small to create wide prediction functions and therefore the whole PDF collapses. The second row shows a zoomed version of the images.

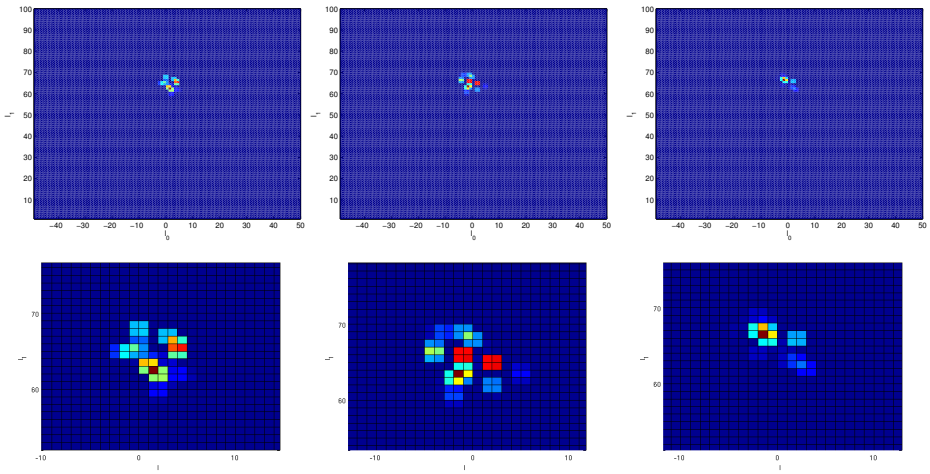


Figure 6.9: The graphs show the predicted estimate in a series of time frames. The axes denote the cells ($d = 0.5$). Already in the first frame the discretization effects are visible. In the second graph the probabilities begin to drift sidewards. The second row shows a zoomed version of the images.

tively vehicles with small absolute velocities could be marked as static or near-static objects. Then they can be tracked by a Kalman filter. The disadvantages of the Kalman filters are small as long as the vehicle velocities are small, since the movement can be approximated as linear movement. Note that this problem is relevant for small absolute velocities and not dependent on the relative velocity of the observed vehicle. In the example above we set the relative velocity to zero in order to make the explanation easier to understand.

Steep PDFs

The previous error types can generate serious distortions of the PDF, but in the end the PDFs are corrected by the new sensor input in the filter step. Are there errors possible that the filtering step cannot correct? The example illustrated in this section gives the answer.

One of the principles of the Bayesian filtering is that with decreasing variance in the transition model, the influence of the sensor input decreases. This is obvious since the variance in the model is something like the trust into the expectation value. Transition to cells away from the expectation value become more and more unlikely with decreasing variance.

The extreme case here uses a similar mechanism, but due to discretizaton errors or pruning of small flows the effect increases even further. The probabilities can only flow into certain cells, everything else is pruned. This happens by the pruning of the prediction function by the selective propagation (cf. Sec. 3.2.2). As a result, the whole probability is flowing with a certain velocity, which may deviate strongly from the movement of the observed vehicle. The missing variance in the transition model disallows velocity corrections of the estimate.

Figure 6.10 illustrates such a situation. Every probability is flowing into a certain direction, no variance in the direction occurs.

In a working BHF with PDFs smooth enough for the grid granularity, this will never happen. The sensor would weight the cell density which is nearer to the sensor input higher. Thereby the flow producing the prior probability at this cell will gain importance in the next step and thereby the direction of the whole estimate changes towards the sensor input over time. But due to the steep transition function and the selective propagation there are no flows possible which target the right direction. There need to be at least a transition probability $\geq p_{pruning}$ to correct the situation by setting a prior probability at the cell, where a vehicle doing a direction change should be in the next time step. But with a grid coarse-enough, the effect of the leaving probability is never stopped, since even small corrections are not possible, since due to the discrete nature of the grid no sampling point is located where the small direction change could be created.

While Fig. 6.10 shows the effect of a small variance to grid distance ratio

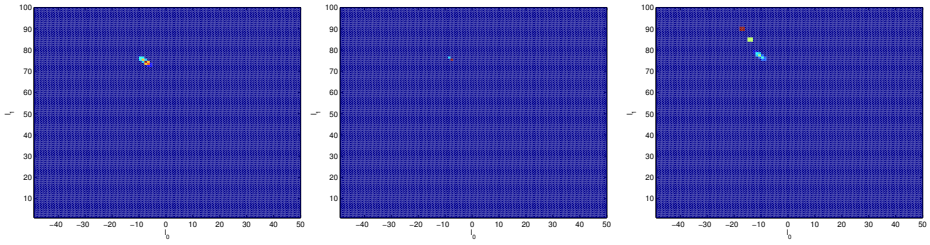


Figure 6.10: The graphs show the estimate in a series of time frames. The axes denote the cells ($d = 0.5$). The estimate drifts to the left while the sensor inputs produced by the observed vehicle leaving with $||v|| = 11.1m/s$ all occur on $l_0 = 0$. The effect of drifting is clearly visible, therefore no zoom is provided.

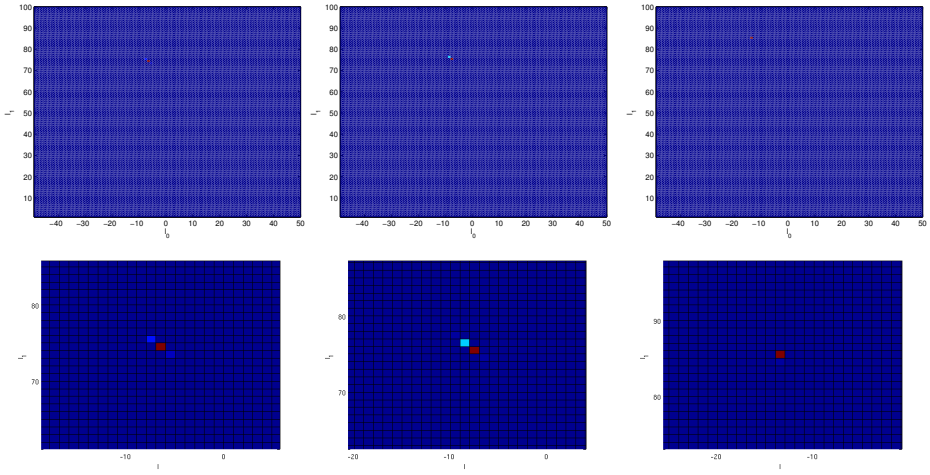


Figure 6.11: The graphs show the (non-normalized) predicted estimate in a series of time frames. The axes denote the cells ($d = 0.5$). The second row shows a zoomed version of the images.

in the direction in the estimate Fig. 6.11 shows the error in the predicted estimate. The small (non-normalized) activities show that the sample points do not hit the high values of the transition PDF.

When the variance in the **sensor** model is too small as well the whole PDF may collapse to zero since no overlap between the sensor model and predicted estimate is left. This unwanted effect may be resolved by setting a very small propability greater 0 to each cell in the grid. This will happen at the cost of loosing the computational time improvement by selective updating. The problem is comparable with the sampling bias in the particle filter limitations (cf. Sec. 2.2.2).

6.2.3 Summary

In this subsection we have analyzed the characteristic error types of the BHF approaches. It should be clear that the task of transforming a continuous distribution into a discrete distribution is not possible without discretization errors (cf. Sec. 2.2.4). Errors can occur when the relevant area of the distribution is near the grid boundary, when the absolute velocity of the observed vehicle is too small in comparison to the grid cell distance or when the sensor model is too steep. The errors can be avoided or reduced by using boundary cells and by choosing the space covered by the BHF in a reasonable manner, by handling slow moving vehicles using an adaption of the time step size or by using another filter for slow moving vehicles, or by using smaller grid cell sizes and a probability greater than 0 in each cell in the grid.

In Sec. 2.2.2 we have mentioned typical errors occurring in other discrete PDF representations (e.g. when using particles or Gaussian distributions).

The number of error types and the quantity of the error may seem rather high. But in the end this is the price of computability. When lower errors are desired the number of sample points need to be increased causing higher computational effort. Other probabilistic approaches have similar problems and even more so the non-probabilistic approaches, which hide the uncertainty in the calculation by not dealing with probabilities at all. Even the clear Gaussian shape produced by approaches that are using Gaussian-representations gives elusive safety. Regardless how bad the filter is working the output is always a nice looking Gaussian shape. BHF representations are more honest from that perspective.

6.3 Comparison of Compensated Tracking with non-Compensated Tracking

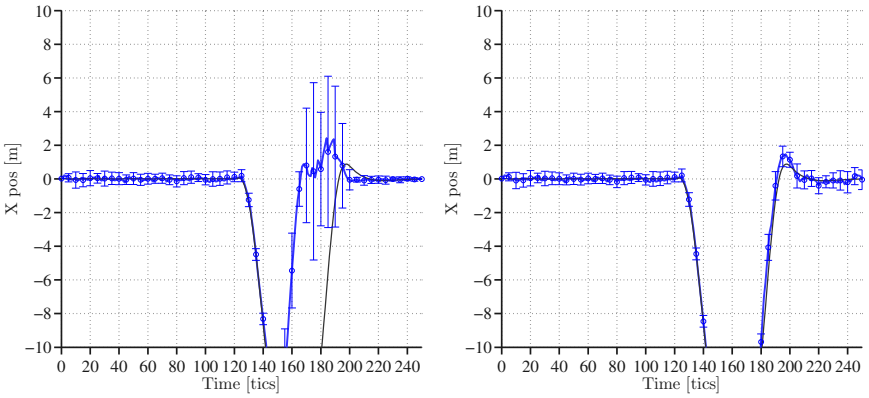
In order to show the obvious benefits of the ego-motion compensation in rotated coordinate systems (introduced in Sec. 4.4.1) for a tracking task we set up an evaluation with Carmaker. The results will show that without ego-movement compensation serious errors occur in curves.

We tested the errors in the tracked position in a scenario in which our ego-vehicle follows an observed vehicle on the same lane into a curve with a radius of 30 m. A smaller curve radius would lead to the result that the observed vehicle is leaving the sensor range to the side (cf. experiment in Sec. 7.3.2). Both vehicles (the ego-vehicle E and the observed vehicle O) drive with a velocity of $v_{Ego} = v_O = 50\ km/h$ in order to simulate an inner-city tracking task. The distance between both vehicles is about 25 m. The tracking error induced by the ego-motion is clearly visible in the evaluation graphs 6.12. At the same time we track the progression of ego-yaw rate ω.

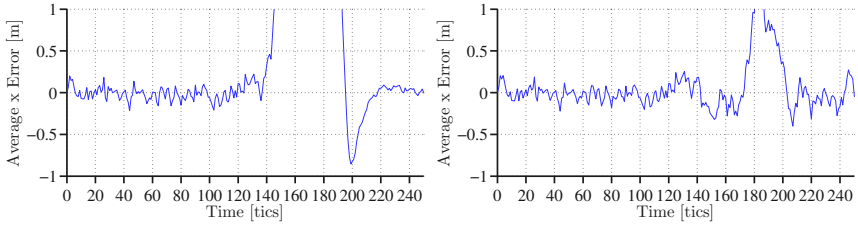
This enables the reader to discover the influence of ω in the tracking results.

The iteration time step size ΔT was set to $0.1s$. At time step (tic) 125 vehicle O enters the curve and in time step 138 vehicle E follows. With increasing yaw rate the error in the x and y direction increases (referred as x_0 and x_1 direction elsewhere in this thesis). When reducing the yaw rate at the end of the curve (time step 185) the x error overcompensates in the negative direction and does not predict the overshoot of vehicle O in the end of the curve. The tracking with enabled ego-movement compensation shows in all stages smaller errors. Only the overshoot when vehicle O leaves the curve is overestimated by the prediction. In order to reduce the remaining error in the compensated tracking results the prediction model is improved by an attractor algorithm which is introduced in Chp. 7. The attractor model allows to model the driving behavior in curves in a more accurate way. In section 7.3 we use the full-blown ICUBHF approach on the same scenario again in order to show the benefits of the attractor approach using ego-movement compensation.

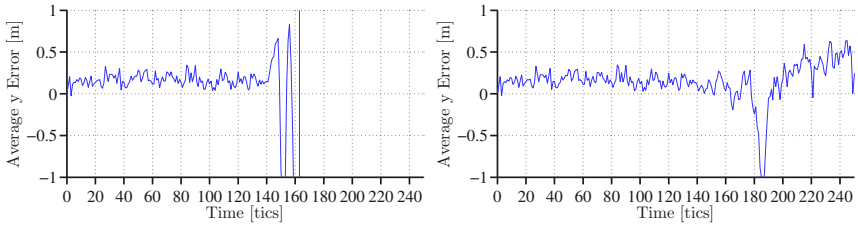
This result and also the referenced results clearly show that the ego-movement compensation is necessary for inner city tracking tasks. In inner city tracking tasks the ego-vehicle will in most cases not drive straight. It drives in curves, turns at intersections or evades parked vehicles or pedestrians standing besides the road.



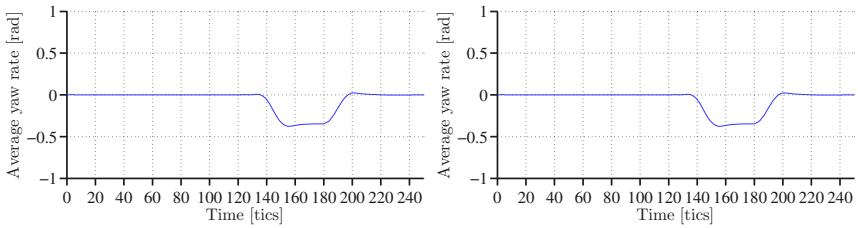
(a) Without ego-compensation **(b)** With ego-compensation



(c) Without ego-compensation **(d)** With ego-compensation



(e) Without ego-compensation **(f)** With ego-compensation



(g) Without ego-compensation **(h)** With ego-compensation

Figure 6.12: Tracking of the x-position during a left curve with radius = $30m$. In the first row the blue line is the estimated x-position and the black line is the ground-truth x-position. The next two rows show the average error in x and y direction. The last line is the yaw rate. The left figures show the tracking with enabled ego-compensation and the right figures show the tracking without ego-compensation.

Chapter 7

Model Driving Behavior

This chapter is about how to model the driving behavior. We begin by discussing why it is necessary to model the driver behavior of other vehicle drivers. ADAS systems profit twice, on the one hand the prediction can be improved by incorporating the behavior model and thereby improving the vehicle tracking. In addition, the perception of the ADAS can be compared with the modeled behavior in order to check if the detection matches a plausible behavior model. We discuss the behavior categorization itself in Part III.

The behavior model is based on the observed context. We know that in certain situations drivers execute a certain behavior with a high probability (cf. Fig. 7.1). The knowledge about this is gained by the human driver due to experience. The idea is to introduce driving models in our MDBHF, which gives us stricter assumptions (4.2) about the other vehicles than the kinematic model introduced in 4.1.2. Every driver knows, with a very high confidence, that an oncoming vehicle on the street will probably stay in its lane.

There are two different ways to incorporate context, and thereby behavior of vehicles. The context can either be incorporated into the filter step or into the prediction step.

The incorporation into the filter step is the state-of-the-art approach, which we will discuss in the following section, while the incorporation into the prediction step has hardly been attempted before to the best of our knowledge. We developed an attractor algorithm in order to find a solution, which is as generic as possible (cf. patent [4]). The attractor approach is the topic of the subsequent section.

7.1 Using Context Information in the Filter Step

A possible solution to the problem of incorporating context information into the Bayesian filter is altering the filter step, but that is not suited to inferring

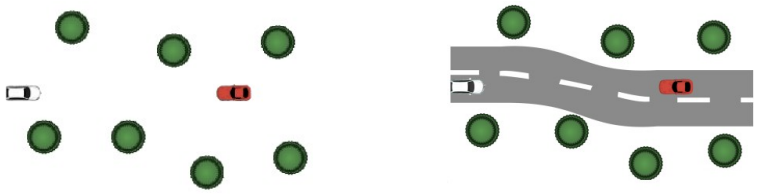


Figure 7.1: Without context the observer would assume a crash. With context behavior leading to a crash becomes unlikely. Human drivers assume that (oncoming) vehicles stay in their lane, and they have a high confidence in this assumption. The image was originally published in [53]

.

behavior. The context can be used in the filter step in order to define states with a higher a priori probability of occurrence and states with a lower a priori probability.

The advantage of this behavior incorporation is that it is easy to implement. Using the Bayesian equation the measurement model has to be multiplied by the prior vehicle state $P(X)$ divided by the prior of the sensor measurements $P(Y)$ in order to receive the inverse measurement model (Eq. 7.1), to reason about the state X. Note that the PDF $P(Y)$ can account for a sensor bias and that the a priori probability of a certain state X is given by $P(X)$.

$$P(X|Y) = P(Y|X)\frac{P(X)}{P(Y)} \tag{7.1}$$

To implement the context in the filter step, the Bayesian filtering equation has to be altered in order to draw $P(X)$ out of the normalization term. The prior state estimate $P(X)$ is no longer a uniform distribution but can be altered to match the context. For example vehicles will not occur on trees, vehicles will not be on the top edge of the image in camera images, and there is a lower probability of vehicles being next to the street. There are several state-of-the-art approaches that are incorporating the context in the filter step (e.g. [26, 8]). Others are using it in a novel, but questionable way in the filter step [58], causing an inhibition of the necessary divergence of the estimate. A rare approach is to use the context in the prediction and in the filter step [31].

The usage of a priori assumptions in the measurement model has some drawbacks. One is that the model's assumptions may be too strong and correct measurements may be ignored due to a low prior, but that is also the

case when using assumptions on the prediction model. The main drawback is that it cannot model the behavior itself. In simpler terms, the behavior needs to be modeled in the prediction step, since the prediction step tells the Bayesian filter how a state can change from one time-step to the other, just like the planned behavior induces an action, which effects a certain position change. The position is just the effect of the position change. When there is a red light in front of us, it does not mean that there is only a small a priori probability that there is a vehicle behind that red light. It rather means that there is only a small probability that a vehicle in front of the red light will cross the red light.

7.2 Using Context Information in the Prediction Step

In the prediction step we can determine which state transitions are more likely than others. As introduced in Chp. 4 the vehicle kinematic determines which states are reachable from the current state, but kinematic reachability is not a strong assumption on the input. Drivers will prefer certain states of the set of reachable states. As a simple example, in most cases drivers will stay in their lane or stop in front of a red traffic light. That means that most probability flow should stay within the lane it originated, and that the velocity of the flow should be reduced when approaching a red light signal.

There are different ways possible to influence the probability flow:

1. Cut unwanted probability flow.
2. Change the probability flow.
3. Change the expectation value in each grid cell
4. Set an overall goal for the whole estimate

Each of these models is detailed in the following subsection and the general principle is shown in Fig. 7.2. In the subsequent subsection we introduce the different parts of the generic ICUBHF approach, which uses attractors in order to model the trajectory that a vehicle would drive.

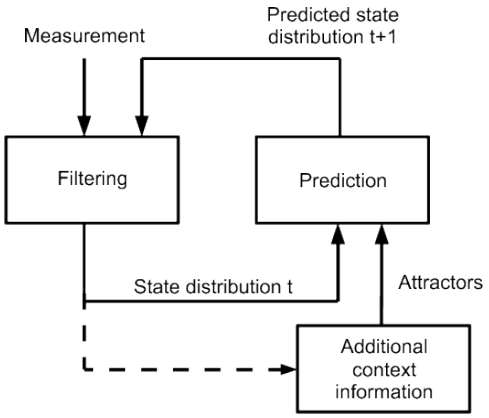


Figure 7.2: The figure shows how the overall information flow is adapted in the Bayesian filter by the context incorporation in the prediction step. Originally published in patent [4] .

7.2.1 Methods Influencing the Prediction Model

The prediction model creates a probability flow which represents the movement the vehicle executes from one time step to the next time step. There are different points at which we can alter the prediction model and therefore the probability flow.

Cut Unlikely State Transitions

One of the easier ways to influence the probability flow is to cut unlikely flow. Unlikely flow may, for example, be probability flow that crosses a lane marking. Since the crossing of lane markings should be less probable than staying in its lane, the flow may be partially absorbed as follows:

$$flow_{k,i} = \begin{cases} flow_{k,i} & , withinSameLane(i,k) \\ flow_{k,i} \cdot (1 - k_{absorp}) & , \neg withinSameLane(i,k) \end{cases}$$

where k_{absorp} is a parameter defining which quota of a lane crossing flow is cut by this method.

The cutting method was evaluated in the TORCS scenario, introduced in Sec. 5.3. The absorption was set to $k_{absorp} = 0.9$.

Due to the flow cutting method, the MDBHF with the lane knowledge reacts slower to lane changes but achieves better tracking during the straight driving phases before and after the lane change. This is visible in Fig. 7.3, where the variances are smaller and the expectation value nearer to the ground-truth position in the straight driving phases. This result is also backed-up by the two further evaluation measures: the distance measure $dist(E(P(X_t)), x_{real})$ (Table 7.1) and at the assigned ground truth proba-

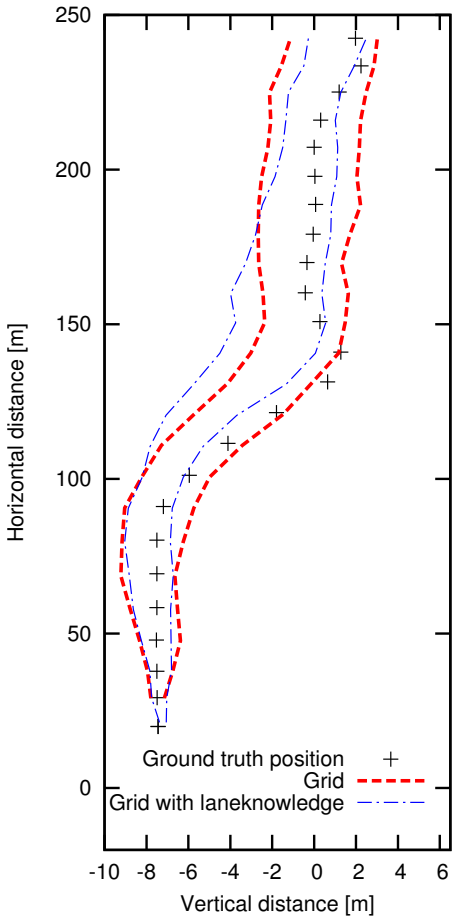


Figure 7.3: Mean and standard deviations of the expectation values of the posterior distributions of the filters are compared with the actual overtaking vehicle position, plotting corridors of one standard deviation from the mean for each filter approach. Compared are the MDBHF without lane knowledge and the MDBHF with lane knowledge.

bility $P(X_t = x_{real_t}|Y_{1:t})$ (Table 7.2). The evaluation measures are assessed and categorized in Sec. 8.4.2 as specified in Eq. 8.17.

In the graphical comparison over time of the assigned probability at ground truth metric, the lane change at time step 10 is clearly visible (Fig. 7.4).

The drawback of the cutting method is that it only distinguishes between wanted and unwanted flow instead of being able to change the direction and the absolute value of the velocity. Additionally, the cutting of the flow is theoretically questionable. Each flow represents the trajectory of an estimated vehicle, and when cutting the flow the trajectory becomes at once unlikely. The existing trajectory hypothesis is suddenly removed (at least

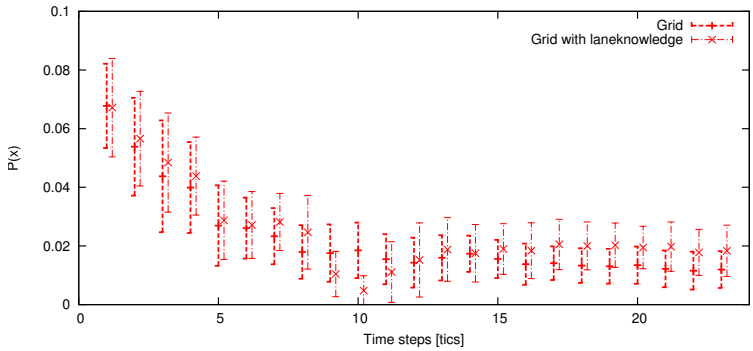


Figure 7.4: The probability $P(\hat{\mathbf{X}} = \mathbf{x}_{real_t}|\mathbf{y}_{1:t})$, and standard deviation, for the likelihood of the true state $\mathbf{x}_{real}$ during the overtaking situation with radar sensor information.

partly) from the estimate. However, just as vehicles cannot disappear at lane markings, neither should the vehicle hypothesis. It is much more reasonable to eliminate the unwanted flow at its root. With an unlimited amount of computational power, the best method would be to trace the flow back through time and eliminate the trajectory (as a branch of a hypothesis tree) which leads later to a lane marking crossing. Unfortunately, this method would be too complex, the first order Markov assumption would be violated and the whole hypothesis tree must be saved. As a consequence, the attractor method was developed, which basically models the driver behavior in a forward directed way in order to enforce stay-in-lane behavior.

Table 7.1: Average Euclidean distances *dist* between real position x_{real} and estimated position x and average standard deviations σ of the expected positions during the different stages of the overtaking maneuver

Stage	**Driving by**		**Changing Lane**		**In front**	
Radar Data	dist	σ	dist	σ	dist	σ
MDBHF (without lane)	0.49	1.51	1.81	2.08	0.62	2.24
MDBHF (with lane)	0.61	1.15	3.19	2.00	0.90	1.62
Camera Data	dist	σ	dist	σ	dist	σ
MDBHF (without lane)	0.18	0.86	0.72	1.36	0.33	1.59
MDBHF (with lane)	0.16	0.89	1.19	1.23	0.32	1.39

Table 7.2: The average probability $P(\hat{\mathbf{X}} = \mathbf{x}_{real_t}|\mathbf{y}_{1:t})$ at the real position x_{real} in the different stages of the overtaking maneuver.

Stage	**Driving by**	**Changing Lane**	**In front**
Radar Data			
Grid (without lane)	0.0374	0.0168	0.0140
Grid (with lane)	0.0406	0.0121	0.0191
Camera Data			
Grid (without lane)	0.0744	0.0246	0.0151
Grid (with lane)	0.0775	0.0225	0.0159

Change the Probability Flow or each State Transition.

Instead of reducing the probability flow, each $flow_{k,i}$ or state transition could be redirected. This is rather difficult in BHFs for two reasons: First, the new target of the flow probably does not lie directly on a cell representative. Sophisticated averaging strategies are needed to distribute the flow to the neighboring cells of the new target position. Another way to solve this could be to change the position of the target cell k. However, a simple position change is not enough. In fact, a new representative needs to be generated for each new target k. For particle filters it may be possible to redirect single particles, but for BHFs redirecting each flow would result in a very costly implementation. Second, the computational complexity would be as high or even higher than the calculation for generating the flow in the prediction model. Calculating an own target for each flow has the complexity $O(N^2)$.

Change each State or Change the Expectation Value in each Grid Cell

An elegant solution is to calculate a new target for the expectation value of each cell, instead of calculating a target for each flow. It is relatively fast to compute with $O(N)$ and it uses the distributed representation in the form of N grid cells. Each grid cell gets the yaw rate and acceleration assigned that a vehicle at the grid cell position would need in order to reach the target direction and the target velocity. The incorporation of context information improves the knowledge about vehicle movements. The improved prediction model reduces the information loss in the prediction step and this can again be realized by smaller variances in the prediction model σ_γ and σ_β.

Note that smaller variances in the prediction model, and therefore less information loss, would also emerge with the "Change each state transition" method due to converging flows. However, this can be sufficiently emulated by setting the prediction model variances to smaller values a priori instead of attracting the flow that is created by the prediction model.

Due to the mentioned properties we have chosen this method for the context incorporation in the ICUBHF (Sec. 7.2.2).

Set an Overall Goal for the whole Estimate or change a single State Representative

This method defines a single target for all states. Usually the target yaw rate and the target acceleration are only sampled at a single representative. In Kalman filter approaches the expectation value would be used as a representative. In the Multiple-Hypothesis Kalman filter each expectation value would state a representative. For the Kalman filter approach, the method of only probing the context influence by distance relations to a (or at most to a few) state representative(s) is the only obvious way to incorporate context information.

The method using the context information at only a single representative has the drawback of ignoring the fact that vehicles at different initial states may have different goals. In a simple example, a Kalman filter (or a MDBHF using this method) is used for vehicle tracking. Let the tracked estimate be in front of a traffic light. The traffic light changes to red and the representative used for context incorporation lies in front of the traffic light. The whole estimate PDF is decelerated in order to model the behavior of a vehicle which is stopping in front of the traffic light. This behavior is correctly modeled, but it ignores all probabilities that already lie behind the traffic light in the position estimate PDF. To be more precise, that vehicle position hypotheses are also decelerated despite most real drivers would not brake within the crossing behind a red light.

The target position that a vehicle at the representative position may have in order to reach a certain goal is applied to all possible vehicle positions.

7.2.2 The ICUBHF Approach

We have pointed out the advantages of distributed representations when it comes to context incorporation. We also made it clear that unwanted probability flow representing unlikely trajectories has to be eliminated at its root. In the following subsections we present a generic algorithm in order to incorporate context information into Bayesian filtering. It can be used to attack the problem of context incorporation either by changing each state transition (especially when using particle filtering) or by changing the yaw rate and acceleration of each state (MDBHF). Parts of this subsection are published in [4]. We apply this approach to the MDBHF and receive a Iterative Context Using Bayesian Histogram Filter (Iterative Context Using Bayesian Histogram Filter (ICUBHF)).

The context is fused into the ICUBHF in the following way. For each grid cell:

1. Attractor candidates need to be identified first.
2. An attractor has to be chosen from the list of candidates
3. A trajectory from the initial state to the attractor state has to be generated.
4. The yaw rate and acceleration is adapted to match the trajectory.

When more than one behavior alternative will be modeled at once, a multiple model approach with multiple ICUBHFs can be utilized (cf. Part III). The following sub-subsections deal with single goal behavior first. Parts of this subsection were originally published in [6].

Finding Attractor Candidates

An *attractor* is a defined goal state $x_A = (\omega(\mathbf{v}_i^A), ||\mathbf{v}_i^A||, \mathbf{l}_i^A)^T$ for each initial state x_i. Each goal state belongs to an assumed behavior $b_k \in B$ given by the context $c \in C$. We will explain this with an example. A vehicle driving straight towards a red light will have one of two different behaviors: stopping in front of the traffic light b_1 or violating the traffic light b_2. Since behavior b_2 is not very likely we will focus on the stopping behavior. The goal state position $\mathbf{l}_i^A$ is best set somewhere in the lane before the stop bar. There is more than one attractor candidate possible. The absolute velocity value at the goal state should be modeled to be zero $||\mathbf{v}_i^A|| = 0$. The goal direction $\omega(\mathbf{v}_i^A)$ is not regulated by the traffic light, but it can be determined by the lane direction at the goal position in front of the stop bar. The concept applies analogously to speed limits or other signs.

When looking at the attractors that only determine the absolute velocity value, it becomes clear that lane-constrained states suffice for that kind of attractors. That means that they do not require a two dimensional spatial distributed representation of the estimate as the MDBHF or the derived ICUBHF provide, whereas attractors defining the direction highly demand such a representation. In the remaining part of this thesis we therefore focus on the influence of certain behaviors on the direction by using attractors. The modeling of velocities is possible by using, next to traffic lights or traffic signs, also the incorporation of the following context into the prediction model: braking lights, indicator lights, other vehicle positions, vehicle-2-vehicle communication and more. For further readings, see e.g. [50] and [60].

By defining an attractor function AF (Eq. 7.2) we can generically model driving behavior. It defines for each initial state, meaning for each grid cell i an attractor location $\mathbf{l}_i^A$ as well as a heading direction $\omega(\mathbf{v})_i^A$ and velocity $||\mathbf{v}_i^A||$ given the state estimate represented in a particular grid cell i and the current context c of the road and surrounding traffic.

$$\begin{pmatrix} \omega(\mathbf{v}_i^A) \\ ||\mathbf{v}_i^A|| \\ \mathbf{l}_i^A \end{pmatrix} = AF(||\mathbf{v}_i||, \omega(\mathbf{v}_i), \mathbf{l}_i, c) \tag{7.2}$$

The general attractor function may be realized by individual algorithms dependent on the attractor type. In the following, we consider an AF algorithm for lane following, meaning a vehicle which is already in a lane should stay in its lane according to its prediction model. The stay-in-lane assumption fits to most situations. We can detect a violation by methods proposed in Part III. The context, and therefore the lane information, has to be provided by a virtual sensor. The virtual sensor may be a lane detection algorithm working on camera images or a map containing a lane-graph. The real-world lane-data provided by HRI-EU is in polyline format. We have chosen to align the proposed attractor algorithm to lane data provided in the polyline format because it is easy and fast to process and other formats can be easily approximated by polylines. However it is also possible to rewrite the AF in other formats.

The proposed AF algorithm is detailed in Alg. 7.1 and illustrated in Fig. 7.5. The algorithm generates attractor points within the lane of the initial position. This means that the vehicle would stay in its current lane. Different parts of the estimate PDF may follow different lanes. The context c is here a tuple K of lanes. Each lane $k \in K$ consists of a left lane border polyline L_k, a right lane border polyline R_k, and the lane center polyline M_k.

Algorithm 7.1: Attractor function algorithm (AF) considering lane context.

input : $(||\mathbf{v}_i||, \omega(\mathbf{v}_i), \mathbf{l}_i, c)$
output : The attrator state: $(\omega(\mathbf{v}_i^A), ||\mathbf{v}_i^A||, \mathbf{l}_i^A)^T$

forall lanes k **do**
  if $\mathbf{l}_i$ within lane $k \in K$ **then**
    $A_k \leftarrow \{\mathbf{m}_{kj} | \mathbf{m}_{kj} \in M_k \wedge \overline{\mathbf{m}_{kj}\mathbf{l}_i} \cap L_k = \emptyset \wedge \overline{\mathbf{m}_{kj}\mathbf{l}_i} \cap R_k = \emptyset\}$
    $A_k \leftarrow \{\mathbf{a}_j | \mathbf{a}_j \in A_k \wedge \mathbf{a}_j$ fulfills (7.3,7.4) and laneDirectionAt($\mathbf{a}_j$) fulfills (7.5)$\}$
    $\hat{\mathbf{a}} \leftarrow arg\max_{\mathbf{a}_j \in A_k}(||\overline{\mathbf{a}_j\mathbf{l}_i}||)$
    return (laneDirectionAt($\hat{\mathbf{a}}$), $||\mathbf{v}_i||$, $\hat{\mathbf{a}}$)
  end
end

The algorithm considers the lane k in which the grid point i is located. It restricts the set of attractor candidates (meaning potential attractor positions) more and more and finally selects the actual attractor $\hat{\mathbf{a}}$. Specifically, it generates the set A_k of all points $\mathbf{m}_{kj}$ that lie on the lane center M_k and whose connection to the grid point location $\mathbf{l}_i$ does not intersect with the

lane borders. This models that the observed driver needs visual contact to the attractor (cf. the concept of gaze-points in [50].) Points $\mathbf{m}_{kj} \in M_k$ are either the polyline vertex or additional equidistantly distributed points on M_k. Next, the algorithm verifies if the points in the considered set are sufficiently close in distance and angle to the starting state at grid cell i. Finally, the algorithm returns the attractor values derived from the attractor point. Note that the absolute velocity value is not changed by the algorithm. Fig. 7.5 illustrates the algorithm in a street-curve scenario exemplarily for one grid cell with indicated estimates for velocity and heading direction.

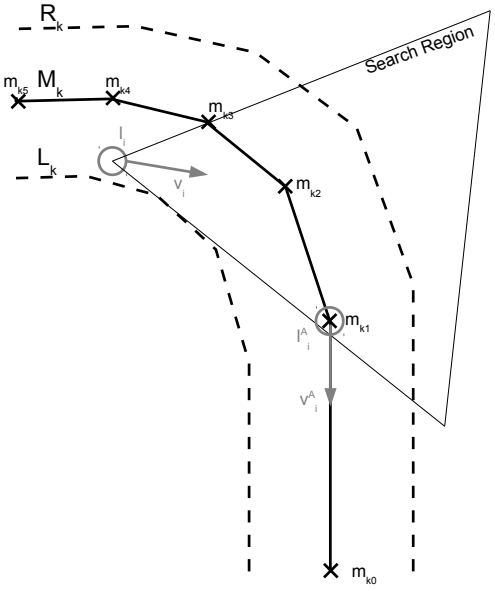


Figure 7.5: The proposed attractor algorithm considering lanes. m_{k1} is selected as attractor for this specific grid cell i. Fig. 7.6 shows the positions m_k in a real-world intersection.

The attractor state reachability assertion in line four of the algorithm considers whether the attractor state may be reached from the current state in a reasonable way. The reachability is approximated by defining a triangular search region (cf. Fig. 7.5). Such reachability assertions may consider additional factors, such as information about the road level, e.g. A-road, B-road, urban, in order to set a looking-forward distance d_{max} and maximum change in the yaw rate β_{max} based on the road conditions. In effect, attractor points are only considered within the area around the grid node i (c.f. search region in Fig. 7.5): The attractor points need to be close enough at grid point i as specified in (7.3) and the direction change should not exceed β_{max}as specified in (7.4). Additionally the aimed-at heading direction $\omega(\mathbf{v}_i^A)$ does not differ too strongly from the initial heading direction $\omega(\mathbf{v}_i)$

in grid point i (7.5):

$$(||\frac{\overline{m_{kj}l_i}}{\Delta T}||) \leq d_{max}, \tag{7.3}$$

$$|atan2(\omega(\mathbf{v_i}), \overline{m_{kj}l_i})| \leq \beta_{max} \cdot \Delta T \tag{7.4}$$

$$|\omega(\mathbf{v}_i) - \omega(\mathbf{v}_i^A)| \leq \beta_{max} \cdot \Delta T, \tag{7.5}$$

where β_{max} specifies the maximum change of the yaw rate and d_{max} specifies the maximum distance change allowed in a time window ΔT.

Figure 7.6: Attractor candidates in a real-world test scenario. Only positions are visualized as marker, no directions are visualized.

Now the attractor state and the initial state are determined. In order to incorporate the newly gained knowledge of the attractor state into the ICUBHF a trajectory (cf. Fig. 7.7) has to be defined between both points. The trajectory itself defines how the yaw and acceleration has to be set in order to follow the trajectory towards the attractor point.

Trajectory Generation

In order to model the trajectory between the initial state $\mathbf{l}_i$ and the attractor state $\mathbf{l}_i^A$ we connect both points by a cubic spline (Eq. 7.6) considering initial direction $\omega(\mathbf{v}_i)$ and attractor direction $\omega(\mathbf{v}_i^A)$. Note that for $\mathbf{v}_i^A$ and $\mathbf{v}_i$ the direction is given by the attractor algorithm, but the length of these vectors is a free parameter. Note that the free parameter cannot be interpreted as velocity. The free parameter is used in order to optimize the trajectory. d is the continuous parameter within the interval $[0, 1]$ where the resulting set of values describes all points on the spline.

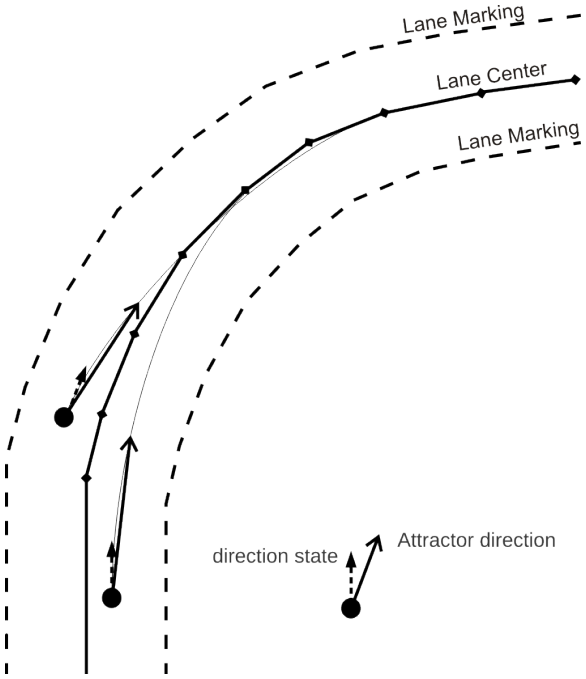


Figure 7.7: A number of initial position with different attractor points.

$$s(\mathbf{l}_i, \mathbf{v}, \mathbf{l}_i^A, \mathbf{v}_i^A, d) = (2\mathbf{l}_i - 2\mathbf{l}_i^A + \mathbf{v}_i + \mathbf{v}_i^A)d^3 + (-3\mathbf{l}_i + 3\mathbf{l}_i^A - 2\mathbf{v}_i - \mathbf{v}_i^A)d^2 + \mathbf{v}_i d + \mathbf{l}_i \quad (7.6)$$

Techniques other than splines are possible in order to generate trajectories between the goal point and the start point, but splines are easy to implement and fast to compute. This is an important feature since the trajectories have to be calculated for each grid cell i. A spline technique has also been proposed by [38] to model trajectories, but more sophisticated trajectories are also possible. A lot of splines do not correctly model vehicle trajectories. In order to reliably generate trajectories with the typical properties of vehicle trajectories, iterative algorithms are necessary, as presented by [41]. Such iterative processes are computationally too expensive for our ICUBHF approach. A comprehensive work on curves that gives a good introduction into iterative smoothing and deformation of curves is [52].

What are the desired properties of a vehicle trajectory? It should be sufficiently smooth. The curvature should be minimal since the driver will usually use minimal steer angles to achieve his goal. The derivation of the curvature should also be small, since usually changes of the steering angles are performed gently by human controllers.

According to these desired properties, the spline should have been optimized in such a way that both requirements are achieved. At first glance

there are two possibilities: minimize the overall (average) curvature or minimize the maximum curvature. Unfortunately, both attempts lead to either integrals that are not solvable in an analytical way or to a numerical root-finding, so that iterative numerical solutions would again be the result. In order to achieve the above requirements in an approximate way, we search the length of $||\mathbf{v}_i^A|| = ||\mathbf{v}_i||$ minimizing the acceleration in x_0 and x_1 directions in our spline (cf. Eq. 7.7). The result of the optimization is stated in Eq. 7.8. Note that we do not use the lane borders as a constraining factor in this optimization, meaning that resulting trajectories may leave the lane in extreme situations. This is an unwanted effect but cannot be easily solved without iterative solutions. The risk of such trajectories is minimized by setting restrictive values in the reachability constraints (Eqs. 7.3, 7.5 and 7.4).

$$||\mathbf{v}_i^A||^* = ||\mathbf{v}_i||^* = argmin_{||\mathbf{v}_i^A||=||\mathbf{v}_i||}(s_0(\mathbf{l}_i, \mathbf{v}, \mathbf{v}_i^A, \mathbf{v}_i^A)''^2 + s_1(\mathbf{l}_i, \mathbf{v}, \mathbf{v}_i^A, \mathbf{v}_i^A)''^2) \quad (7.7)$$

$$||\mathbf{v}_i^A||^* = ||\mathbf{v}_i||^* = \max\left(\frac{3(\mathbf{l}_{i_0}^A \cos(\omega(\mathbf{v})) + cos(\omega(\mathbf{v}_i^A)) + \mathbf{l}_{i_1}^A(\sin(\omega(\mathbf{v})) + \sin(\mathbf{v}_i^A)))}{2(2 + \cos(\omega(\mathbf{v}_i^A) - \omega(\mathbf{v})))}, 0\right) \quad (7.8)$$

The spline (Eq. 7.6) is then used with the re-sized velocity vectors $\mathbf{v}^*$ and $\mathbf{v}_i^{A*}$ in order to receive the optimized spline $s(\mathbf{l}_i, \mathbf{v}^*, \mathbf{l}_i^A, \mathbf{v}_i^{A*}, d)$. The maximum operator in Eq. 7.8 was introduced in order to avoid negative values that can emerge from the optimization when a backward motion would be smoother than a forward trajectory.

The trajectory is thereby generated. We can now determine the new yaw that needs to be set in grid cell i in order to follow the trajectory.

For this the distance d^* a vehicle at grid cell i will drive in a time step ΔT is estimated by Eq. 7.9. The heading direction (yaw) to reach this point $\omega^*(\mathbf{v}_i)$ is calculated by postulating the vector from the start position to the position d^* on the spline. The yaw $\omega(\mathbf{v_i})$ is therefore changed by the attractor algorithm to $\omega^*(\mathbf{v}_i)$ (Eq. 7.10).

$$d^* = \min(||\mathbf{v}_i|| \cdot \Delta T/(\mathbf{l}_i^A - \mathbf{l}_i), 1) \quad (7.9)$$

$$\omega^*(\mathbf{v}_i) = atan2((s(\mathbf{l}_i, \mathbf{v}^*, \mathbf{l}_i^A, \mathbf{v}_i^{A*}, d^*) - \mathbf{l}_i), \mathbf{x}_1) \quad (7.10)$$

The direction change is derived by probing the spline at a distance d around the start node (7.9). Here, d is an approximation of the ratio between the drivable path length during a time step ΔT and the length of the spline

generated. We approximate it by using the velocity $||\mathbf{v}_i||$ and the distance between $\mathbf{l}_i$ and $\mathbf{l}_i^A$. This approximation is illustrated in Fig. 7.8 and fits well for splines with small curvature.

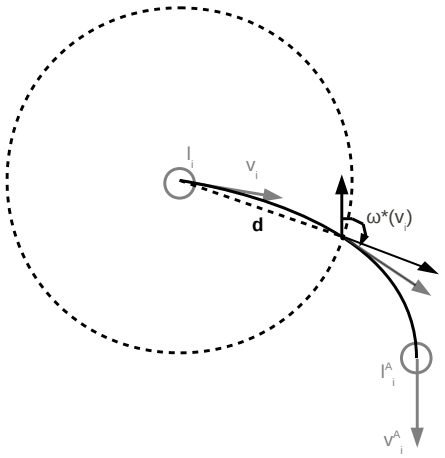


Figure 7.8: The spline is probed in order to receive the new $\omega^*(\mathbf{v}_i)$.

Using Context Information via Attractor Functions

In this sub-subsection we will summarize the steps necessary to infer context information via attractors. We will also critically discuss the attractor approach and show the performance with an evaluation. Fig. 7.9 gives an graphical overview on the ICUBHF, resembling all steps for the context information via attractors.

The underlying assumption for attractor usage was that drivers react to context. Therefore, the prediction model can be better modeled by using context instead of using only kinematic models. In order to change the prediction model several steps were taken:

1. The attractor function delivers goal points towards which a driver will drive (Eq. 7.2 Alg. 7.1)

2. Trajectories between the current estimated positions and the goal points are generated (Eq. 7.8, 7.9 and 7.10).

3. The prediction model in each grid cell is changed, according to the trajectory, by adjusting yaw and velocity.

The first item is clear and understandable when looking at the human driver. Deviations may occur, but in many situations the human will define the goal first, and then control the car in a way that will achieve the goal

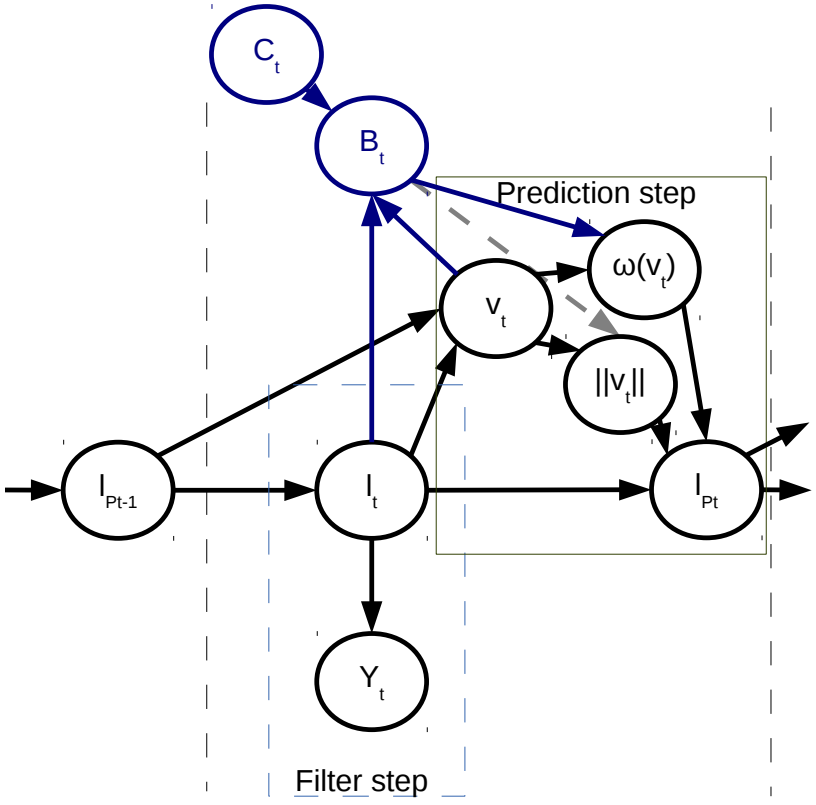
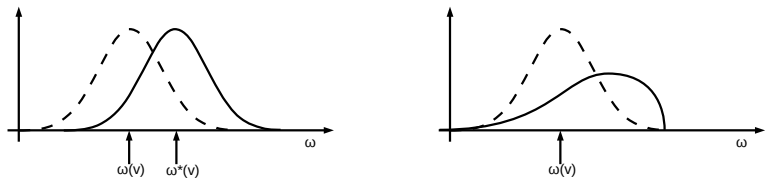


Figure 7.9: A simplified graphical view on the ICUBHF. B_t is the behavior model variable influencing the yaw rate and velocity change (determined by the attractor function). The behavior depends on the context C_t and the vehicle state. A more elaborated insight on the behavior model is given in Sec. 9.1.1

position. The second item is more arguable. While it is clear that a trajectory has to be generated between the goal and the initial point, the way the trajectory is stated is a compromise between fast computability and accuracy. For example, we do not test whether the trajectory leaves the lane, even when the gaze line to the attractor itself is not allowed to intersect the lane border. The way the prediction model is changed by determining the yaw rate and the velocity change by the trajectory to the attractor is again rather elegant. However, by resetting the expected direction and the expected velocity, the model ignores the kinematic constraints when the change in direction or velocity is too high. This is indirectly compensated by the reachability assertions Eqs. 7.3, 7.4 and 7.5. The drawback of a more precise solution would be that the prediction model is no longer Gaussian in its different locations. The benefit of the usage of Gaussians is that they are the weakest assumption and therefore obligatory, since the specific kinematic parameters such as β_{max} of the different vehicles are not known. A graphical comparison is illustrated in Fig. 7.10.



(a) The expectation value is set to the new $\omega^*(\mathbf{v})$. The maximum steering angle assumption may be violated by a small amount of probability.

(b) A maximum steering angle is determined and it is more probable to steer towards the trajectory. No probability exceeds the maximum steering angle.

Figure 7.10: The figures show the direction dimension of the prediction function PDF and how the direction PDF is changed by the attractor.

The drawbacks to the use of Gaussians are further reduced due to the decreased variance of the Gaussian distribution after the attractor incorporation. The Gaussian distribution becomes smaller and steeper, and does in a best case scenario not exceed the dimensions of the original distributions as illustrated in Fig. 7.11. We treated relative reduction in direction variance as a free parameter. A future work may include the task to quantify the information gain by comparing the model to real driver behavior.

There are other ways to possibly improve the attractor incorporation in the future. When looking back to Fig. 7.8 it becomes clear that there are at least two ways to set the new yaw in the new grid cell. In both cases the prediction function is set to the direction which is the difference between the reachable point on the trajectory and the initial position: $\omega^*(\mathbf{v})$. However,

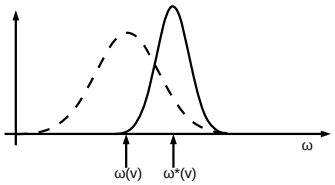


Figure 7.11: The variance in the direction dimension is reduced by the increased information on the steering.

the velocity of the predicted estimate could either be set to $\omega^*(\mathbf{v})$ as we did in our evaluations, or set to the direction that a vehicle at the reachable point on the trajectory would have (indicated by the gray arrow in Fig. 7.8). Such changes may further improve the results achieved by the attractor function.

Comparative Evaluation of the MDBHF and the ICUBHF

In the following we want to show the benefits of the ICUBHF in the tracking task. The main difference, of course, occur in curved roads. The average run should match to the ground-truth position (average error = 0), but also in straight roads the estimate will follow the lane, and therefore the deviations, between the ground-truth and the estimate will be smaller for the individual tracking run in the ICUBHF (the variance in the error is smaller). The average error is already near zero in the MDBHF in straight driving phases.

We used different curved road scenarios in order to evaluate the benefit of the ICUBHF with the attractor model over the MDBHF in curved roads. In the first scenario the ego-vehicle E follows an observed vehicle O into a right curve. In the second scenario an oncoming vehicle is tracked in a left curve (left from the ego-perspective). Some parts of this evaluation have been published originally in [6].

Scenario: Follow into Curved Road The ego-vehicle is following another vehicle in a curved road segment. Both vehicles are driving with a velocity of 90 km/h. The results are shown in Fig. 7.12 comparing the results from a Bayesian filter without context information with the Bayesian filter with context information. The graph shows that the average position error and the variance in the position error can be reduced by using relevant context in the ICUBHF. The remaining error is mainly caused by the fact that the attractor model is not perfectly matched to the discrete steering behavior of the TORCS controller of the ego-vehicle. In addition, the ego-movement compensation is not enabled in this experiment, but the curve radius is wide enough to avoid large errors. Fig. 7.13 confirms the advantage

of the lane following attractor, plotting the likelihood at the ground truth $P(X = x_{real})$, which is greater when the attractor is applied. The results show that the attractor-based adaptation of the motion model can handle radial motion well, even without measuring yaws as in [13] and [14].

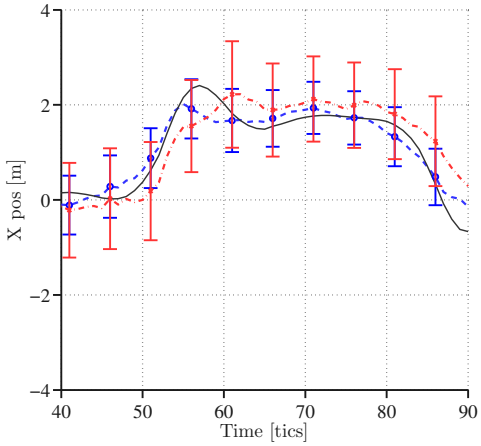


Figure 7.12: The x position of the observed vehicle relative to the ego-vehicle over time. The small solid line is the ground truth position, the blue dashed line is the average tracking result of the ICUBHF, and the dotted red line is the MDBHF.

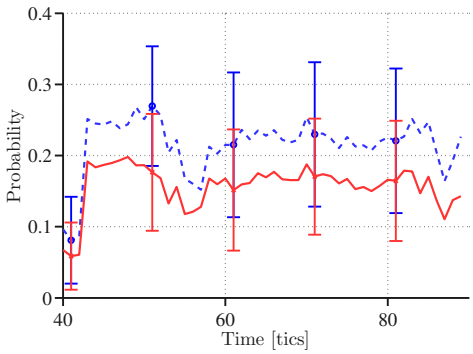


Figure 7.13: The probability that the grid filter assigns to the ground truth position (cf. Sec. 8.4.2). The dashed line is produced by ICUBHF and the solid line is given by the MDBHF.

Scenario: Passing Opposing Vehicle This scenario is a setting comparable to that used in [13] and [14]. In that experiments a Kalman filter is used to track oncoming vehicles in curve scenes. In our scenario the opposing vehicle O drives at 50 km/h, while our vehicle is static. This setting is more challenging because there are not many time steps in which the other vehicle is observable, and the relative velocities are higher. As a result, there is not

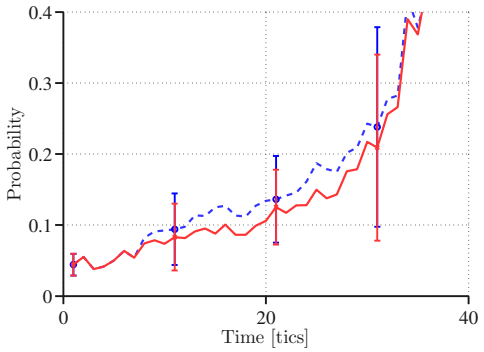


Figure 7.14: The probability that the grid filter assigns to the ground truth position (cf. Sec. 8.4.2). The dashed line is produced by the ICUBHF, the solid line is given by the MDBHF as a baseline.

much time for the Bayesian filter to calibrate its internal velocity estimates, since the filter is initialized with zero velocities in all grid cells.

The evaluation shows that the attractor benefits are smaller than in the previous scenario setting. This is for several reasons. First, the lane information was not available during the first seven time steps, since the lane detector in TORCS is not able to see behind curves. Second, with the observed vehicle approaching our ego-vehicle, the radar sensor noise becomes smaller (cf. Sec. 5.1), so that the attractor model influence decreases since the influence of the measurement model increases

The results in Fig. 7.14 show that the additional attractor concept again improves the likelihood of the predictions. As explained, the attractor does not apply as the lane-detector does not deliver context information during the first time steps and therefore no difference compared to the results without the attractor can be seen in the graph. After about seven time steps, however, a difference becomes visible and stays visible until about the 33rd time step. At this point, the vehicle is rather close to the ego-vehicle, which minimizes the sensory noise of the simulated radar sensor. The distance to the ego-vehicle over time and the respective estimates of the MDBHF and the ICUBHF are illustrated in Fig. 7.15 for comparison.

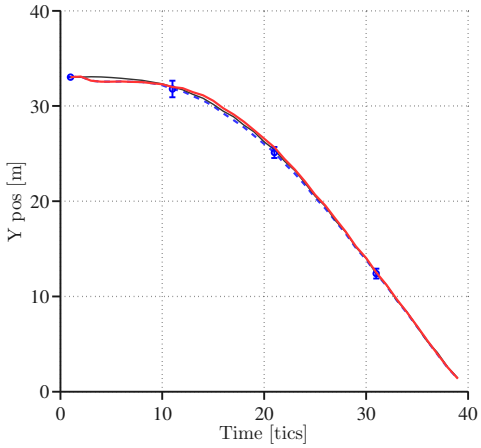


Figure 7.15: The y position of the observed vehicle relative to the ego-vehicle over time. The small solid line is the ground truth position, the dashed line is the average tracking result of the ICUBHF and the red solid line is given by the MDBHF as a baseline.

7.3 Error Evaluation of the ICUBHF Approach using Ego-Compensation

This evaluation section continues the experiments in 6.3 and shows the performance obtained by using the attractor based context and the ego-movement compensation together on the scenarios introduced in that section.

We will show how the average tracking is improved and then how the ICUBHF copes with systematic sensor errors and temporarily detection loss.

7.3.1 Optimal Tracking In a Curve Scenario

The same scenario is used as in Sec. 6.3. The tracking during the curve is clearly improved as depicted in the evaluation graph (cf. Fig. 7.16). E.g. the overshoot when the observed vehicle is leaving the curve is significantly smaller. During straight driving the kinematic model used in the MDBHF also performs well in the average. However, it becomes clear that the smaller variance in the prediction model due to the incorporation of the context information leads to lower variances between the different simulation runs, not only during the curve, but also during the entire tracking.

7.3.2 Systematic Sensor Errors and Sensor Limitations

The prediction model was substantially improved by introducing the attractor approach. While systematic offset errors in the sensor input will automatically lead to offset errors in the estimate, temporary detection loss

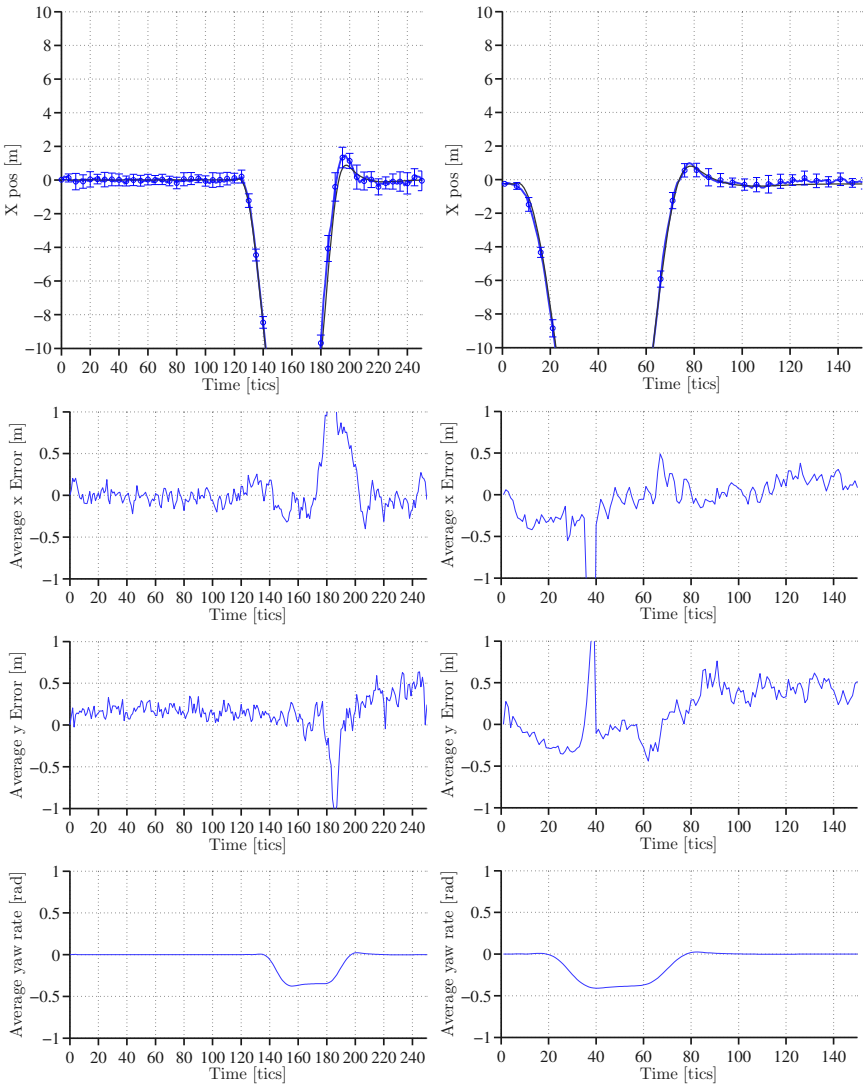


Figure 7.16: Tracking of the x-position during a left curve with radius = $30m$. In the first row the blue line is the estimated x-position and the black line is the ground-truth x-position. The next two rows show the average error in x and y direction. The last line is the yaw rate. The left figures show the tracking with enabled ego-compensation in the MDBHF and the right figures the tracking in the ICUBHF (obervation time-span reduced).

should be handled well with a reasonable prediction model. This evaluation shows that the ICUBHF copes in an adequate way with both, systematic sensor errors and temporary detection loss.

The evaluation was carried out with a constant offset of $0.25m$ and a detection loss during 4 iteration steps (tics). The detection loss occurs since the observed vehicle is temporary leaving the sensor range to the side during the curve driving. This is because of the reduced curve radius for this experiment. We used the same setting as in the $30m$ scenario but reduced the radius to $25m$. During the detection loss the sensor detection returns $\mathbf{0}$. Such a detection loss event is caught by a routine that runs the ICUBHF in a pure prediction loop (cf. Sec. 4.3) during the duration of the event.

The result of the evaluation is shown in Fig. 7.17. First, it can be seen that the temporary detection loss does not lead to a tracking loss. The estimate quickly adapts to the new sensor information after the detection loss phase with the ICUBHF running in a pure prediction loop. In the same figures we can see that the sensor offset does not lead to problems during the straight driving phase. The estimates offset complies with the offset of the sensor input. A better tracking behavior is not possible since the filter does not know about the real state. During the curve we see that the estimate deviates with an offset almost twice as high as the sensor offset in the opposite direction. This is a reasonable effect of the attractor approach. The offset in the sensor input suggests to the ICUBHF that the vehicle is in the right half of the lane during the left curve. This is equivalent with an offset in the context information. The attractor approach then models trajectories leading into the center of the lane in order to compensate for the errors. We learn that the attractor approach can only be used in order to improve the tracking when the context information is accurate and no offset between context and vehicle position exists. In this situation the offset was rather small. But when the offset is increased in size the following principle applies: Tracking cannot be improved when putting additional wrong information into the filter. While the Bayesian filter is meant to deal with noisy inaccurate information, feeding in wrong information such as a sensors with an offset or in the same way context information with an offset the filter will perform worse than without that information. In the following part we will focus on behavior detection with the help of the ICUBHF.

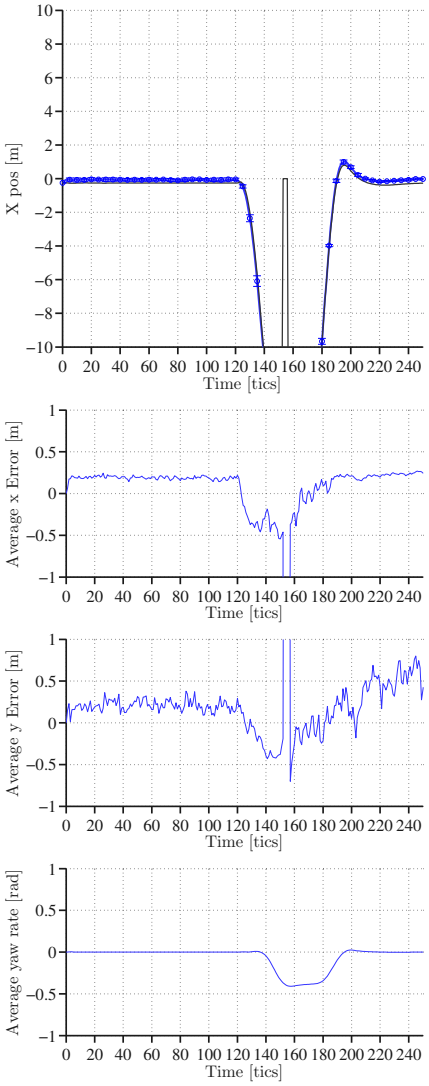


Figure 7.17: Tracking of the x-position during a left curve with radius = $25m$. In the first row the blue line is the estimated x-position and the black line is the ground-truth x-position (note that there is an exception: during the detection loss at time step 150, the simulator returns 0 for the position). The next two rows show the average error in x and y direction. The last line is the yaw rate. The sensor detection has an offset of $0.25m$. After the detection loss, which occurs roughly in time step 150, the tracked object was not lost by the filter.

Part III

Driving Behavior Tracking

ADAS systems need not only the knowledge about the current position of the vehicles, but also knowledge about the position the other vehicles will occupy in a few seconds in order to avoid risky situations or warn an inattentive driver in the right moment. The future position depends on the current position and the behavior of the driver. This part of the thesis deals with the problem of how to anticipate the driving behavior of others.

Chapter 8

Behavior Tracking Theory

In order to track behavior, a stochastic variable B needs to be introduced in our Bayesian network. When using Bayesian filtering the variable also needs to be part of the state vector. In this chapter we will first discuss behavior estimation beyond the Bayesian filtering framework. After understanding the general concept of behavior estimation, we introduce behavior estimation within the Bayesian filtering theory.

In both cases the behavior is referred to as variable B. Later we will be more accurate and differentiate between the behavior model and the behavior mode (Sec. 8.1). When it comes to behavior tracking it is very important to distinguish between both terms, since in practical applications it is indeed the goal to estimate the behavior mode, but the estimators initially provide a behavior model. However, to avoid confusion we will start the introduction without the concept of behavior modes and make the behavior concept more concrete later. The behavior variable B stands for the behavior model here. In the ICUBHF the behavior model is generated by the attractor approach, which was introduced in Chapter 7 for improving the prediction model.

The overall idea of behavior tracking is that the right behavior model b^* has to be estimated given all (sensor) inputs $\mathbf{y}_{1:t}$ [43].

$$b^* = \arg\max_{b \in B} P(b_{1:t}|\mathbf{y}_{1:t}) \tag{8.1}$$

This means that given all available knowledge about the other vehicle collected from time step 1 to t, we want to know the most probable behavior b^* from the set of all behaviors $b \in B$.

But even when the evidence $\mathbf{y}_{1:t}$ suggests that a certain behavior b^* is correct, it is not guaranteed that this behavior is actually the ground truth behavior B_{true}. It is possible that the noise in the sensor measurement causes a behavior other than the ground truth behavior to be estimated as the correct one. Hence, it is advantageous to know not only the best

matching behavior, but the whole distribution $P(B_{1:t}|y_{1:t})$. It gives the probability for each $b \in B$ that it is correct given $\mathbf{y}$, making it easier to tell if the situation is rather clear or ambiguous.

8.1 The Behavior Mode and the Behavior Model Detection

In this section we will learn that it is important to distinguish between the behavior variable B modeled by the attractor, the estimated behavior mode $\hat{B}$ and the unknown real behavior mode $\hat{B}_{real}$. B is what the attractor approach models. When we estimate whether a certain behavior model B is correct, we test whether the estimated state trajectory matches a behavior modeled by the attractors. First, however, we will have a look at an explanatory example.

8.1.1 Example without Bayesian Statistics

After reading this section it will become clear why a distinction between model and mode is necessary. Therefore, we look at a basic example.

Imagine we live in an idealistic world where all sensors are accurate. How can the behavior be detected in such an environment? It seems clear that we do not need a probabilistic model due to the deterministic world knowledge. This is not quite correct, though: Let a vehicle O in front of the ego-vehicle E approach an intersection. We detect the position of O accurately in each time step. We want to know if vehicle O stops in front of the intersection or if O immediately enters the intersection. Since E tracks the position accurately we also know O's velocity accurately. We can now derive a behavior model accordingly: a braking vehicle is going to stop in front of the intersection (b_1) and a vehicle with constant or increasing velocity will enter the intersection (b_2). For each vehicle decelerating we detect b_1. Of course the behavior detection will be correct in most cases due to the accurate velocity estimate, but some drivers may deviate from that reasonable behavior model. For example, there are certainly drivers who decrease velocity and enter the intersection anyway and others who speed up first only to brake suddenly. Due to that uncertainty in the driver action the behavior detection can only be solved probabilistically. Experienced human drivers are often prepared for common, or sometimes uncommon, behaviors of other drivers. By that we unknowingly utilize the concept of behavior modes. While braking or accelerating, given stopping or entering the intersection, respectively, is the behavior model, the concept of stopping in front of an intersection or entering an intersection is the behavior mode. Therefore, we can say that the assignment from model to mode is only possible in a probabilistic way. A more formal introduction and definition

follows in subsequent sections.

We will give another example, in which the behavior modes are categorized into turning or straight driving in the intersection. When the direction component of the velocity points to the side (behavior model b_1) we assume turning $\hat{b}_1$, and when the velocity of the vehicle points straight (behavior model b_2) we assume straight driving $\hat{b}_2$. Alternatively the position can be used for detection of the behavior. A position which is nearer to the sideward exit of the intersection implies turning, and a position nearer to the straight intersection exit implies straight driving. The reader may easily identify vehicle trajectories that lead to wrong behavior mode detections. For example, a vehicle that overtakes a bicycle in the beginning of the intersection may lead to a wrong turning mode detection.

Temporary deviations from the modeled behavior can be compensated by doing an integration over time using a HMM on the behavior mode. But this also has the drawback that due to the integration over time the detection may lag behind. Looking at these examples it has already become clear that there is a difference between behavior (mode) and the modeled behavior. In the following subsection we will introduce the definition for this from the literature, mode and model, before getting into recursive Bayesian estimation of behavior.

8.1.2 Behavior Modes versus Behavior Models

Li and Jilkov [49] introduce a nomenclature that differentiates between the real behavior and the behavior model. The mode space S is a discrete random variable, and state transitions are realized via Markov chains. The mode refers "to a pattern of behavior or a phenomenon, or a structure of a system" and a model "is a mathematical representation or description of the phenomenon pattern (system structure) at a certain accuracy level" [49]. In the examples in the previous section, we have already seen that the estimation we are doing is based on the mathematical model that only hints at which mode (real behavior) is correct. In most Bayesian filter approaches not all modes are covered by a model (meaning that the model space M is smaller than the mode space S). In our ICUBHF approach we cover the possible modes with a low number of complex models using a generic attractor model concept. The theory of using mode spaces also allows the system designer to alter the sensor noise characteristics depending on the mode. In our vehicle tracking task the modes only influence the prediction model, since there are no sensor modes. It is assumed that the observed vehicles are detected with the same accuracy independently from the behavior the observed vehicle is executing. Therefore, we denote the mode as the behavior $\hat{B}$ and the model as behavior model B. Sometimes we will use the original notation from the literature to make it clear that a mode can affect both, the sensor and the prediction model.

8.2 Multiple Behavior Models in Bayesian filters

In the example above (Sec. 8.1.1) we saw that it is hard to track the real behavior, even when there is no noise in the position measurement. The task becomes even harder when the position estimates are noisy. As shown in the intersection example from Sec. 8.1.1, when the position measurement is noisy, the derived velocity is even nosier. Therefore, the estimated vehicle direction can no longer be used for behavior detection when the noise becomes too high. Even integration over time becomes unrewarding. It therefore becomes necessary to use Bayesian filtering combined with a probabilistic comparison of different models.

This section details the strategy of how to deal with behavior models in Bayesian filtering and how to determine the right behavior model.

8.2.1 Overview over Multiple-Model Approaches

Li and Jilkov [49] distinguish between different types of approaches that deal with dynamic systems in which more than one mode $s \in S$ is possible. Simple approaches (so called non-multiple-model) pick one model in each time step from the set of possible models M_t and do the filtering and prediction with that model. The drawback of the non-multiple-model (non-MM) approach is that the model cannot be retroactively changed when proved wrong. It is also difficult to find out if the model was right since no other model was tested for comparison.

For this reason, more than one model is usually executed in parallel. These are the so called MM-approaches, which utilize, at least in some time steps, more than one filter in parallel [49].

In many MM approaches the results of the different models are combined again in order to improve the tracking. This can be done by a weighted fusion of the estimate PDFs or by selection of the estimate with the highest plausibility. In order to do this, a measurement is needed that quantifies the plausibility or quality of certain estimates. Section 8.4 is dedicated to those measures. There are two ways to combine the estimates. First, they can be merged in the output of independently running filters. Second, estimates can interact with respect to the transition probabilities between modes and the plausibilities of initial mode estimates.

These filters are called Cooperating Multiple-Model (CMM), or in older literature switching or dynamic MM algorithms. The estimates iteratively interact with each other (cf. Fig. 8.1). In each time step the hypotheses are merged in order to avoid a hypothesis tree emerging from the model transitions (cf. 2.2.2). From the CMM increasingly complex filter approaches have emerged. For example, the so called Variable-Structure Multiple-Model (VSMM) creates new models and eliminates inapplicable ones. Due to their high complexity, CMMs and VSMMs are, in most cases, not used with parti-

cle filter or BHF approaches, but only in simple point estimator approaches [49].

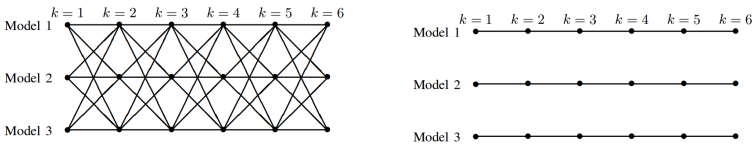


Figure 8.1: The estimates merge in the CMM approach and run in parallel in the AMM approach. The time steps are denoted with k in this figure. The figure was originally published in [49].

It is also for this reason that our ICUBHF is made up of multiple individual filters without interaction. Multiple model approaches of this kind are referred to as Autonomous Multiple Model (AMM) in [49]. They work optimally when the unknown mode or the unknown true behavior does not change over time. However, also when this assumption is incorrect the use of an autonomous filter has convincing benefits. First, we do not have to deal with the increasing complexity of the algorithm and the emerging hypothesis tree. The complexity of BHFs is already much higher than those of point estimators and Kalman filters. Second, we assume that the transition probabilities between the modes are small. A driver who has decided to drive straight, and whose behavior matches our straight moving model, will only in rare cases revoke his decision and turn. Hence, the errors implicated by a missing interaction between the filters are low and the modeling of an interaction could only improve accuracy with high effort.

8.3 Delimination between Behavior Detection in Non-Recursive and Recursive Estimators

The overall task of MM filters is to estimate the state $\mathbf{x} \in \mathbf{X}$ and the model $\mathbf{m} \in \mathbf{M}$ at the same time, given all inputs $\mathbf{y_{1:t}}$. That means in an accurate solution the hybrid state $(\mathbf{x}, \mathbf{m})$ has to be estimated. The Point estimators technique can be used to estimate the hybrid state by $P(X_{1:t}, M_{1:t}|Y_{1:t}) = P(X_{1:t}|M_{1:t}, Y_{1:t})P(M_{1:t}|Y_{1:t})$ [49].

When estimating the model by the sensor input $P(m_{1:t}|\mathbf{y}_{1:t})$, $\mathbf{y}_{0:t}$ should be considered noisy. Actually, we want to estimate the behavior given the true state $P(m|\mathbf{x_{real}}_{1:t})$, but the real state is not known. We are therefore forced to use the sensor input as the only available information on the real state. The noisier the sensor input, the higher the errors in the behavior estimation.

These general reflections also apply to recursive estimation. In order to do an online estimation, we use a Bayesian filter approach which iteratively

applies the Markov model on the sensor input to receive a state estimate. We cannot estimate the behavior and the state space at simultaneously. In other words the attractors specify the model, but the model-likelihood itself is not modeled by them.

The model is estimated in an indirect way by measuring the quality of certain PDFs in the Bayesian filter, and comparing the quality with that of another Bayesian filter running with a different model. Which PDFs need to be compared and how they can be compared is the topic of the next chapter. Note that we focus solely on **behavior** models from now on and therefore use the b notation instead of the general denomination m from the literature [49], which denominates **arbitrary** models. In other words this means that in contrast to b, m can be a sensor and a behavior model.

8.4 Model estimation and selection in Bayesian filtering

We will now estimate the model $P(b_{1:t}|y_{1:t})$ and detect a probably correct model $b*$ in Bayesian filtering (cf. Eq. 8.1). In the previous sections we pointed out the need for more than one model in order to deal with different behavior modes. We already mentioned that the models need to be selected or that a weighted combination of the estimates may be possible in order to improve tracking. For this we need to estimate which model fits better with the given sensor input $\mathbf{y_{1:t}}$. In the first subsection we show how to compare different parallel-running Bayesian filters in order to quantify which model is most likely to be correct. The comparison itself is possible by different kinds of measures. A number of possible measures are stated and categorized in the next subsections.

As already stated, $P(b_{1:t}|y_{1:t})$ is not known and needs to be derived. In recursive Bayesian filtering the Markov assumption is used, and therefore the task is to derive $P(b_t|x_{t-1}, y_t))$.

8.4.1 Derivation of the recursive Bayesian Behavior Estimation

Now we will step-by-step describe how to derive the quality measure for the behavior estimation.

First we apply the Markov assumption to convert the state estimation in a recursive state estimation.

$$P(\mathbf{X}_{t+1}|\mathbf{X}_t, B, \mathbf{Y}_{t+1}) \xleftarrow{Markov} P(\mathbf{X}_{1:t+1}|\mathbf{Y}_{1:t+1}) \tag{8.2}$$

Now we want to know, how likely it is that the estimate $\mathbf{X}_{t+1}$ emerged from the predicted estimate $\mathbf{X}_{P_t}$ given the sensor input $\mathbf{Y}_{t+1}$ and a certain

behavior $b \in B$. For this we use the Bayes theorem to restate the equation. Since we are using an AMM we want to quantify the likelihood of a certain behavior $P(b)$, instead of quantifying the whole probability distribution $P(B)$ at once.

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{Y}_{t+1}) = \frac{P(b|\mathbf{Y}_{t+1}, \mathbf{X}_t)}{P(\mathbf{X}_{t+1}|\mathbf{Y}_{t+1}, \mathbf{X}_t)} P(\mathbf{X}_{t+1}|\mathbf{X}_t, b, \mathbf{y}_{t+1}) \tag{8.3}$$

Equation 8.3 allows us to deduce the likelihood of a behavior model from the state estimate using the behavior model. $P(b)$ and $P(\mathbf{X}_{t+1})$ are approximated for the sake of computability as constant terms and we use them as a normalization term η in the next equation. The prior state estimate itself is a uniform distribution in our MDBHF and ICUBHF. $P(b)$ is set equal for each model.

In a further step the recursive state estimate can be rewritten: The estimate and the behavior $(\mathbf{X}_t, b)$ can be aggregated to the predicted behavior $\mathbf{X}_{P_t}$, since it is fully determined by the two variables:

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{Y}_{t+1}) = \eta P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}, \mathbf{y}_{t+1}) \tag{8.4}$$

The new estimate is the product of the predicted estimate and the inverse measurement model for a given input.

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) = \eta P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}) P(\mathbf{X}_{t+1}|\mathbf{y_{t+1}}) \tag{8.5}$$

That means we have to compare the state estimate using the predicted estimate $P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t})$ (and therefore depending the previous estimate $\mathbf{X}_t$ and the behavior mode b) for the estimation, with the estimate based on the sensor input $P(\mathbf{X}_{t+1}|\mathbf{y_{t+1}})$. The better the PDFs fit together, the higher the quality or plausibility of the filter and the better is the behavior model b used by the filter. The quality/plausibility measure is determined by the function Pl:

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) = \eta Pl(P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \tag{8.6}$$

Since no space transformation is necessary from the predicted estimate to the estimate or from the sensor space to the estimate space, the sensor and predicted estimate PDFs can be directly compared in our approach. We use another notation:

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) = \eta Pl(P(\mathbf{X}_{P_t}|\mathbf{y}_{1:t}), P(\mathbf{Y}_{t+1}|\mathbf{y}_{t+1})) \tag{8.7}$$

In other words, we can estimate the behavior by checking what measurement a filter with a certain predicted estimate would expect. Or, inversely, the sensor input has to be transformed by the inverse sensor model

into a state estimate. This state estimate has to be compared with the predicted state estimate. The better the compliance of $P(X_{t+1}|Y_{t+1})$ and $P(X_{t+1}|X_t, b_t)$, the more likely the position X follows the behavior model b.

In the following we look at different quality/plausibility measures which can be used to quantify how well the predicted estimate fits to the sensor input, and thereby how likely it is that a certain prediction model is correct.

We distinguish between different categories of quality measurements. First, we categorize between quality measures utilizing the whole PDF of the compared stochastic variables and quality measures using only single properties (representatives) of the PDF (such as the position of the peak or the variance). While the representatives category may fit for filters using Gaussian representations (like the Kalman filter), filters with more complex PDFs would benefit from full PDF comparisons.

Moreover, a literature study has shown that different variables were utilized for the behavior estimation. While we used the plausibility approach from [34], which fits to Eq. 8.7, other plausibility measures in literature compare $\mathbf{X}_{t+1}$ and $\mathbf{X}_{P_t}$ (Eq. 8.8) and other plausibility measures use only $\mathbf{X}_{t+1}$ (Eq. 8.9). We will use this equations for the categorization of the different plausibility measures.

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx Pl(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1}), P(\mathbf{X}_{P_t}|\mathbf{y}_{1:t})) \tag{8.8}$$

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx Pl(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1})) \tag{8.9}$$

In the following we will list a number of quality measures from the literature. While we cannot make a statement as to which concrete measure is best, the reader will already see that measures using Eq. 8.9 are theoretically less powerful than the other two categories because the information available is lower. Nevertheless, the value of the quality measures not only depends on the information used, but also on the concrete way the information is analyzed. In each case the satisfaction of the user with a certain quality measure depends highly on the task and the concrete situation. The evaluation of quality measures is a promising area for future research.

8.4.2 Quality Measures based on Representatives

The quality measures introduced in this subsection are rather simple. Instead of comparing whole PDFs in order to derive the best fitting model, single properties (or representatives) of the PDF are selected. Some of these quality measures originate from the point estimator theory itself.

Highest posterior peak or MAP

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx Pl(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1})) \sim \max_{\mathbf{x}_t} P(\mathbf{x}_t|\mathbf{y}_{1:t}) \tag{8.10}$$

The quality measure was evaluated in [72] in a CMM Kalman filter in order to improve the tracking of a camera observing a bouncing ball. The filter switches to the model with the highest likelihood estimated by the quality measure. For their application the highest posterior peak performs best next to the modified Kullback-Leibler-divergence. The measurement is in the category $Pl(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1}))$ and is therefore theoretically one of the weakest quality measures.

Point Estimator techniques and the Innovation Vector

Several measures can be classified under this category. First, the representative is chosen, then the distance between the representative and the measurement is determined. In the last step the distance is scaled to a probability, e.g. by a Gaussian distribution.

One possible representative is the Maximum a Posteriori (MAP) method. It is widely used in the point estimation theory [49] and is used for searching for the most probable state.

$$x^* = \arg\max_{\mathbf{x}_{t+1}} P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t}) \tag{8.11}$$

The other measure used most often is the Minimum Mean Square Error (MMSE). It sets the representative to the expectation value of the PDF.

$$x^* = E[P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t})] \tag{8.12}$$

In the next step the distance between the representative x^* and the measurement y_{t+1} is determined. This is the so called *innovation vector*. When the innovation vector is small, "the sensor measurements are well explained by the model" [42]. The term innovation vector originates from the Kalman filter theory. In Kalman filters both of the above measures are identical. Multiple-model Kalman filters using the innovation vector approach are used, for example, for automotive ego-localization in [42].

$$d_{Inno} = y_{t+1} - x^* \tag{8.13}$$

The scaling in the innovation vector approach is done via a Gaussian distribution.

$$\begin{aligned} P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx & Pl(P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \\ \sim & N(d_{Inno}, 0, \sigma(P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t})) + \sigma(P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \end{aligned} \tag{8.14}$$

The standard deviations σ of the predicted estimate and of the measurement are calculated and their sum is the standard deviation of the scaling Gaussian distribution.

The quality measure belongs in the first category (Eq. 8.7).

Quotient of standard deviations of prior to posterior

This measure was stated and evaluated in [72]. They reason that a decrease in the standard deviation from the prior to the posterior estimate indicates a measurement that is consistent with the prediction. Due to the information used it is a quality measure of the second category. The PDF is abstracted to the standard deviations. It is therefore based on representatives.

$$\begin{aligned} P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx & Pl(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1}), P(\mathbf{X}_{P_t}|\mathbf{y}_{1:t})) \\ \sim & \frac{\sigma(P(\mathbf{X}_{P_t}|\mathbf{y}_{1:t}))}{\sigma(P(\mathbf{X}_{t+1}|\mathbf{y}_{1:t+1}))} \end{aligned} \tag{8.15}$$

Measured Position Estimate Probability (MPEP)

The MPEP was developed by us for our first behavior detection evaluations. It probes the estimated probability at the sensed position. It is similar to the innovation vector approach, but takes the non-Gaussian estimate PDF into account. The drawback is that it does not account for the sensor model and probes the estimate PDF at a single representative position.

$$\begin{aligned} P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx & Pl(P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \\ \sim & P(\hat{\mathbf{x}}_{P_t} = y_{t+1}|\mathbf{y}_{1:t}) \end{aligned} \tag{8.16}$$

Although the plausibility measure is superior to this measure, it is well-suited for probing the estimate at the ground truth state x_{real} (cf. Eq. 8.17), as in simulations when the ground truth state is known. We will explicitly denote those evaluations as MPEP_{real}.

$$P(\hat{\mathbf{x}}_{P_t} = x_{real_{t+1}}|\mathbf{y}_{1:t}) \tag{8.17}$$

Note that even MPEP_{real}, despite knowing the real position, will also deliver a likelihood that a certain model is correct, and not the likelihood that a certain mode is true. But in simulated environments the driver will in most cases, drive reasonably and therefore in accordance to the modeled behavior.

8.4.3 Quality Measures based on PDFs

These quality measures compare whole PDFs with each other in order to evaluate how well the PDFs fit to each other, or how likely one PDF develops from another PDF and a behavior model.

One of these measures is the (modified) Kullback-Leibler-divergence [72], comparing prior and posterior distribution. The plausibility introduced in [34] is also a measurement which compares two PDFs. In our evaluations we

first used the MPEP measure introduced in the subsection above, and later replaced that measure with the plausibility measure. Generally, Bayesian filters that are able to represent arbitrary PDFs instead of Gaussian PDFs should be able to benefit from the additional information that a whole PDF provides. Therefore, we limit our evaluations to quality measures based on PDFs. Counterexamples showing better results of quality measures based on representatives instead of PDFs in certain situations do not prove the opposite.

(Modified) Kullback-Leibler-divergence

The Kullback-Leibler-divergence [29] measures how well the posterior fits to the prior distribution by using the entropy measure from information theory. Eggert [72] argues that a "higher [Kullback-Leibler-divergence] refers to a stronger decrease of entropy of prior to that of posterior due to a reliable likelihood which is consistent with the prediction". Since it is "easier for a prior with a higher standard deviation to get a larger change towards posterior" they introduced the modified Kullback-Leibler-divergence that is normalized by the standard deviation of the prior estimate.

$$P(b|\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{y}_{t+1}) \tag{8.18}$$

$$\approx Pl(P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}), P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t}))$$

$$\sim \frac{\int P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}) \log(\frac{P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})}{P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t})}) d\mathbf{x_{t+1}}}{\sigma(P(\mathbf{X}_{P_t}|\mathbf{y}_{1:t}))} \tag{8.19}$$

Scalar product of prior and posterior

The scalar product of prior and posterior distribution was evaluated as quality measure in [72]. When the overlap between the prior and the posterior distribution is low, the scalar product is small. A higher overlap generates a high scalar product.

$$P(b|\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{y}_{t+1}) \tag{8.20}$$

$$\approx Pl(P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}), P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t}))$$

$$\sim \frac{P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}) \cdot P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t})}{||P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})|| \cdot ||P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t})||} \tag{8.21}$$

$$\tag{8.22}$$

with

$$||P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})|| = \sqrt{P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}) \cdot P(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})} \tag{8.23}$$

$$||P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t})|| = \sqrt{P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t}) \cdot P(\mathbf{x}_{P_t}|\mathbf{y}_{1:t})} \tag{8.24}$$

A drawback of this measure is that a very wide sensor distribution cannot produce the same overlap as a rather accurate sensor distribution. Such effects are normalized in the plausibility measure of [34] in Sec. 8.4.3. We preferred that plausibility since in our application the sensor model has to account for higher noise in higher distances between the ego-vehicle and the observed vehicle and the scaling of the measure should not depend on these distances.

Plausibility

The plausibility measure is the quality measure used mainly in this thesis. Unless otherwise stated Pl stands for this plausibility measure which was originally formulated by [34]. Parts of this subsubsection have been originally published in [7].

The maximum possible overlap with the sensor model is used for normalization. It can be seen as a shape-independent scalar product. The benefit of this measure in our application is that the output is independent of the form of the sensory noise, which is not constant but declines with shrinking distance between us and the observed object. We introduced the already discretized plausibility approach in Eq. 8.25. The continuous approach uses integrals instead of sums and can be reviewed in the original literature [34].

$$Z_{\mathbf{d}} = \sum_{i}^{N} P(\hat{\mathbf{x}}_{t+1}|\mathbf{y}_{t+1} + \mathbf{d}) P(\hat{\mathbf{x}}_{P_t}|\mathbf{y}_{1:t}) \tag{8.25}$$

$$Z_{max} = \max_{\mathbf{d}} Z_{\mathbf{d}} \tag{8.26}$$

$$P(b|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \approx Pl(P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \tag{8.27}$$

$$\sim Pl = \frac{Z_{\mathbf{d}=\mathbf{0}}}{Z_{max}} \tag{8.28}$$

Equation (8.25) is a discrete convolution of the predicted state (second factor) and the sensor model (first factor) shifted by the vector $\mathbf{d}$. It is therefore a quality measure of the first category. The convolution is executed over the whole state space $\hat{\mathbf{x}}_{t+1}$. In (8.26) the maximum over all possible shifts is calculated, which is used for normalization in (8.27). The intuitive output of the calculation is the convolution (or overlap) of the sensor distribution and the predicted state distribution normalized by the highest possible overlap an optimal fitting sensor measurement could produce. For example, if the sensor creates the highest possible overlap with the predicted state, the sensor fits best with the predicted state, therefore receiving $Pl = 1$. Despite that, if the sensor model shifted by an optimal $\mathbf{d}$ would fit better than the sensor itself, Pl receives a smaller value by the normalization term in (8.27).

8.4.4 Summary

There are several quality measures and choosing the correct quality measure depends highly on the task and the concrete situation. In our following evaluations we use the plausibility originally formulated by [34]. Quality measures based on PDFs have advantages over quality measures based on representatives when arbitrary probability distributions are compared. The representation of the ICUBHF generates such arbitrary distributions. In comparison to other PDF based quality measures, such as the scalar product of prior and posterior, the plausibility measure has the advantage that it normalizes the plausibility output by the maximum possible overlap. Due to this fact, the quality measure is independent from the sensor model width and therefore independent from the distance to the other vehicle. In some evaluations we have also used the MPEP and the MPEP_{real} quality measure. This evaluations are explicitly denoted.

Chapter 9

Behavior Tracking in Street Environments using the ICUBHF

In the previous chapter we introduced behavior tracking in a generic manner. This section deals with behavior tracking in street environments applying the ICUBHF approach.

Why does the ICUBHF needs model selection? This question arises, when using the filter only for the position tracking and not for behavior detection. Instead of running different filters with different models, an overall filter dealing with all behaviors could be created. This overall generic behavior filter can be simply achieved by creating a single prediction model considering all behavior types simultaneously. But in addition to the fact that a multiple-model approach allows us to categorize the current behavior for subsequent ADAS components, the multiple-model approach also delivers a better vehicle tracking result. Applying an overall model like the kinematic model to a vehicle that is driving in a street curve is not optimal, since the steering in the curve direction will be constant over a certain time span. A filter selecting the likeliest model (or weighting the models by their likelihood) can use the turning model in that situation, and therefore provides much better predictions compared to an unspecific model. The behavior model B is available as a quite constant state allowing for improvement of the prediction and therefore the tracking.

The exceptional feature of the ICUBHF, is that these behavior models exceed the complexity of the state-of-the-art approaches. That means we do not use the usual short-term models like deceleration, acceleration or direction change. The ICUBHF approach allows medium-term models via the attractor concept and therefore allows medium-range predictions. It allows the estimation of not only primitive modes, e.g. if a vehicle changes the velocity, but also more complex modes like if a vehicle follows a certain

lane or if the typical behavior for turning is executed. The determined behavior can then be used to predict the future state of the vehicle (maybe in a prediction only loop, cf. Sec. 4.3) based on the appropriate behavior model.

In the first section of this chapter we show how the ICUBHF is influenced by the behavior modeled by the attractors. We show, on the basis of Bayesian network graphs, the overall concept of dependencies between the behavior of the driver and the state of the vehicle. After this we introduce the Bayesian network of the ICUBHF approach and show how the quality measure/plausibility measure can be used to compare PDFs in order to detect the behavior model and the behavior mode. Next, we show evaluation results of the behavior detection. The chapter concludes with a survey of the latest research in behavior modeling and behavior detection.

9.1 Using the Attractors to Generate an Overall Representation

By implementing behavior modes and behavior models with the help of the attractor approach, the Bayesian network of the ICUBHF increases in complexity compared to the MDBHF. The resulting Bayesian network graph can be interpreted in different and interesting ways. In the first subsection we will show the Bayesian network graph in different detail or abstraction levels. In the next subsection we will show which PDFs we compare with the quality measure/plausibility measure in order to select the right behavior model. In the next subsection we make the step from the behavior model B tracking to the behavior mode $\hat{B}$ tracking.

9.1.1 Our Multiple-Model ICUBHF Setup

In the ICUBHF additional variables were introduced. For the sake of clarity it is useful to aggregate some variables or parts of the Bayesian network. There are different ways to interpret the ICUBHF as a Bayesian network. We will start with the most detailed graph and simplify it in the subsequent subsubsections.

Detailed Interpretation

Fig. 9.1 shows a very detailed interpretation of the ICUBHF. The new variables are the behavior model B_t, the behavior mode $\hat{B}_t$, the context C_t and the context sensor $C_{sensor,t}$.

The driving behavior model B_t determines the action a driver is doing at the moment, which depends on the context C_t and on the current vehicle state (the location l_t and velocity vector v_t). The current driving behavior determines the yaw-rate and acceleration, since the vehicle driver performs

actions in order to achieve the goal of his behavior. The goal is implemented as the attractor state in the behavior model. However, the driving behavior B_t which is modeled by the attractors is not only dependent on the vehicle state and the context, it also depends on the unknown ground truth behavior mode $\hat{B}_t$. $\hat{B}_t$ itself is driven by the unknown overall goal of a driver. This goal could be the journey destination, an intermediate goal according to the driving route (an intersection exit etc.) or a behavioral goal such as malicious driving, crazy driving, fast destination reaching or overcautious driving. Note that while the behavior mode categorization is artificial and can be done using different criteria, the influence of the mode on the model is real. Due to the overall goal, $\hat{B}_t$ also depends on $\hat{B}_{t-1}$ since the behavior mode is constant over a certain time or the transition probability is known. For example, the behavior mode "deceleration" is often followed by a "drive a curve"-mode, and each mode will be fixed over a time-span. The different $\hat{B}$ are modeled by the behavior model B_t which itself depends on the context C_t. When it models curve-driving it needs to compare the vehicle state with the lane position (cf. the attractor function). C_t is not directly known, but sensed by a lane detection sensor or by localization in a map $C_{sensor,t}$. But in our implementation we do not explicitly model noise in the lane like in this graph. The noise in the lane position can be modeled by adding the ego-localization noise onto the sensor noise in $P(X|Y)$.

Note that the overall goal is not modeled by us, but this can be useful, e.g. when using vehicle-to-vehicle communication. In concrete terms this means that the destination in the navigation device may give a priori assumptions on the behavior mode.

It has already become clear that the behavior mode $\hat{B}_t$ cannot be detected directly, but it is possible to measure how well the sensor input matches the predicted model B_t and then estimate the behavior mode. This is the topic of Sec. 9.1.2.

Interpretation: Behavior model likelihood does not depend on vehicle state

This interpretation (cf. Fig. 9.2) focuses on the behavior model likelihood and does not interpret the behavior model as the attractor function. v_R is the output of the attractor function receiving the context and the behavior model likelihood. This means the output of the behavior model, which are basically the attractor defined velocities (the effect of the behavior model), are still depending on the previous vehicle state. But the overall behavior model likelihood is considered to be independent from the vehicle state. For example, a vehicle driver rarely spontaneously changes his behavior when his vehicle is at a certain position in the lane. When the vehicle is leaving the lane his intention does not change spontaneously, but the leaving of the lane was already intended by the driver. It was already derived by the overall

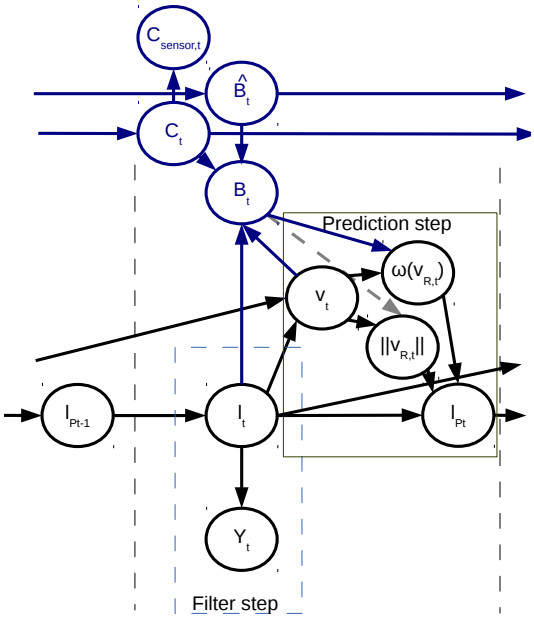


Figure 9.1: Detailed ICUBHF Bayesian model interpretation for an individual grid cell: l is the position variable, it depends on the previous position, the sensor input Y, and the velocity v. The velocity is then transformed using the behavior model B and the ego-movement. The ego-movement compensation is not visualized in the graph. The behavior model B is implemented by the attractor function that assigns a new velocity direction $\omega(v_R)$ and absolute velocity $||v_R||$ to each grid cell position l, depending on the context representation C and the behavior mode $\hat{B}$. The context itself is measured by a context sensor C_{sensor}. The black arrows illustrate the MDBHF approach, the blue arrows illustrate the extension needed in the ICUBHF approach. The dashed line is not implemented, since absolute velocity changes are not modeled in our attractor implementation.

driving goal which determines the behavior mode $\hat{B}$. Of course, this is only approximately correct, for example, when the driver changes his behavior mode during an unforeseen event.

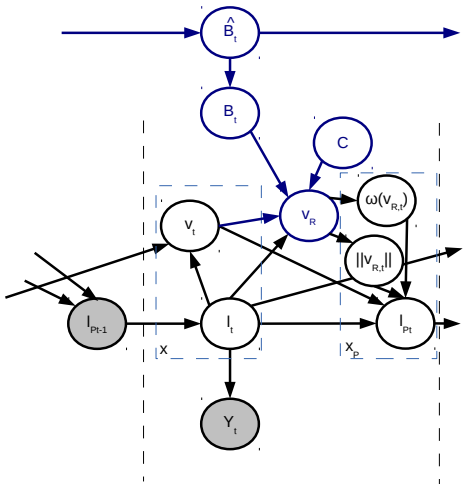


Figure 9.2: Alternative ICUBHF Bayesian model interpretation for an individual grid cell: In contrast to the illustration in the previous figure, the behavior model is not interpreted as the attractor function itself. The attractor function is receiving the behavior model B as an input variable and changes the velocity v to the new velocity v_R with usage of the context C. B is therefore not depending on the location and the velocity. This means that the behavior B is depending on the behavior model only.

Autonomous Multiple-Model ICUBHF

Our multiple-model ICUBHF is modeled as AMM (cf. 8.2). This means that the behavior switch is not modeled directly in the ICUBHF. Separate ICUBHFs are used independently. Figure 9.3 shows the ICUBHF from the autonomous implementation perspective. It also illustrates that the position and velocity can be aggregated to estimate x and that the attractor velocities v_R and the predicted location l_P can be aggregated to x_P. The behavior models do not interact with each other as illustrated by the separate array of behavior variables.

9.1.2 Model Comparison in the ICUBHF

The reader now has insight into different abstraction levels when looking at the autonomous multiple-model ICUBHF (AMM-ICUBHF). For the behavior detection we applied the implementation according to Fig. 9.3.

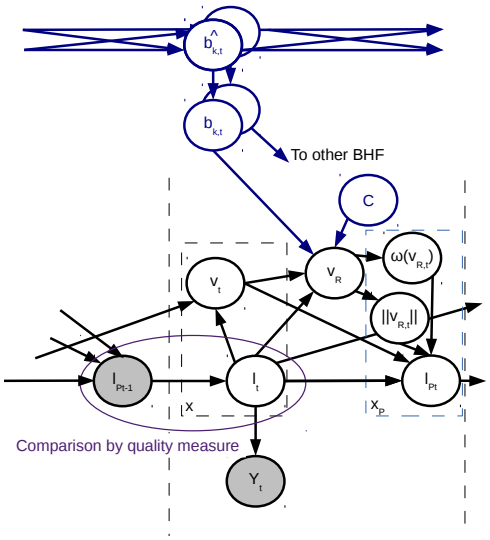


Figure 9.3: AMM-ICUBHF Bayesian model interpretation for an individual grid cell: In contrast to the previous figure, the ICUBHF is illustrated as an AMM approach. This means that the individual MDBHF (black) depends on an individual behavior model b_k and behavior mode $\hat{b}_k$ instead of the whole behavior space $\hat{B}$. Each MDBHF used represents an individual behavior mode $\hat{b}_k$ only. Behavior mode changes are not modeled by the BHF itself, and therefore a HMM approach is needed for the behavior mode changes (blue). This Bayesian model interpretation fits best to our ICUBHF implementation. We also marked the two random variables that are compared by the quality measure in order to estimate the behavior model.

Now we want to know which behavior fits to the input vector $P(B_t|\mathbf{Y}_{1:t+1})$. Due to the Markov assumption, all previous sensors are condensed into the state $\mathbf{X}_{P_{1:t}}$ or, as used here, $\mathbf{l}_{P_{1:t}}$. This means that $P(B_t|\mathbf{l}_{P_t}, \mathbf{Y}_{t+1})$ needs to be used for behavior detection. The variables are marked in gray in the Bayesian network graph. The PDF $P(B_t|\mathbf{l}_{P_t}, \mathbf{Y}_{t+1})$ is by definition not given. It needs to be chosen by using the quality/plausibility measure introduced in Sec. 8.4. The plausibility measure evaluates how well two PDFs fit together. To do this we compare the state $\mathbf{X}_{P_t}$ or position $\mathbf{l}_{P_{t-1}}$ PDF with the state or position that is based on the newest measurement only ($P(X_t|Y_t)$ and respectively $P(l_t|Y_t)$). It is important to note that the predicted estimate PDF contains the information of all input sensor inputs $\mathbf{Y}_{1:t}$ while the second PDF applies the inverse sensor model to the latest sensor input and therefore contains only the information in $\mathbf{Y}_{t+1}$.

The quality measure now compares both PDFs and therefore identifies how well the model based predicted estimate fits the new sensor input. A more precise behavior estimate could be calculated when doing a Bayesian smoothing instead of a Bayesian filtering. Therefore, the predicted state estimate PDF containing all previous sensor inputs is not only compared to the state estimate using the latest time step $t + 1$ but to the estimate state knowing about all (or at least a number of) future time steps. One reason not to implement a smoothing is that an online-detection of the behavior is needed in ADAS systems. Each smoothing, also one with a low number of future measurements, would decrease the value of the behavior detection output, because the information from the past that is incorporated into the smoothing will be inevitably already somewhat outdated. A new kind of prediction would be needed that adds new inaccuracies. We therefore limit our view to Bayesian filtering.

The comparison of the two PDFs is visualized by the oval shape in Fig. 9.3. The comparison by the plausibility measure in time step $t + 1$ can be generally stated by Eq. 9.1. Since our ICUBHF measures positions but not velocities, we use Eq. 9.2, which limits the PDF comparison to the position estimate PDF.

$$P(B_t|\mathbf{Y}_{1:t+1}) := \eta Pl(P(\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1})) \tag{9.1}$$

$$P(B_t|\mathbf{Y}_{1:t+1}) := \eta Pl(P(\mathbf{l}_{P_{t-1}}), P(\mathbf{l}_t|\mathbf{y}_t)) \tag{9.2}$$

The variable η is a normalization constant that can be easily determined by an integration over all model plausibilities.

9.1.3 From the Behavior Model to the Behavior Mode

The plausibility measure delivers the probability that a certain behavior model fits the input vector $P(B_t|\mathbf{Y}_{1:t+1})$. Since the behavior mode cannot be measured directly we make use of the Bayesian network graph (Fig. 9.3) to

recursively estimate the behavior mode from the current detected behavior model and the previous behavior mode. The resulting equation (Eq. 9.3) emerges from the graph.

$$P(\hat{B}_t|\mathbf{Y}_{1:t+1}) = P(\hat{B}_t|\hat{B}_{t-1})P(\hat{B}_{t-1}|\mathbf{Y}_{1:t})P(\hat{B}_t|B_t)P(B_t|\mathbf{Y}_{1:t+1}) \tag{9.3}$$

$$P(\hat{b_t}|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) = \sum_{\hat{b}_{t-1}} P(\hat{b}_t|\hat{b}_{t-1})P(\hat{b_{t+1}}|\mathbf{X}_t, \mathbf{X}_{t-1}, \mathbf{y}_t) \\ P(\hat{b}_t|b_t)P(b_t|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1}) \tag{9.4}$$

Equation 9.3 is a recursive equation determining the behavior mode $P(\hat{B}_t|\mathbf{Y}_{1:t+1})$ by the previous behavior mode $P(\hat{B}_{t-1}|\mathbf{Y}_{1:t})$ that is projected into the future using the transition model $P(\hat{B}_t|\hat{B}_{t-1})$. $P(B_t|\hat{B}_t)$ accounts for the fact that the prediction model does not exactly match the prediction mode. Using the behavior detection by the quality measures from Sec. 8.4.1, the equation can be written as the recursive notation Eq. 9.4.

When the likelihood of the behavior model given the behavior mode is not known, it can be transformed using the Bayesian theorem Eq. 9.5, but in most cases both PDFs will be guesses.

$$P(B_t|\hat{B}_t) = \frac{P(B_t)}{P(\hat{B}_t)} P(\hat{B}_t|B_t) \tag{9.5}$$

The recursion needs an initial value or a priori probability on each behavior mode $\hat{b} \in \hat{B}$. The a priori behavior distribution $P(\hat{B}_0)$ can be either set to a uniform distribution or a worst-case mode can get assigned the full probability: $P(\hat{b}_{worst_0}) = 1$. This means that the behavior estimator assumes the worst case behavior to be true in the beginning. The estimator believes in the worst-case behavior until there is clear evidence against the initial assumption. This is used excessively in our evaluations, e.g. when an oncoming vehicle approaches an intersection and we want to analyze in which direction the vehicle will drive. The worst case behavior in this situation is the behavior which leads to an interception of the trajectories of the ego-vehicle and the observed vehicle, and hence implies the risk of an accident. By assigning the worst-case behavior the highest initial likelihood in the behavior estimator, the estimator needs strong evidence before assigning a low probability to the worst-case behavior in ambiguous situations. Additionally, a hysteresis is assigned to the detector, before switching its deterministic output to another behavior. This means that we do not apply Eq. 8.1 directly. The detected behavior mode $\hat{b^*}$ does not switch directly when another behavior mode seems to be correct with a higher likelihood.

The detection switches to a mode $k = 2$ from a mode $k = 1$ when the following equation is true:

$$P(\hat{b_{k_t=2}}|\mathbf{y}_{1:t+1}) > P(\hat{b_{k_t=1}}|\mathbf{y}_{1:t+1}) + \theta \tag{9.6}$$

This hysteresis or threshold θ avoids oscillation between various modes during ambiguous moments.

The recursive estimation of the behavior mode with the HMM model, constructed in Eq. 9.3, has two main advantages. First, it accommodates for the fact that the behavior mode is not the behavior model. The behavior mode $\hat{B}$ is probabilistically assigned by measured behavior model B by the term $P(\hat{B}_t|B_t)$. The HMM is doing a kind of averaging over time and this averaging improves the estimation since the behavior model will fit better or worse to the behavior mode, depending on the position in the state space in relation to the context. That means the quality of the behavior model itself, which states how well the model fits to the actual driver behavior, will fit better in the average case than in some specific time steps. Second, the mode itself can change over time. The HMM takes care of that with the $P(\hat{B}_t|\hat{B}_{t-1})$ term. In our evaluations we use a combined PDF $P(\hat{B}_t|B_t, \hat{B}_{t-1})$ for the uncertainties $P(\hat{B}_t|B_t)$ and $P(\hat{B}_t|\hat{B}_{t-1})$.

The behavior detection by the AMM-ICUBHF is evaluated in the next section of this chapter. However, first we want to add another factor to the recursive behavior estimation (Eq. 9.4).

9.1.4 Reachability as Additional Factor

The likelihood that a certain behavior mode is correct $P(\hat{b_t}|\mathbf{X}_{t+1}, \mathbf{X}_t, \mathbf{y}_{t+1})$ is never 1, since the mode can change from the last time step to the current time step and the behavior model does not exactly match the behavior mode. This is represented by the PDF $P(\hat{B}_t|B_t, \hat{B}_{t-1})$.

Additionally, $P(\hat{B}_t|B_t)$ assigns a likelihood to behavior modes that are no longer reachable. We therefore augment the PDF with a reachability correction R. We will first explain how this is achieved and then discuss the effects.

Implementation of the reachability

$$r_{i,t} = N(\omega(\mathbf{v}^A_{i,t}) \ominus \omega(\mathbf{v}_{i,t}), 0, \frac{\beta_{max}}{3}) \tag{9.7}$$

$$\alpha \ominus \beta \leftarrow (\alpha - \beta + \pi)\%(2\pi) - \pi \tag{9.8}$$

$$R_t = \frac{\sum_i r_i * p_{i,t}}{\sum_i p_{i,t}} \tag{9.9}$$

The attractor reachability of a certain grid cell is $r_{i,t}$ (Eq. 9.7). A Gaussian model is used to quantify the reachability in each grid cell. The argument of

the Gaussian is the smallest angle between the attractor direction $\omega(\mathbf{v}_{i,t}^{A})$ and the original velocity direction in the grid cell $\omega(\mathbf{v}_{i,t})$. The maximum steering angle β_{max} was chosen as 3σ. A possible definition of the shortest angle $\ominus$ is stated in Eq. 9.8. The attractor reachability is weighted in each cell with the probability mass in the grid cell to achieve the overall reachability in time step t (Eq. 9.9).

The overall reachability is multiplicatively incorporated in Eq. 9.3 in the $P(\hat{B}_t|B_t)$ term, or respectively, in Eq. 9.4. The resulting behavior estimation using the reachability is denoted as $P(\hat{B}_t^{Reach}|\mathbf{Y}_{1:t+1})$ in our evaluations.

Effects of the reachability scaling

With higher steering costs and decreasing reachability of an attractor point, the probability decreases that the attractor and thereby the behavior model is a realistic representative for the behavior mode. This is intuitive, since the attractor-defined behavior model fits the behavior mode only as long as the behavior mode is possible (or reachable). For example, the attractor behavior model does not model a turning behavior well when the turning can no longer be executed, since the intersection is nearly passed. The behavior mode "turning" becomes obsolete. The fact that the behavior mode is no longer possible is not represented by the model itself but by the probability $P(\hat{B}_t|B_t)$, which states how likely a certain mode is given a certain behavior model.

In other words, the behavior model no longer models a turning when the intersection is passed or nearly passed, since the attractor points are no longer reachable (cf. Eq. 7.3,7.5,7.4). The behavior model degenerates to a kinematic model when the reachability constraints are no longer fulfilled. The reachability scaling in $P(\hat{B}_t|B_t)$ accounts for that effect and enables us to do a meaningful assignment from the behavior model to the behavior mode.

9.2 Evaluations

In the evaluation section we will first introduce the typical graphs used in all following evaluations, thereby avoiding the need of multiple explanations. In the following subsection we will then show the behavior detection during a lane cut-in. The subsequent experiment deals with an oncoming vehicle in an intersection scene. Portions of the subsections have been published in [6] and [7].

9.2.1 Overview over the Typical Graphs

The behavior detection process can be separated in up to four phases.

1. The first phase is the transient state. The behavior estimation adjusts from the a priori assumption $P(\hat{B}_0)$ to the "sensed" behavior estimate. The duration of the phase depends on the inertia implied by $P(\hat{B}_t|B_t, \hat{B}_{t-1})$.

2. The second phase is optional. It occurs only when the compared models are identical in the temporary relevant area of the state-space. Whenever two models are temporarily identical the estimated behavior persists in a uniform distribution. The detected behavior will be due to the hysteresis θ not switching until the models separate.

3. The third phase occurs when one model becomes more probable than the other. When the probability that a certain model is correct rises, the mode becomes more and more likely and the hysteresis will finally be exceeded. When the mode with the higher plausibility is not the prior assumption mode, the behavior detection changes. If not, the a priori mode is still the detected mode.

4. The fourth phase occurs when a model no longer models the belonging mode and the models are identical again at the relevant area in the state space. The relevant area are the grid cells with high assigned likelihoods. Due to the hysteresis, this has no negative effects on the behavior detection, but can be compensated by using $P(\hat{B}_t^{Reach}|\mathbf{Y}_{1:t+1})$ instead of $P(\hat{B}|\mathbf{Y}_{1:t+1})$ (cf. Sec. 9.1.4).

The quality measure graph

This graph shows the result of the quality measures over time. It shows the non-normalized output of the plausibility measure, e.g. $Pl(P(\mathbf{X}_{t+1}|\mathbf{X}_{P_t}), P(\mathbf{X}_{t+1}|\mathbf{y}_{t+1}))$. Usually the quality measure of more than one model is depicted. The graph is easy to understand and is used in the next evaluation (Fig. 9.8). The standard deviation shows the deviation of the quality measure over the different runs with a different noise series ϵ_t in $y_t = x_{true_t} + \epsilon_t$. It is necessary to do several runs in order to check the robustness against sensor noise.

The mode plausibility graph

This graph shows the plausibility or probability of certain modes over time, meaning that it shows $P(\hat{B}_t|\mathbf{Y}_{1:t+1})$ (cf. Eq. 9.3 or Eq. 9.4) observed over time (cf. Sec: 9.1.3). Usually we draw the average value over several runs with a different noise series ϵ_t in $y_t = x_{true_t} + \epsilon_t$. The variance is shown at every few time points. Again, it is necessary to do several runs in order to check the robustness against sensor noise. Of course, it is only possible to do so in this manner in simulations, since in real world scenarios the trajectory $x_{true_{1:t}}$ cannot be set constant over several runs under real-time conditions. Therefore, simulations provide a good fundamental testing environment.

The mode plausibility graph can be shown for several a priori settings $P(\hat{B}_0)$. Usually the graphs that show a wrong a priori setting (non-conform) are more interesting than the graphs that show a correct a priori setting (conform), because the non-conform ones produce a changing behavior detection in the behavior detection graph (cf. 9.2.1)

In the example graph (Fig. 9.4) we implemented a reachability scaling. In most evaluations we do not use the reachability scaling, since the reachability masks the effects of the plausibility measure. Masking is not desirable since the plausibility measure applied on filters with different behavior models, designed by attractors, is our key innovation.

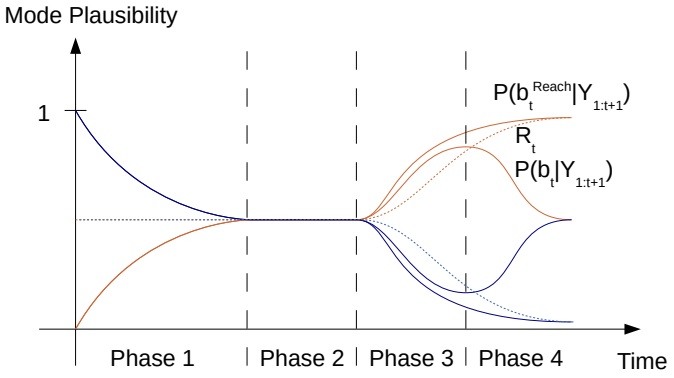


Figure 9.4: The (mode) plausibility graph showing $P(B_t|\mathbf{y}_{1:t+1})$. In some situations we will show a special case of the plausibility graph, as additionally illustrated in the figure as graph $P(B_t^{Reach}|\mathbf{y}_{1:t+1})$. In either case the overall mode plausibility adds up to 1. Therefore we have in the two mode cases two mirrored curves. Additionally, the reachability scaling R_t is illustrated as dashed curve. An algorithm using the reachability only instead of the plausibility would output the reachability values.

Special case: The mode plausibility graph at the ground truth position

This graph is well-suited for debugging purposes in simulated environments. Instead of observing $P(B_t|\mathbf{y}_{1:t+1})$, the $P(B_t|\mathbf{x}_{\mathbf{real}1:t+1})$ is observed. It can thereby be detected whether the sensor noise or the prediction model is the reason for wrong detections. In previous chapters we denoted this measure in a simplified way as $P(X = x_{real})$ in order to achieve an intuitive explanation there, and a compact introduction of all measures in the current chapter.

The behavior (mode) detection graph

This graph shows the detected behavior mode $\hat{b}^*$ over time using Eq. 9.6. It shows the number of runs with different noise series ϵ_t, where the detection algorithm decided that a certain behavior mode is true over time. When the mode changes because the a priori mode has proved as non-conform, then over time more and more evidence aggregates for the new behavior mode, and therefore the detected mode changes. Of course, not all runs switch the modes at the same time, since the noise can mask the true behavior for a while. This means that the detection time varies from run to run. It is also possible that in some runs the detected mode does not change to the ground truth mode at all or oscillates back over time. The reason for this lies in the fact that the behavior model does not match the behavior mode of the driver exactly or that the differences between the models is too small in comparison to the noise level. It is also possible to some extent that approximation errors in the grid representation are the reason for that in a few runs. However, these approximation errors can also be interpreted as being a deviating (discrete) model, which does not exactly match the real behavior mode. Therefore, the HMM approach in 9.3 is already the best way to cope with that effect. In border cases it can help to adjust the parameters of the PDFs in Eq. 9.3 and, respectively, Eq. 9.4 in order to make the behavior detection more sensitive or more robust. The PDF $P(\hat{B}_t|B_t, \hat{B}_{t-1})$ can be adjusted to have a higher inertia, which makes it more robust but the detections are delayed since more evidence needs to be accumulated before the plausibility increases. The same applies to a higher hysteresis θ, but theta not only regulates the number of evidence needed, but also gives a minimum limit on the strength of the evidence. This essentially also implies that when in the two mode cases both models are very similar, the minimum evidence may be never exceeded even when some evidence is present for a long time. Lower inertia and lower *theta* values improve sensitivity and the detection is detected shortly after the first evidence, but it is therefore prone to wrong evidence induced by sensor errors or temporary/local deviating behavior models.

False positive and false negative mode switches can be discovered in the behavior detection graph. A false positive mode switch is detected when a run switches (temporary) to the non-conform mode even when the non-conform mode is not the true mode. A false negative is seen in the graph when not all runs detect the new behavior in the end.

9.2.2 Behavior Tracking in a Cut-in Lane Event

ADAS systems are able to drive partly autonomously in highway scenarios, meaning they are able to stay in the lane and to assert a minimum distance between its vehicle and the vehicle ahead. They also have to deal with

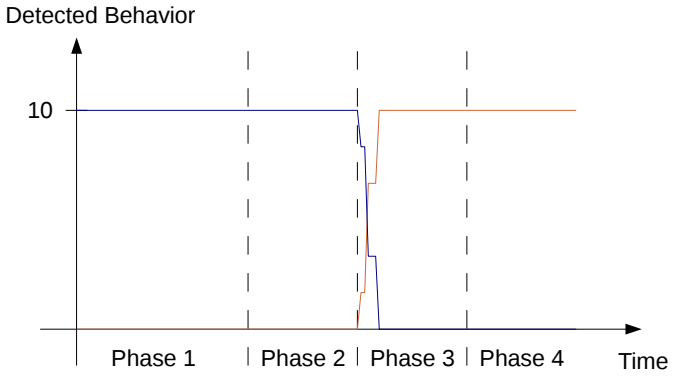


Figure 9.5: The detected behavior graph shows in how many runs, a certain behavior mode $\hat{b}^*$ is assumed to be true at a given time. The graph shows a non-conform setting fitting to the plausibility graph in Fig. 9.4. In the beginning of phase 3 the first AMM-ICUBHFs switch to the assumption that the non-conform mode is right, until the new behavior mode was finally detected in all 10 runs. When the prior mode assumption is conform to the mode the detected behavior graph should show two constant lines, since no change in the behavior detection should occur.

unforeseen situations. In current systems the vehicle actions are determined by rather simple rules. The ego-vehicle tries to keep distance to the vehicle ahead without object representation, and when the parameters are not right, the control is returned to the human driver. One example of an unforeseen situation in a highway scenario is the sudden cut-in of another vehicle into our ego-lane. This scenario is illustrated in Fig. 9.6.

To implement a cut-in or lane-change detection into an object-based tracking we use our AMM-ICUBHF with two different movement models. The attractor approach is implemented in a way that allows us to build fitting models in two different ways, and therefore two different classifications of the categories for the modes.

1. Mode A: Lane-keeping, Mode B: Lane change (or kinematic model)

2. Mode A: Keep lane no. 1, Mode B: Keep lane no. 2

The mode is always an artificial term, that allows to subsume different trajectories under that abstract concept. The assignment from a trajectory to a certain mode depends on the spectator. With both kinds of mode classifications we can create a lane change detection.

Fig. 9.7 shows the multi-modal estimate of a filter using the lane-keeping model during a lane change. This is model A in classification 1.

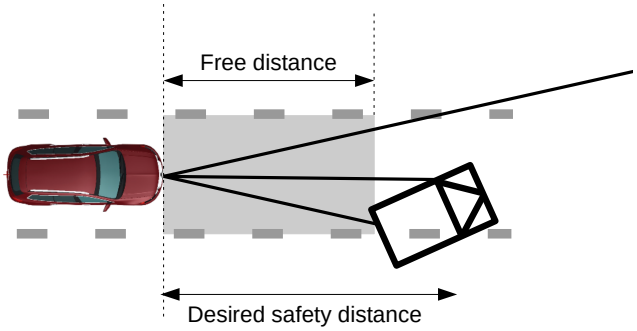
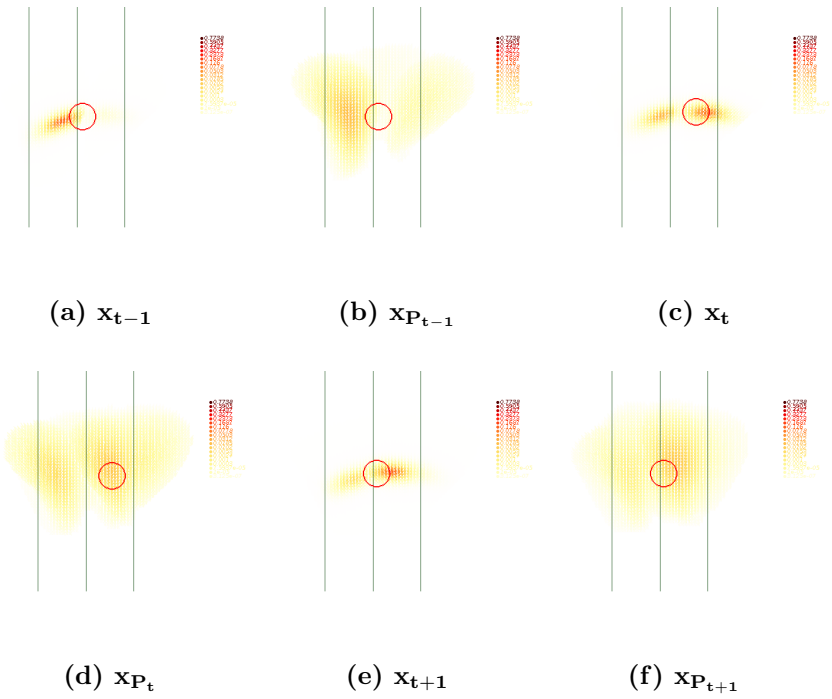


Figure 9.6: An ADAS without vehicle or object representation is measuring the free distance in front of the (red) ego-vehicle by using the minimum distance the sensors measure. When a vehicle cuts into the ego-vehicle lane and the remaining minimum free distance is smaller than the desired safety distance (such as in this example) the ADAS reacts to the situation.



(a) $\mathbf{x_{t-1}}$ (b) $\mathbf{x_{P_{t-1}}}$ (c) $\mathbf{x_t}$

(d) $\mathbf{x_{P_t}}$ (e) $\mathbf{x_{t+1}}$ (f) $\mathbf{x_{P_{t+1}}}$

Figure 9.7: A multi-modal estimate of an ICUBHF using the lane-keeping model during a lane change from left to right. Note that this image was recorded in retrospect of the evaluation and shows a lane-change scenario which is not the evaluation scenario mentioned in this section. In time step $t-1$ the first measurement input within the right lane occurs. In the following time steps the left peak fades while the right peak gains in the estimate PDF. The predicted estimates show the lane following behavior of the attractor algorithm.

In the cut-in scenario in the following subsection we use the same classification of the modes with a lane keeping model and a kinematic model. When the kinematic model is more plausible than the lane keeping model we assume that the lane changing mode is true.

Scenario and results

In this scenario we actively compare the predictions of the model without attractor concept with the one with attractor concept, considering the lane information. The active comparison allows us to build a detector for a lane change. If the measurement fits better with the kinematic model than the attractor-based lane following model, the system can be assumed to be "surprised" and changes the behavior mode.

We evaluated the lane change detector with a cut-in scenario on a freeway with two lanes for each direction (similar to Fig. 5.6). The simulation was done with the TORCS Simulator [1]. The ego-vehicle drives 120 km/h in the left lane. The observed vehicle drives in the right lane maintaining 100 km/h while cutting into the left lane in front of the ego-vehicle. This is a frequently seen risky situation in which the ego-vehicle needs to brake quickly.

Evaluation with the MPEP measure

We use an attractor function, which models that the observed vehicle stays in its lane. The MPEP quality measure, which can be interpreted as a belief in the observed measurement, drops quickly when the lane change begins, since the measurement does not comply with the prediction model. The graph in Fig. 9.8 compares the MPEP quality measure of the lane-keeping ICUBHF with that of an ICUBHF using the pure kinematic model. Note that the HMM approach for the behavior detection was not fully developed at the time these evaluations were done, instead a behavior mode was assumed to be true when the quality measure of one behavior model was higher than the other for three time steps in a row.

To evaluate the noise robustness, we decreased the angular noise in the radar sensor to 0.04 and to 0.02 rads for comparison purposes. In the behavior detection graph in Fig. 9.9 it can be seen that the detector detects the lane change events rather well (for 0.02rads). The actual lane crossing occurs after 30 time steps. A lane crossing is defined due to the nature of the ICUBHF as the time-step, when the center of mass of the vehicle crosses the lane-marking. With increasing noise levels, more false positives occur (with 0.04rads, gray solid line). When the sensory noise can be kept at a reasonably low level, lane change events can be anticipated reliably while the monitored vehicle approaches the border of a lane.

With high noise not all lane changes were detected, but usually sooner

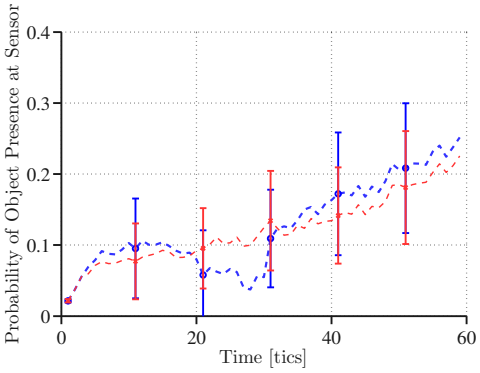


Figure 9.8: The probability the grid filter assigns to the measured position. The blue line is produced by the Bayesian filter with lane keeping model, the thinner red line is given by the Bayesian filter using the kinematic model only.

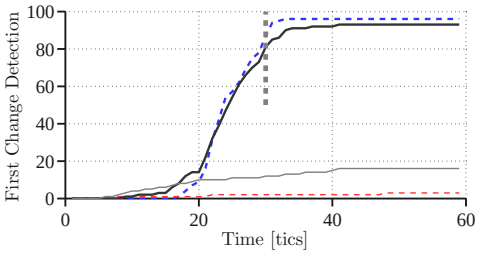


Figure 9.9: The integrated time-point of the first lane change detection in the lane changing scenario (thick) versus the lane change detection in a non-lane-changing scenario with same noise sequence (thin) showing the false positive results. Solid lines are measured with radial sensor noise of 0.04rad, and dashed lines with 0.02rad. The vertical line marks the time at the passing of the lane marking.

or later all ICUBHFs should switch to the new behavior. The reason for this is that due to the sensor noise it is possible that a lane change is missed by the quality measure. Since the behavior modes are lane keeping and the kinematic model there is no way to detect a lane change afterward. This states a difference to the keep lane 1 and keep lane 2 model, where the quality measure always gets evidence that the model 1 is violating while driving in lane 2. An evaluation of this kind is done below. Additionally, the results could be improved by taking the sensor noise into account. For this evaluation we used the MPEP quality measure which does not take the sensor model into account. If sensory noise is high, more false positives can be expected when disregarding the measurement noise distribution, for example by simply accidentally perceiving measurements on the right and left side of the lane. This was our first evaluation on behavior detection, originally published in [6]. In later evaluations we used the plausibility measure. We re-evaluated this scene with the plausibility measure at a later

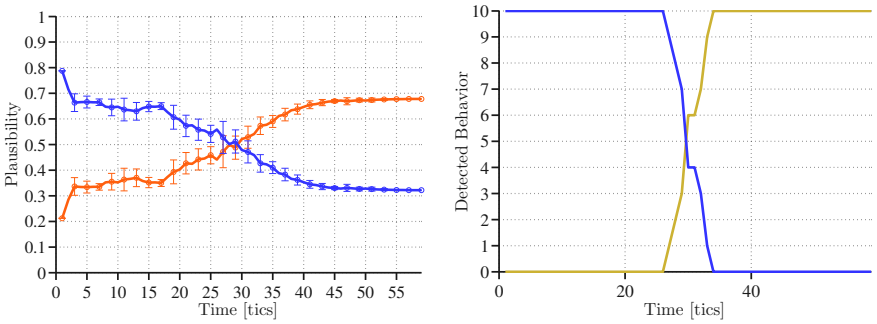


Figure 9.10: Evaluation of the cut-in scenario using the plausibility measure. In the beginning mode A is the true behavior mode $\hat{b}^*_{true}$, later mode B becomes true. Blue is the prior assumption (Keeping lane no. 1). (a) The mode plausibility graph. (b) The detected behavior graph of 10 different runs.

time. The results are published in the following subsubsection.

Evaluation with the Plausibility measure

In order to have a consistent evaluation, the cut-in scene is here evaluated with the plausibility measure. This time the second behavior mode classification categories were used. The a priori mode A is "Keep lane no. 1", the non-conform mode B is "Keep lane no. 2". The algorithm detects a lane change when mode B becomes more likely than mode A.

In the beginning, lane-keeping is the correct mode. During the lane change no mode is correct, and after the lane change mode B is the true mode. The result of the evaluation is depicted in Fig. 9.10. In comparison to the other mode categorization in the previous evaluation, this time every run detects the lane change. The drawback is that during the phase of the lane change both behavior modes from our categorization are false. With an own lane change model for the state transitions the detection time may be improved.

The ego-vehicle maintains its velocity of 120 km/h and approaches therefore the other vehicle. The plausibility graph (Fig. 9.10) shows that with decreasing distance between the vehicles, the noise in the mode plausibility graph also decreases.

9.2.3 Behavior Tracking in an Oncoming Intersection Scene

When the ego-vehicle arrives at an intersection it is necessary for the ADAS to know how the current driver is acting and how the other vehicles that are approaching the intersection will act. In the case that the human driver of the ego-vehicle does not brake in front of an intersection, the ADAS should evaluate the behavior of the other vehicles. When the ADAS detects that

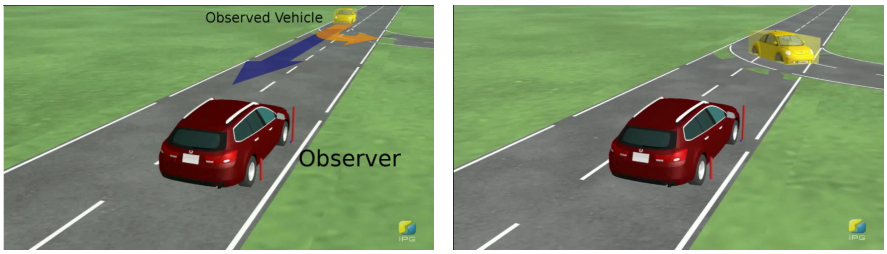


Figure 9.11: Intersection scenario. The first picture is identical in scenario mode A and B. (b) shows the turning action of scenario A at a distance of about 20-30 m.

it is likely that the ego-vehicle trajectory will intersect with another vehicle trajectory the ADAS should warn the driver or perform an emergency brake.

The ADAS can determine if two trajectories will intersect either by a pure kinematic extrapolation of the past vehicle trajectories or by detecting the behavior and then predicting according to the assumed behavior. In this evaluation we will show how such a behavior detection works with the developed AMM-ICUBHF.

Scenario description

We tested the situation with a simulated carmaker at a T-intersection scenario (cf. Fig. 9.11) in order to detect false positive and false negative detections by adding artificial sensor noise in 10 runs. This simulation is a pre-evaluation for the real-world scenarios executed later. To reiterate, the advantage of a simulation is that we can run the simulation several times with different sensor noise while using the equivalent, simulated series of vehicle positions in each run. It is also possible to split the trajectories into different behaviors starting from exactly the same conditions. In real-world scenarios the vehicle will never start from exactly the same position when splitting into different behavior modes.

The intersection scenario seen in Fig. 9.11 evaluates the algorithm with an oncoming vehicle E (yellow). The vehicle E has two behavior alternatives/modes. It can follow the straight lane or turn left. The simulation allows us to drive both alternatives with exactly the same path until the bifurcation point. Both vehicles drive at 30 km/h and no velocity reduction takes place before entering the curve. This makes it impossible for the algorithm to use the absolute velocity as an easy criteria for behavior detection. The modeled sensors of the red ego-vehicle detect the position of the other vehicle with a quite strong sensor noise. This would, in practice, also lead to difficulties using the velocity as criteria since the derivative of a noisy time series is even noisier than the time series itself. As a result, the ICUBHF approach using direction attractors should unfold it's full strength. The reader

should also recall that even the noisy position alone without tracking could be used to estimate the behavior of the observed vehicle, but the high noise level would lead to a very late detection. For example, when the vehicle is sensed in the other lane, that may indicate a turn of the vehicle, but the high sensor noise is a more probable explanation for that sensing. Without Bayesian tracking, non-model-based methods like a moving averaging must be done, but non-model-based methods imply a serious temporal delay for the behavior detection. Such high delays are unwanted in the ADAS domain. With these simulation runs we want to show that our system copes with high sensory noise. False positives and true positives can be prevented with the right parameter set. Therefore, when the a priori behavior assumption was right, the behavior of no single run should flip to the false model assumption during the scene (avoiding false positives). When the a priori behavior assumption was wrong, the behavior of all models should switch to the right model assumption in time (avoiding false negatives).

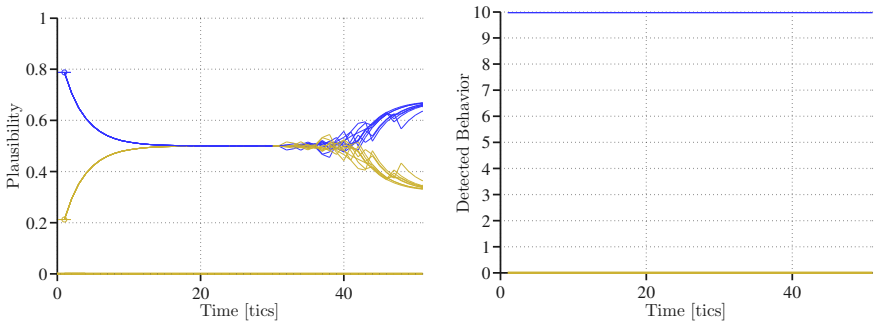
Evaluation Results

The result of 10 runs with the vehicle turning (Mode A is $\hat{b}^*_{true}$) is depicted in Fig. 9.12 and 9.13. The cross validation result of the algorithm when the vehicle drives straight (Mode B is $\hat{b}^*_{true}$) is shown in Fig. 9.14 and 9.15. In Fig. 9.12 the prior plausibility is set to turning. This is the most important setting incorporating risk reflections into the prior. It is a risky situation for the red car if yellow turns to the left unexpectedly, while driving straight ahead is without risk for either. In this setting the belief in the risky turning behavior stays active in all 10 runs because no evidence contradicts the prior assumption (cf. Fig 9.12b). To test the unexpected detection capability, we ran the same scenario with the prior set to driving straight ahead. Thus, sufficient evidence against the prior needs to be accumulated (cf. Fig. 9.13a). All runs detected this change (Fig. 9.13b).

The simulation also enables us to test if the algorithm correctly detects that the vehicle is not turning (non-conform mode as a priori mode). When the prior is set to turning behavior, Fig. 9.14 shows that the algorithm's belief appropriately changes from "turning" to "straight driving". Note that the horizontal position in Fig. 9.14b indicates how far from the yellow vehicle's lane center this occurs (lane width is 3 m).

Figure 9.15 shows the conform prior again. Assuming that the vehicle drives straight from the beginning, the belief should not change since the vehicle is indeed driving straight. But this time a false positive occurs due to the high sensor noise. This false positive vanishes if we change the threshold value from $\theta = .12$ to $\theta = .13$, but in this case the detection occurs slightly later. This shows that the threshold value allows us to fine tune the trade-off between detection delay and detection accuracy or sensitivity.

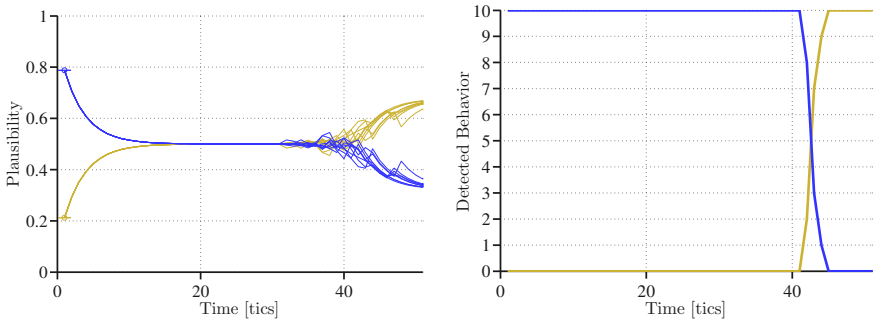
The characteristic of the noise was fixed to a relatively high level for



(a) The mode plausibility graph

(b) The detected behavior graph of 10 different runs.

Figure 9.12: Evaluation of scenario mode A is $\hat{b}^*_{true}$. Conform mode setting: Blue is the prior assumption (Turning mode)



(a) The mode plausibility graph.

(b) The detected behavior graph of 10 different runs.

Figure 9.13: Evaluation of scenario mode A is $\hat{b}^*_{true}$. Non-conform mode setting: Blue is the prior assumption (Driving straight mode).

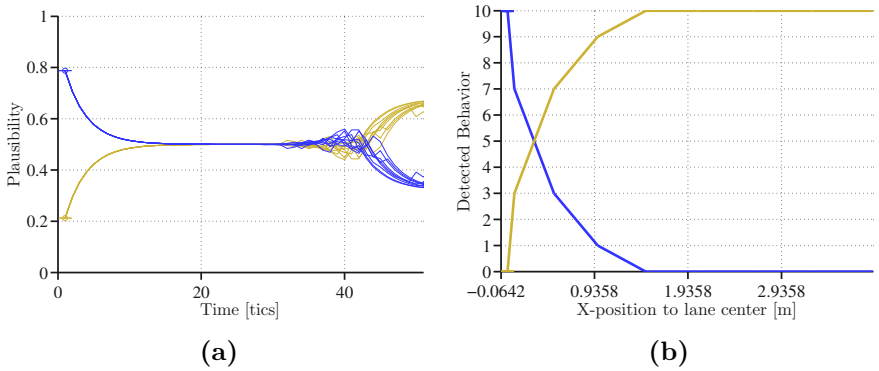


Figure 9.14: Evaluation of scenario mode B. Non-Conform mode setting: Blue is the prior assumption (Turning). (a) The mode plausibility graph. (b) The detected behavior of 10 different runs with respect to the x-position. In order to see at which distance from the lane center the lane change was detected we rescaled the time axis to the x-position (this is the orthogonal dimension to the lane-marking). x-position 0 is the center of the left lane. At x-position 1.5 m the street center line is crossed. Or in other words (b) is a behavior mode detection graph with the x-axis rescaled to the distance from the lane center.

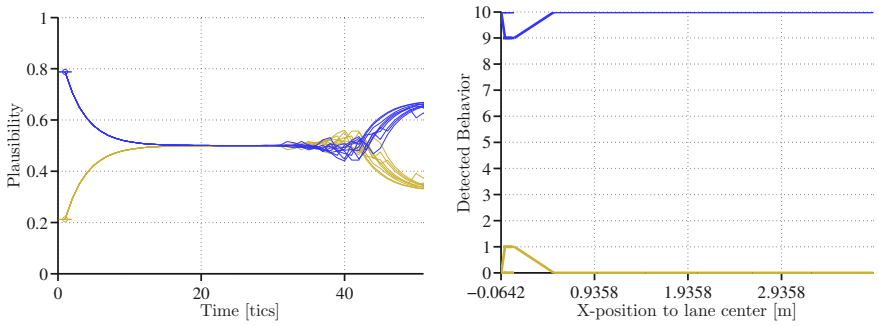


Figure 9.15: Evaluation of scenario mode B. Conform mode setting: Blue is the prior assumption (Driving straight) (a) The mode plausibility graph. (b) The detected behavior of 10 different runs with respect to the x-position. In order to see at which distance from the lane center the lane change was detected we rescaled the time axis to the x-position (this is the orthogonal dimension to the lane-marking). x-position 0 is the center of the left lane. At x-position 1.5 m the street center line is crossed. Or in other words (b) is a behavior mode detection graph with the x-axis rescaled to the distance from the lane center.

all runs (Gaussian white noise with $\sigma_x = 1.8m$ and $\sigma_y = 0.9m$). Higher noise values would lead to a higher false positives rate, lower values to less false positives. These errors can be avoided by adapting the θ value, where a higher theta value leads to a later detection but more accurate behavior detection. Thereby the detection time is also indirectly determined by the sensor noise, via the threshold value.

Further evaluations in intersection scenarios will follow in the real-world evaluation in Part IV.

9.3 Survey of the Ongoing Research

Driving behavior detection and anticipation algorithms is a hot topic in current research. While state-of-the-art approaches focus on the behavior detection of a driver without a reasonable context incorporation, researchers are giving more and more attention to a Bayesian incorporation of context information. Often the approaches are still limited or at least focused on the behavior of the ego-vehicle.

Most research groups are now aware that prediction models need to work in arbitrary situations. The behavior detection algorithms have to work in more than one intersection or in arbitrarily curved roads. Such generic models can be created either by defining goal states utilizing a lane graph (referenced here as attractor-like concepts) [71] or by utilizing machine learning techniques [39]. The development of generic concepts that give implicit context-independent or self-adapting rules for behavior modeling instead of postulating each single rule explicitly is just at its beginning.

Lane graph data was also used in previous approaches, as in [8], but instead of using the lane-level map data to improve the prediction models, the search-space was limited to the lanes, as already mentioned in Section 2.2.4. The probability distributions move on lanes like trains on rails. These methods may be suitable for simulations with deterministic starting points, but not for Bayesian vehicle tracking, which needs to incorporate sensor information on every possible 2D position on the road plane.

However, such lane-constrained approaches may be useful in special applications that differ in their requirements from the position tracking and behavior anticipation task. In [64] a lane-constrained particle filtering approach (cf. Fig. 9.16) was used to improve a vehicles navigation system state estimate. The output space is hence limited to the lane-space and answers the question of whether or not the ego-vehicle left the highway via an exit lane or not. Therefore, the on-board camera image was classified into a small set of states. The classified camera image is then used, together with the GPS position, as an input into the particle filer. The lane-constrained search-space is sufficient for two reasons. First, the output and, somewhat the input, is the vehicle's driving lane and not its 2D position. Second, the

real-time demand is low and therefore vehicle dynamics and local predictions can be neglected. The worst case scenario for a wrong assumption through the neglection of kinematic predictions would be a navigation system giving wrong directions to the driver.

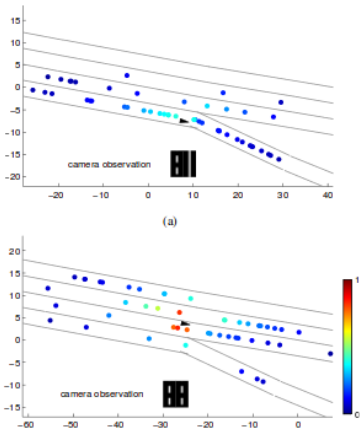


Figure 9.16: The lane-constrained particle-filter. Originally published in [64].

Another lane-constrained ego-vehicle behavior anticipation approach is presented by [50]. The vehicle position in a digital map, the indicator signal, the velocity profile and the driver's gaze direction are used as inputs. The models are not learned, but are generated by user studies. The probability representation is, of course, again too coarse to model spatial vehicle dynamics, but the task limited to the ego-driver behavior detection avoids the need for such indirect observations like the vehicle position. In an ego-vehicle the velocity profile and the drivers gaze direction can be captured without high noise values. A comparative approach using learned models instead is proposed by [60].

In [71] an alternative approach to Bayesian filtering is proposed for maneuver prediction at intersections. The mathematics emerge from a control theory background. The steering and deviation of the acceleration needed by a vehicle to reach a goal state is represented as cost function. The goal state is set on the lane center on a lane graph like the attractor points in our ICUBHF approach. It defines a desirable end state of the vehicle and a trajectory is generated between the goal state and the current state of the vehicle. The vehicle state is represented as a deterministic vector without probabilistic considerations. The necessary costs to reach each goal point from the vehicle state vector are then compared in each time step to derive the most plausible behavior. Unfortunately, this approach seems to lack a way to cope with sensory noise, instead a DT-Filter is used to smooth the behavior outputs, but no real probabilistic framework is used. For evaluation the approach was used for a ego-vehicle behavior detection tasks with

state information provided by GPS and IMU. The non-probabilistic representation makes it likely that the approach will not work sufficiently well with more noisy position data. State information delivered by a low-cost on-board vehicle detection will most likely not work with this approach.

[51] uses a classic IMM approach to detect lane changes of an ego-vehicle. The correct model is chosen by comparing the models by their "innovation vector" (cf. 8.4). IMM is the state-of-the-art approach for predictions with short-term models. Here a map is utilized to create the prediction models. Time will show if that Kalman filter IMM approach will be generic enough to cope with more than two lanes and arbitrary situations. The prediction model for lane-changing and lane-keeping is based on the lane heading and is implemented as a pre-programmed non-linear function. This pre-programmed nature of the approach will make it difficult to cope with arbitrary situations exceeding the straight street lane-change situation in the evaluation scenario. An attractor-based dynamic is more likely to generalize to generic situations and especially when the noise level is high the non-linear dynamics cannot be represented well in the Kalman filter approach.

A more advanced probabilistic representation is given by [39]. The probabilistic framework is used in a simulation setup for stationary intersection surveillance. The used *"map serves only as source of information for the prediction process instead of constraining the motions of the traffic participants"* [39]. Therefore the framework can cope with sensory noise and uncertainties in the prediction model. In contrast to the ICUBHF approach, the authors use an EM-learning algorithm in order to adapt the prediction model instead of utilizing an attractor-like concept. Relations to other vehicles (e.g. distances or angles to other vehicles) are also treated as context information by this algorithm. Due to computational limits this cannot be done with complex vehicle position PDFs. The vehicle state representation needs to be reduced to a single or a few representatives.

The publication dates show that behavior anticipation is a field of ongoing research. The correct representation of the state depends on the task, the sensory input, the output space and the needed output quality. Regardless of which approach is chosen by an engineer, a measure is needed which compares the (estimated) state with the predicted state or goal state in order to determine if a certain behavior model is realistic at the moment. Depending on the terminology this measure can be referred to as costs, innovation vector, quality measure or plausibility measure. But when making the effort to choose a Bayesian framework in order to anticipate behavior, the measure should also be correct from a probabilistic perspective.

Part IV

Real-World Application

Up to this point we have used simulations in order to evaluate the MDBHF and the ICUBHF approaches. Simulations allow us to generate the same trajectory again and again and add a random error vector ϵ to the trajectories. It is also possible to simulate two behavior modes starting from exactly the same trajectory and therefore do a complete evaluation of false positive and false negative behavior detection. None of this is possible with real-world data. The drawbacks of simulations are that they are always abstractions from the real-world. Not all issues and problems will be discovered when using simulations. Therefore, we first started a proof-of-principle to determine under which conditions the ICUBHF is suitable for real-world tasks. Second, we added a row of real-world evaluations with different datasets to further test the application in the real-world. This part of the thesis deals with the, proof-of-concept, the real world evaluations and the revealed issues.

Chapter 10

Real-World Application

In real-world applications we need real-world sensors delivering the data needed for our MDBHF or ICUBHF. Vehicle positions are already available from virtual sensors that detect vehicles in images or videos. Two benchmarks used provide images and vehicle position outputs. All those positions are measured in the reference frame of the ego-vehicle and are therefore provided in relative coordinates to the ego-vehicle.

The datasets also provide data from an Inertial Measurement Unit (IMU) that allows us, together with the ego-velocity from the Controller Area Network (CAN)-data, to compensate the ego-movement. The MDBHF is already operative with all mentioned data. Since we focus on the behavior detection, we also need the context information in order to run an IMM-ICUBHF.

The context that is needed to generate the behavior models by the attractor function can be delivered in at least two ways, either by

- on-board lane detection sensors working on the camera image
- or by self-localization in a global lane-level map

A simple way to obtain access to a map is the Open Street Map (OSM) interface. Before we elaborate further on this point, we want to outline the problems of the on-board lane detection. First, the street is partially occluded by other vehicles. Especially when the vehicle that should be tracked is a vehicle driving in front of us in the same direction, the lane detection-algorithm will usually not see the street in front of the vehicle and is therefore sometimes unable to give a lane output for the street in front of the vehicle. This is a big problem, since this is the most relevant area for generating the behavior of in-front driving vehicles. Since it is difficult to give statements on the street under other vehicles or even behind those vehicles, most street detection algorithms focus on free driving space detection. ADAS systems may use that output in order to derive a space for

performing driving actions or to set up a priori assumptions $P(X)$ for vehicle detection or tracking algorithms. The data is not usable for generating behavior models for tracking and behavior detection algorithms. A good example of free driving space detection is [46] or, for ego-lane detection [47].

The detection of lane markings can also be disturbed in another way. In addition to occlusion by other vehicles or foliage, detection algorithms working on camera data are always prone to errors from occlusion by other objects, by rain and the thereby produced reflections, fog, darkness or sun glare. Despite those drawbacks, the online lane-detection has an outstanding advantage: When the traffic patterns or lanes are changed permanently or temporarily (as by construction sides), the online detection is theoretically able to detect the lanes anyway. However, in practice a priori assumptions of current lane detections algorithms are often violated in construction side situations and therefore the performance is weak. But an algorithm using map-data in order to receive the context-information does not perform well either. The database needs to be updated in such cases by a data-provider or administration. There are first indications that Google could become an innovator or even a map provider for ADAS or autonomous driving. Their current autonomous car project seems to rely excessively on map data that are collected by Google itself. [69]

Since we do not have access to a commercial lane-level database we have taken the OSM into account. In Sec. 10.1 we present a strategy as to how Open-Street-Map data can be used together with accurate ego-vehicle position data, like that provided by the KITTI-benchmark, in order to improve tracking. Driven by the results, we decided that OSM data is too inaccurate for that task and we used manually annotated lane data in the subsequent chapters in order to estimate the behavior mode of the observed vehicle. In Sec. 10.2 preexisting HRI-EU data was used in order to estimate the behavior in a vehicle following task. The recordings show an ego-vehicle that follows the observed vehicle into different intersections. In Sec. 10.3 we evaluated the same recordings with the reachability scaling introduced in Sec. 9.1.4. In a final evaluation (Sec. 10.4.3) we detected the behavior in an arranged intersection setting. In contrast to the previous evaluation we performed similar recordings with different behavior modes on the same intersection.

10.1 KITTI and Open-Street-Map

In this section we give a short introduction to the KITTI benchmark and to context retrieval through OSM.

10.1.1 The KITTI-Benchmark

After the evaluation phase of our project, KITTI also provided a benchmark for tracking. This dataset was not yet available for the evaluations, so we therefore used the raw data originally provided on the KITTI homepage [36].

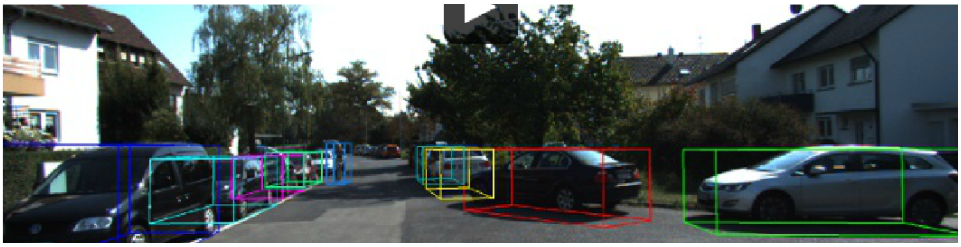


Figure 10.1: Annotated Objects in the KITTI benchmark. Copied from KITTI website [36].

In order to use the data provided by the KITTI benchmark coordinate transformations together with the map data, different reference systems were necessary. The details are omitted here, but Fig. 10.2 shows a general overview of the different coordinate systems. More information can be found on the website [36] and in [37], as well as in the readme-files included in the raw data. Altogether, the KITTI benchmark is a well-designed dataset easy accessible via the website and the concrete individual benchmarks are convenient to use.

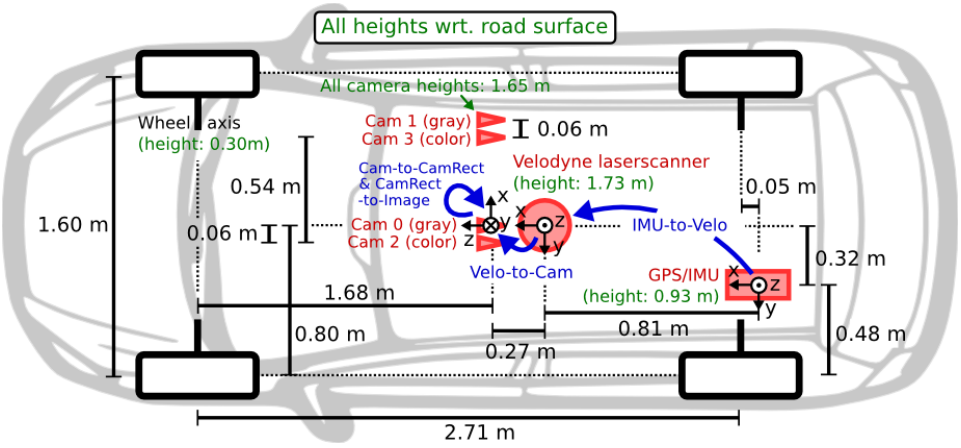


Figure 10.2: Reference systems used in the KITTI raw dataset. Copied from KITTI website [36].

The main advantage for our work was, that the vehicle positions were hand-annotated and therefore very exact. Furthermore the ego-coordinates and the ego-heading provided by the dataset are very accurate since the

GPS is used together with an IMU.

10.1.2 Using OSM as a Map Provider

OSM is an openly editable and freely accessible map database. The streets and lanes are provided by volunteers. New entries can be contributed with the assistance of simple home-use GPS-systems and freely available software, or mobile phones with GPS modules and free phone applications.

The quality of OSM data is improving steadily but depends highly on area and especially on the number of volunteers active in that area. In the areas of Karlsruhe (KITTI dataset) and Offenbach (HRI-EU dataset) street maps are comprehensive and quite accurate.

The streets are represented as polylines in OSM and therefore match the needed context representation of our ICUBHF. In our test application we use the vehicle coordinates to do an OSM query to the so called Overpass API [68]. The Overpass API gives us the street(s) around the given coordinate. A manual selection of the right street polyline was done. Such a manual selection could be avoided by developing an own API that delivers street graphs using more specific criteria than the coordinate alone (e.g. additionally velocities and directions). The overall application using OSM data and the vehicle sensors as input for the ICUBHF is depicted in Fig. 10.3

In our test application we tried several tests with the combination of OSM and KITTI data. From KITTI we used the world coordinates provided by the GPS+IMU, the velocity provided by the CAN-data and the yaw rate provided by the IMU. We used the context from KITTI and we assumed the street polyline from KITTI to be the center of the street and added lanes on both sides of the street center.

This test application shows that there were several problems with this procedure.

- The map data was too inaccurate.
- The assumption of how to generate lanes from a street center polyline was too simple.

In sum, this means that the OSM polyline does not lie exactly on the true street coordinates, and even if it did, the polyline does not lie in the *center* of the street in true coordinates (cf. Fig. 10.4). Therefore, it cannot be used to improve the tracking results. Fusing wrong information into the filter is worse than fusing in inaccurate but statistically right information. This position error is not based on a fix offset. The error is a combination of two factors: First, the positioning error of the GPS device of the map recording vehicle and second, the fact that the recording vehicle device is not positioned in the street center. The last issue leads to problems especially when OSM data is to be used in intersection scenarios. Different exits off

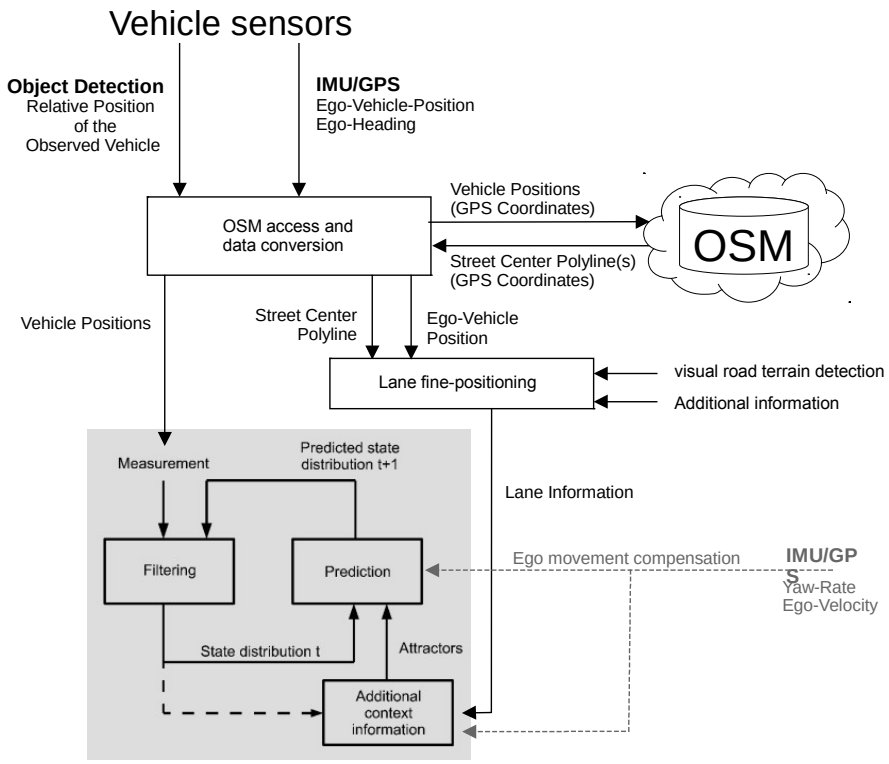


Figure 10.3: The overall system. In the lower part the ICUBHF is depicted in an abstract Bayesian filter representation. The remaining part shows the pathway of the information from the object detection algorithm and the IMU/GPS over the OSM interface to the ICUBHF.

the intersection were recorded by the same vehicle or different vehicles that were driving on different lanes or in different directions. The OSM data is therefore suited for navigational purposes but not for being used as a replacement for lane-level street graphs. The second problem is correlated to the implementation we used in that first test application. Traffic islands are not considered when adding lanes directly to a street center polyline. In comparison to the first issue, this problem could be easily solved.

Does that mean that street-level graphs like the OSM data are generally not qualified to improve tracking or detect behavior? It is possible that OSM data could be used together with an on-board lane detection algorithm to mutually improve the outputs of the different systems. Creating such a fine-positioning system is another complex task and lies beyond the scope of this work. We therefore manually created lane-level graphs by annotating Google earth data.

10.1.3 Lessons from using OSM with KITTI Data

One intention behind the ICUBHF was to improve the tracking by inferring context information (cf. Sec. 4). This was not possible in our setting due to the inaccurate map data. Additionally the intention was to use the ICUBHF with data provided from HRI-EU. The HRI-EU dataset does not use a GPS localization supported by an IMU, and therefore the ego position in the real world is much more inaccurate than in the KITTI dataset. Even when using the annotated lane graph, the positioning within this accurate map data is a problem. We therefore abandoned our goal to improve tracking and focused on behavior detection. The task to improve tracking, even with annotated data is only possible with an accurate positioning, e.g. by using on-board lane detection or by using LIDARs for positioning. However, today's LIDARs are expensive and with having a LIDAR on board, the engineer will use the LIDAR as an accurate sensor for the vehicle tracking itself in order to accomplish two things simultaneously. Therefore, the use of a LIDAR for ego-localization is a hypothetical option in the camera vehicle tracking application.

10.2 HRI-EU Real-World-Test scenarios

The following selected real world scenes are from the HRI-EU datasets. We selected scenes in which an ego vehicle approaches different intersections and monitor the behavior of other vehicles that approach the intersection. A vehicle O driving in front of the ego vehicle E is observed in order to detect by which exit of the intersection it is likely to leave. The behavior modes were set manually. Each drive to an intersection exit is an own mode. We filtered out modes whose a priori likelihood is very low. For example we assume that vehicles will not enter a one-way street into the wrong direction. The models

Figure 10.4: The Google Earth view is plotted against the OSM data at a test intersection. The figure on the bottom shows the OSM overlay. The figure on top shows a plain view of the intersection. In the south intersection exit the left lane is the OSM street center. In the road from west to east the northern lane is represented by OSM. The wide intersection area is not properly represented in the OSM data. (Map data by Google Images/GeoBasisDE/BKG and AeroWest)

for each mode are created by annotated lane level graphs. We researched intersections with wide intersection areas, meaning that vehicles can follow a wide variety of trajectories in order to achieve their overall behavior goal. An algorithm working in such intersections has to be much more generic and will, with a high likelihood also work in simple intersections.

Since the ego-positioning was not very accurate and deviates up to more than a street width, we manually initialized the position and vehicle heading ω_0^{Ego} of the ego-vehicle E at time step 0 in the global coordinate to the estimated position, thereby simulating an ego-positioning system that works well. The further positions were determined by a dead-reckoning, using the velocity from the CAN-data and the yaw rates from the vehicle IMU. Errors in the vehicle heading are therefore especially critical since the error in the future positions rises with the distance to the set initial position $\mathbf{l}_0^{Ego} \approx \mathbf{l}_{real_0}^{Ego}$.

10.2.1 Real-World Intersection Scenario 1

Parts of this subsection have been published in [7]. We used the same parameter set gained from the Carmaker-simulated intersection evaluation in Sec. 9.2.3 in order to evaluate the AMM-ICUBHF in real-world intersection scenarios.

The scenario intersection with the annotated polylines is shown in a Google Earth image (Fig. 10.5). Additionally the on-board camera view (Fig. 10.6) of Scenario 1 is shown. The observed vehicle and our ego-vehicle are leaving the road heading north via an exclusive left-turning lane towards the east. We model two behavior modes: turning left towards one of the two destination lanes and driving straight on the left-most straight-driving lane. The other lanes are omitted since their prior would be near zero as a result of the fact that the vehicle was detected in the left-most lane. Additionally the left turning mode does not differentiate between the left exit lane and the right exit lane, as positioning uncertainties are too high to make such a distinction. Setting the prior belief to the "turning left" mode (Fig. 10.7a) does not lead to any change in the behavior belief. In Fig. 10.7b the "driving straight" mode as prior belief was quickly deemed incorrect. In time step 17 the new behavior mode was detected. Note that a mode behavior detection graph makes no sense in real-world applications since only one run changes the assumed mode. In mode 2 the mode probability graph does not assign the same probability to both modes. This is because the straight-driving model is not locally identical with the turning model, since the straight-driving model is assigned to the left straight-driving lane, while the turning model is assigned to the turning lane. Because of the separated lanes, the behavior mode is detected early.

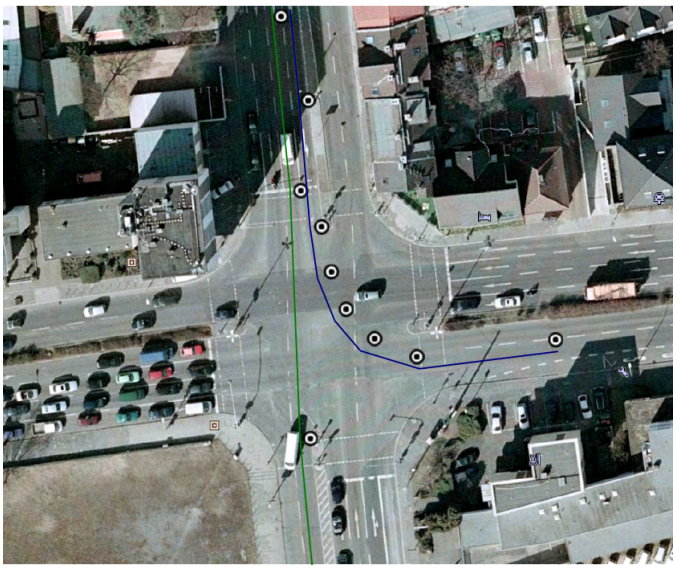
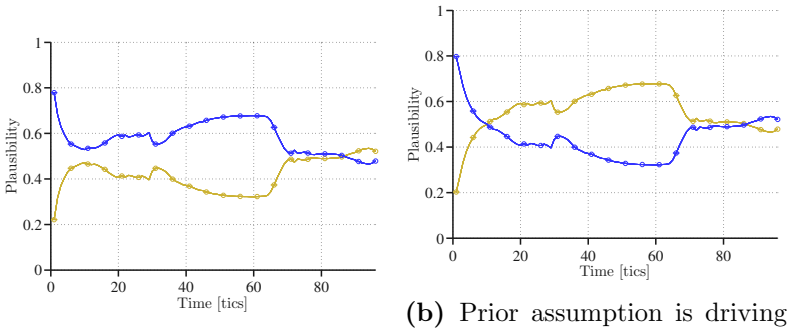


Figure 10.5: In scenario 1 the ego vehicle is coming from the north and turning east. There is one exclusive left turning lane and two possible target lanes in the road Spessartring. The annotated lane center polylines for the two behavior modes are illustrated as overlay. (Map data by Google Images/GeoBasis-DE/BKG and AeroWest)

Figure 10.6: Intersection scenario 1 camera output



(a) Prior assumption is doing a turn. Turning was assumed the whole time.

(b) Prior assumption is driving straight. Turning was detected at time step 17 through the end of the scenario.

Figure 10.7: Scenario 1. Mode probability graphs. Blue is the prior assumption.

10.2.2 Real-World Intersection Scenario 2

Scenario 2 is a rather unusually skewed intersection (cf. Fig. 10.8 and 10.9). The observed vehicle and the tracking vehicle are approaching from the south and turning left. This time all modes are starting from one lane. The road from the north is a one-way road, so that either a left-turning behavior or right-turning behavior can be expected. The evaluation (Fig. 10.10) shows that the detection works well. In comparison with Scenario 1 however, the plausibility graph is not as clear-cut. The main reason is that the left turning is modeled by an insufficiently wide turn. The real behavior mode of the vehicle in front of the ego vehicle deviates from that. Nevertheless, the threshold value ($\theta = .12$) ensures that "left turning" was detected successfully. The graph shows a strong increase of the right mode during time steps 30 and 45, however, this effect only occurs because an attractor candidate which is placed in the intersection fits very well to the left turning behavior. The next attractor candidate on the polyline again does not fit as well to the turning behavior so that the mode probabilities intersect again before the actual effect occurs. Therefore, the early detection is based more on coincidence than on real dependencies. We reevaluated the recording by defining a *free intersection area* in the next sub-section. The free-intersection area avoids giving a predefined trajectory within the intersection area.

Figure 10.8: Intersection scenario 2 shows an intersection scene with lanes for west-east traffic only. The ego vehicle follows the observed vehicle which approaches from the south and turns left. Since the northern road is a one way road the vehicle can turn left or right. (Map data by Google Images/GeoBasis-DE/BKG and AeroWest)

Figure 10.9: Intersection scenario 2 camera output

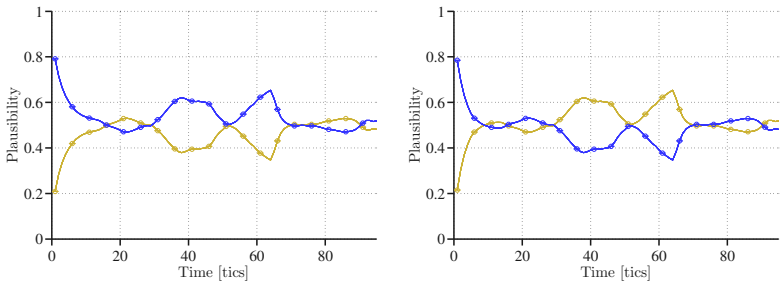


Figure 10.10: Scenario 2. The mode probability graph. Blue is the prior assumption. (a) Prior assumption is a left turn. (b) Prior assumption is a right turn. In (a) left turning was assumed the whole time. In (b) from time step 34 to the end it was detected that a right turn is not the real behavior. The merging of the mode probability in phase 4 (cf. 9.2.1) is highly pronounced.

10.2.3 Real-World-Test Scenario with Free Intersection Area

The evaluation of the last intersection has shown that the "turning left" mode is not modeled very well by the chosen attractor approach based on the polyline. We therefore evaluated the concept of free intersection areas at this intersection. The idea is that the vehicle trajectory for the left-turning mode can vary between a wide range of trajectories from wide to short routes to the intersection exit.

The free intersection area is defined as a polygonal *Area* (cf. Fig. 10.11) marking the intersection area. Within this polygon no attractor candidates are selected. The attractor algorithm (Alg. 7.1) can be easily augmented by adding the additional constraint "$\mathbf{a}_j$! *within Area*" in line 4. Additionally the lane borders are ignored within the free-area polygon. Nevertheless it has to be assured that an attractor candidate is available at the intersection exit outside of the free intersection area.

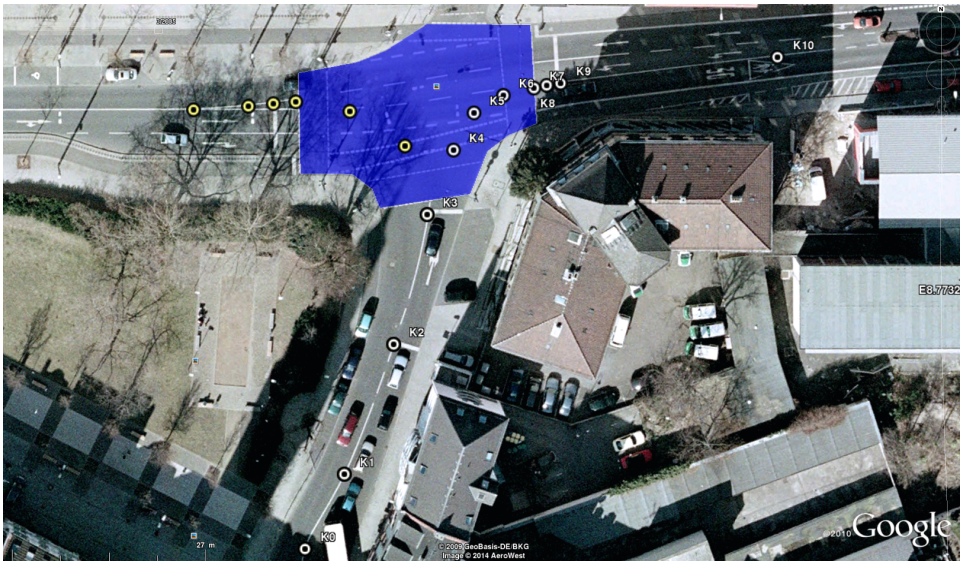


Figure 10.11: The image shows the free area polygon and the left boundary limit of the lanes used in the attractor algorithm. The lane width was estimated to be 3 m. (Map data by Google Images/GeoBasis-DE/BKG and AeroWest)

The concept of the free intersection areas is particularly interesting for wide intersections without lane markings within the intersection, since the variety of possible trajectories is highest there.

We applied the free area polygon to the "left-driving" mode in scenario 2 as depicted in Fig. 10.11. The right-driving mode, is not being changed, since the right-turning lane is guided within the intersection by the sidewalk. It does not make sense to apply the free area polygon to such modes. The result is depicted in the mode probability graph (Fig. 10.12).

When comparing the graph with Fig. 10.10 certain differences can be

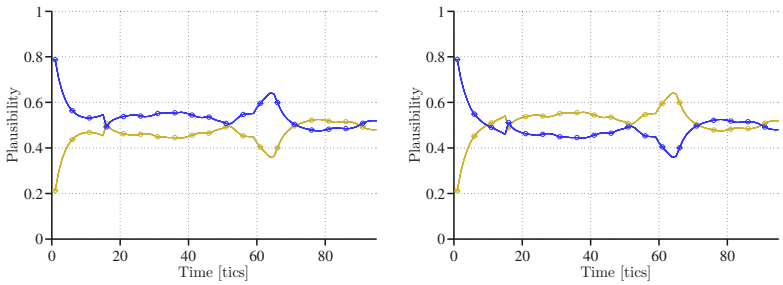


Figure 10.12: Scenario 2 with free area polygon for mode "left-driving". The mode probability graph. Blue is the prior assumption. (a) Prior assumption is doing a left turn. (b) Prior assumption is a right turn. In (a) left turning was assumed the whole time. In (b) from time step 60 to the end it was detected that a right turn is not the real behavior.

seen. First, note that in phase 1 the modes do not have the same likelihood. The reason is that due to the free area polygon, the attractors already affect the area in front of the intersection, independent from the fact that the polylines in front of the intersection are identical. By chance the sensed position of the vehicle lies more in the direction of the "left-driving"-mode assumption. Due to the hysteresis, the difference is not enough for the behavior detector to change behavior. Phases 3 and 4 show the typical progression. In time step 60 the behavior change is detected in the non-conform a priori assumption.

In the result this shows that in wide intersections the free area concept helps to avoid a wrong detection by allowing the correct model to match a high number of trajectories. The concept cares successfully of the fact that trajectories do not need drive over the individual attractor point. Only since a single attractor point matches better to the wrong behavior mode, the results in the evaluation graphs (Fig. 10.12) look worse in comparison to the results from the same scenario without using a free intersection area (Fig. 10.7). In other words, the detection in time step 60, and not the detection in time step 17, is the wanted result.

10.3 Real-World-Test Scenario with Plausibility Scaling and Reachability Scaling

In Sec. 9.1.4 we introduced the reachability scaling in order to avoid phase 4 which could lead to difficulties when framework-algorithms do not sort out non-reachable hypothesis themselves. Figure 10.13 shows the mode plausibility with reachability scaling $P(\hat{B}_t^{Reach}|\mathbf{Y}_{1:t+1})$.

In order to avoid a false positive straight-driving detection the hysteresis

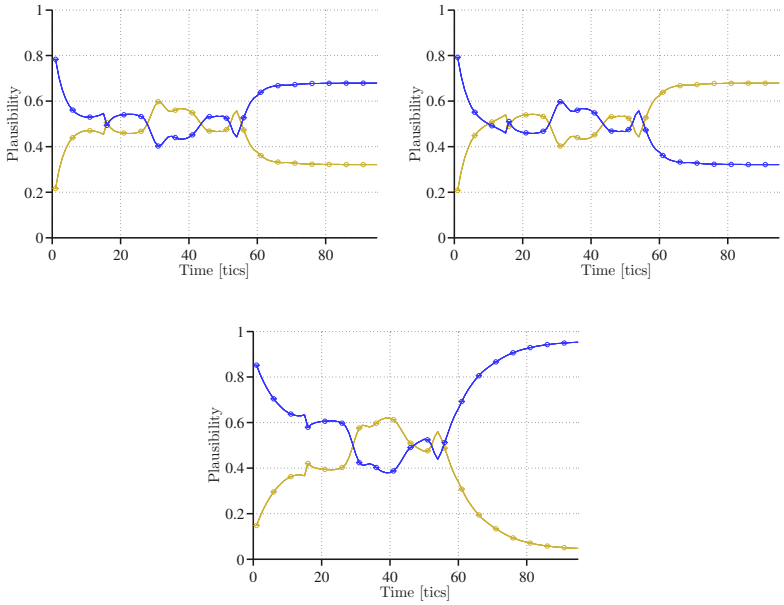


Figure 10.13: Scenario 2. The mode reachability scaled probability graph. Blue is the prior assumption. (a) Prior assumption is doing a left turn. (b) Prior assumption is a right turn. In (a) left turning was assumed the whole time. In (b) from time step 60 to the end it was detected that a right turn is not the real behavior. In (c) the $P(\hat{B}_t|B_t, \hat{B}_{t-1})$ (cf. Sec. 9.1.3) was set from 0.9 to 0.995 in order to show that the probabilities will approach 1 and 0 when this PDF becomes more deterministic.

θ needs to be adapted to the new mode probability scaling.

10.4 Real-World Oncoming Intersection Scenario

In the previous real-world evaluations it was impossible to check the evaluation for false positive detection, since in each situation only one behavior mode was executed. Therefore we have recorded multiple trajectories with different behavior modes in a bend priority road intersection scene with an oncoming vehicle. Such a scene was already evaluated in the simulation (cf. Sec. 9.2.3). Note that the simulation is better qualified to proof the theoretical functionality, even when more than one trajectory is used in the real-world situation. Other than in the simulation, it is impossible in real world to generate trajectories with exactly the same history. Nevertheless, the real-world evaluation will give us valuable experience as to if and how the concept can be applied to a real-world intersection scene.

In this intersection scene it is important to know for an ADAS if the oncoming right-of-way vehicle makes a turn or is driving straight. In the first case the ADAS has to brake when the ego-driver is not reacting himself.

10.4.1 Challenges of the Dataset

The real-world data has to be converted and adapted before using it in our ICUBHF. All data needs to be in sync - the vehicle detections, the IMU data and the CAN-data. As already stated, the delivered GPS position is too inaccurate, and therefore the vehicle positions were calibrated before starting each evaluation run. Specifically, the ego-position and yaw in the beginning was set by the calibration. The further ego-positions and velocities were calculated by a death-reckoning. The vehicle detection algorithm delivers several static vehicles, other moving vehicles and non-vehicle objects. The ID of the relevant vehicle was selected manually at the beginning of the evaluation. The detection algorithm has at least two problems. The first is that true distance to the observed vehicle is systematically wrongly estimated and the second is that it has a jumping behavior. The first issue was solved by doing a calibration of the detected position in the forward direction $\mathbf{x}_1$. A time-linear additive correction value was added to the detected $\mathbf{x}_1$ position. The correction was necessary since such systematic errors cannot be handled by Bayesian filters. In practice another algorithm is needed that calibrates the detection algorithm.

Secs. 10.4.1 and 10.4.2 deal with the second issue. Unfortunately, the jumps are not systematic enough to use the proposed counter measures. The ICUBHF without those measures is used instead in the main evaluation in Sec. 10.4.3. The threshold parameter θ was increased to cope with the high additional uncertainties.

Object Detection Algorithm

The object detection algorithm is based on a disparity map that is created by the on-board stereo camera. The object depth in the camera image or the distance of the object can be calculated with the help of the disparity, but first the object is detected within the disparity map. The detected object is considered as a region of interest (ROI) and mapped from the two-dimensional disparity map into the three-dimensional world coordinates. Both, the detection algorithm as well as the the mapping from the ROI to the world coordinates can be adjusted by several parameters, not detailed here.

Vehicles that are visible from the side produce a greater ROI, and it is more difficult to map that huge ROI to an exact position. Therefore, lateral movements produce higher uncertainty in the position that the virtual sensor delivers. Besides that, the mapping is not very exact in far detection distances. Additionally and most importantly, the ROI's size does not have a continuous range, but increases only in fix steps. When an observed vehicle is approaching our vehicle, the sensed position jumps up to 4 meters in a time step (0.1s) when the size of the ROI increases. After the jump in the sensed position, the sensed position keeps nearly constant or even moves back slightly.

There are three possibilities of how to deal with such sensor characteristics.

1. Use a moving average on the sensor information.
2. Adapt the sensor model.
3. Use adaptive simulation time-step sizes.

The first item has the advantage that it is easy to implement. The drawback is that the moving average has, as a low-pass filter, the property of delaying the estimate. Additionally, the sensor information is no longer independent over the time, a basic assumption of the Bayesian filter. The Markov-assumption can no longer be used on the sensor model. Therefore, this possibility should be discarded.

The adaption of the sensor model seems to be a good solution within the Bayesian filtering theory. The drawback is that the sensor noise shows a complex undocumented behavior, depending on the vehicle position and the vehicle direction. The task of modeling the sensor noise would be rather complex since the jumps in the disparity algorithm need to be modeled in the sensor model. We therefore decided to aim for the last item.

The simulation filter time steps size ΔT is adapted. The measurement is only fed into the system after a detection jump, meaning that the size of the ROI has just increased. Therefore, the sensor input is analyzed if a jump occurs, while if no jump occurs the prediction time step size is increased by

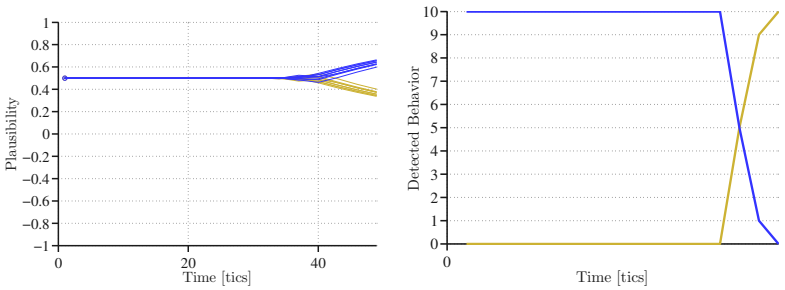


Figure 10.14: $\Delta T_{Sum} = 3 \cdot \Delta T$. (a) Mode probability graph showing the 10 individual runs. Blue is the prior assumption. The prior assumption is the turning mode. (b) The behavior mode detection graph for non-conform setting. All runs detect the change. In the conform setting no false positive detection occurs.

another ΔT_{Sum}+=ΔT until a new jump is detected. Once the new jump is detected the prediction is done with the ΔT_{Sum} as filter time step size. Note that this is in contrast to the pure prediction loop (cf. Sec. 4.3). We admit that in a real-world situation the ICUBHF would not be fast enough to do the prediction instantaneously and give the result in time for the new sensor input to be incorporated. However, in practice different predictions with different ΔT_{Sum} could run in parallel and then a multiplexer like approach would be used to select the right ΔT_{Sum}.

10.4.2 Pre-Evaluations: Adaptive Update Time-steps

In order to evaluate the ICUBHF with adaptive update time-steps a pre-evaluation within a simulated environment was done. We have again been using the Carmaker T-Crossing situation. Figs. 10.14 and 10.15 show the evaluation for a fix time-step $\Delta T_{Sum} = 3 \cdot \Delta T$. In Figs. 10.16 and 10.17 we show the same evaluation for a random ΔT_{Sum} which skips a time step with a chance $p_{skip} = \frac{2}{3}$.

Wrong detections, or in other words wrong categorizations of the modes appear when the model does not fit to the mode. But what was not modeled correctly? It is necessary to take a deeper look into the ICUBHF estimate to discover what was incorrectly modeled. Figure 10.18 shows the estimate of the ICUBHF running with the different behavior models.

It can be seen that a long δT_{sum} (identifiable on the long rising line segment) appears just before the intersection. Due to the long δT_{sum} the d^* in Eq. 7.9 becomes the maximum possible value 1. That means the new direction in the certain grid cell is not set by a location on the spline itself, but by the end point of the spline $\mathbf{l}_i^A$. The new direction $\omega^*(\mathbf{v}_i)$ (cf. Eq.7.10) is therefore heading directly to the attractor that already lies in the intersection. More abstractly, the dynamics modeled by the attractors

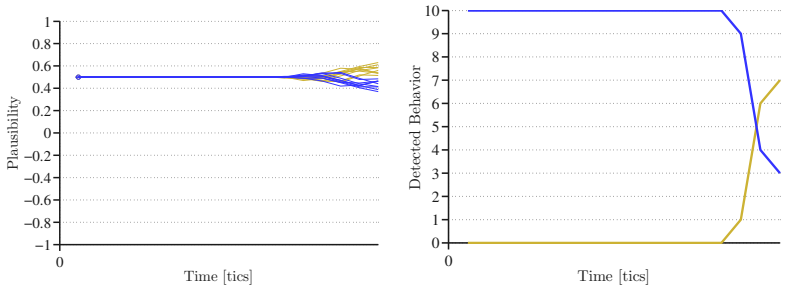


Figure 10.15: $\Delta T_{Sum} = 3 \cdot \Delta T$. (a) Mode probability graph showing the 10 individual runs. Blue is the prior assumption. The prior assumption is the straight driving mode. (b) The behavior mode detection graph for non-conform setting. Some runs are late in the change detection. In the conform setting no false positive detection occurs.

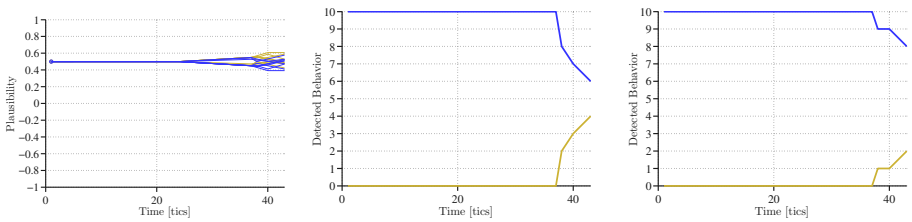


Figure 10.16: Random ΔT_{Sum}. (a) Mode probability graph showing the 10 individual runs. Blue is the prior assumption. The prior assumption is the turning mode. (b) The behavior mode detection graph for non-conform setting. False negatives or late detections occur. (c) The behavior mode detection graph for conform setting. False positives occur.

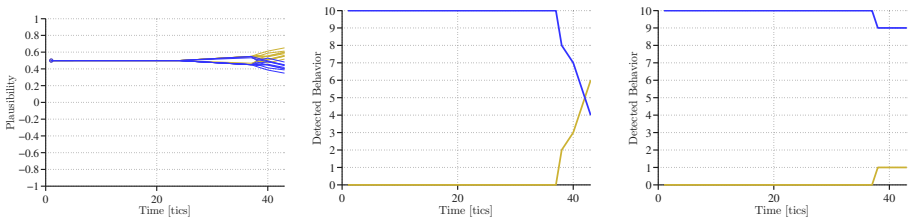
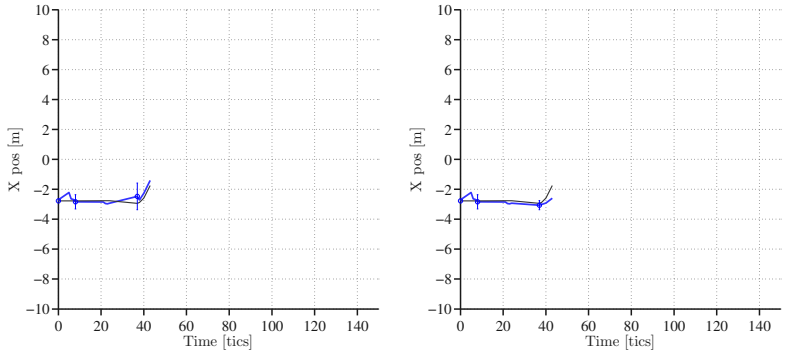


Figure 10.17: Random ΔT_{Sum}. (a) Mode probability graph showing the 10 individual runs. Blue is the prior assumption. The prior assumption is the straight driving mode. (b) The behavior mode detection graph for non-conform setting. False negatives or late detections occur. (c) The behavior mode detection graph for conform setting. One false positive is seen.



(a) Using turning model. **(b)** Using straight driving model.

Figure 10.18: Random ΔT_{Sum}. The estimate position (blue line) compared with the ground truth position (black line). It is seen that in the conform mode (=turning mode) the estimate anticipates the turning too early in some of the runs.

are only valid locally and long predictions over several ΔT may therefore fail in modeling the mode. We admit that this would not occur with a pure-prediction loop.

For small multiples of ΔT the predictions are local enough, but for a long ΔT_{sum} this can cause problems. With this in mind we can cautiously use the adaptive filter step size in the real world situation.

10.4.3 Oncoming Vehicle in a Real-World Situation.

In this section we evaluate the performance of the ICUBHF in the oncoming ICUBHF in the intersection shown in Fig. 10.19. In order to get an accurate positioning of the ego-vehicle it is necessary to calibrate the ego-position in the calibration area. In practice, more precise ego-localization sensors or algorithms will avoid such a calibration.

The vehicle detection algorithm delivers very noisy position data. The trajectory of the sensor detections of all three real-world runs is depicted in Fig.10.20. The evaluation was executed without time adaptive-filtering, since the jumps in the detection trajectory were not systematic enough to state a time-step selection algorithm.

Scenario Description

The scenario setting consists of two vehicles. The ego-vehicle E approaches the intersection from the give way road with about 30 km/h, meaning from the east in Fig. 10.19a or from the street in the right side of the figure. In order to avoid a collision it was necessary to brake abruptly in front of the intersection. The ego-vehicle starts from the calibration area with velocity 0 and quickly speeds up to 30 km/h. The observed vehicle O starts from the

(a) The intersection area from a pedestrian perspective.

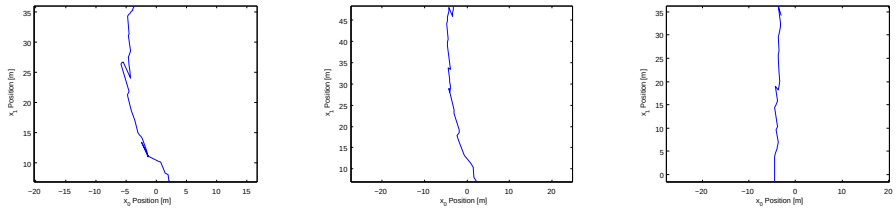
(b) The ego-vehicle in the calibration area.

(c) The intersection area from a bird's-eye view. The street center polyline is shown as overlay. The calibration position of the ego-vehicle is shown in red. (Map data by Google Images/GeoBasis-DE/BKG and AeroWest)

Figure 10.19: The real-world evaluation scenario from different views.

west and approaches the intersection at about 40 km/h. The first detection of the observed vehicle O occurs at about a distance of 50m. The observed vehicle then either drives straight east or turns left and follows the priority road towards the north. The first stated mode can lead to an accident when the ego-vehicle does not brake. The second mode is non-threatening. It is therefore useful for the ADAS to know if the vehicle will turn or drive straight.

In the next paragraphs we will show the results of the runs following turning behavior mode. Thereafter the straight driving run is presented.



(a) First run **(b)** Second run **(c)** Third run

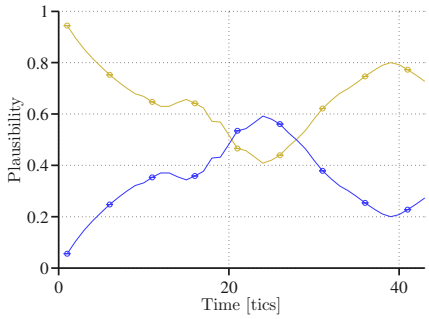
Figure 10.20: The calibrated vehicle detections $\mathbf{y}_{1:t}$ relative to the camera of the ego-vehicle. The first detection of the vehicle is seen at the top of each image. The last detection in time step t is seen at the bottom.

Turning mode

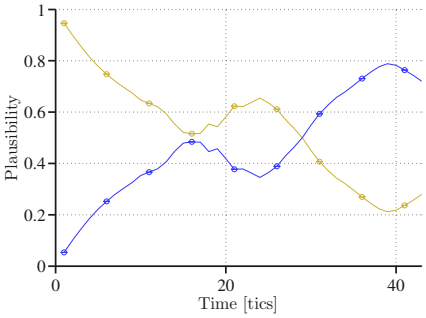
The first run is the noisiest. Figure 10.20c shows a number of reversions in the trajectory. In order to cope with these high uncertainties, the threshold variable for the behavior mode detection was increased to $\theta = 0.25$ for all real-world runs of this evaluation. It can be seen in the mode probability graphs (Fig. 10.21a and 10.21b) that $\theta = 0.2$ would be sufficient, since in the conform setting (Fig. 10.21a) the false mode never exceeds a probability of 0.6. Nevertheless it is rather unfair to adapt the threshold value to the minimum possible value in order to get an earlier detection. Wrong behavior detections would be likely in further runs with trajectories other than the evaluated ones. The remaining 0.05 to 0.25 acts as a security margin.

With that security margin, the behavior change that was detected in time step 32 is annotated in the sensor input trajectory in Fig. 10.21d. The camera input in time step 32 with the ROI of the detection is depicted in Fig. 10.21c. For the human observer, this detection time seems late. However, the detection time is plausible when looking at the underlying conditions. This is discussed in Sec. 10.4.3.

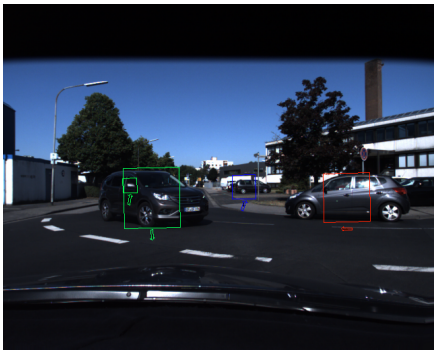
We did a second run with the observed vehicle in the turning mode. Figure 10.22 shows the results. It can be clearly seen that the sensor input trajectory (Fig. 10.22d) looks better than the trajectory in the previous run. Nevertheless the trajectory indicates a backwards driving of the observed vehicle during the early detections. Note that the graph shows the relative position of the observed vehicle with respect to the ego-vehicle. Since the ego-vehicle is itself moving towards the observed vehicle, a constant distance in the graph indicates a backward driving of the observed vehicle. The evidence for the turning mode starts gathering early in that run (cf. Fig. 10.22b). Due to the high threshold value, the behavior change was detected in time step 28, but it is clearly visible that an earlier detection (even before time step 20) is possible with lower θ values. The high gap between the two mode probabilities in the conform setting (Fig. 10.22a)



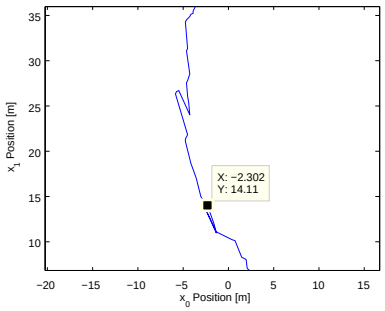
(a) Prior assumption is doing a turn. Turning was assumed the whole time.



(b) Prior assumption is driving straight. Turning was detected in time step 32.

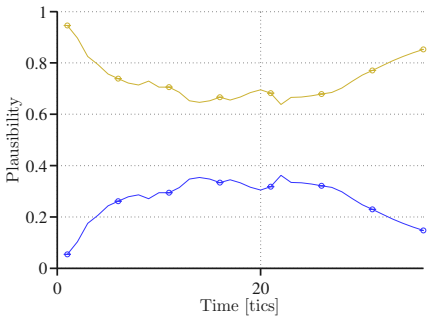


(c) The camera image with ROI of the detection in time step 32.

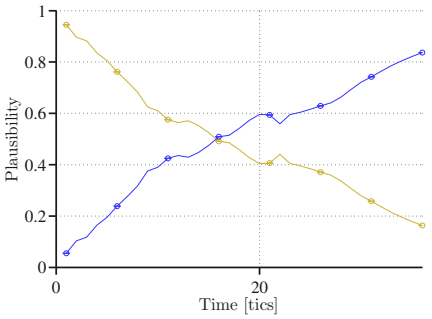


(d) The relative trajectory with marked behavior detection position.

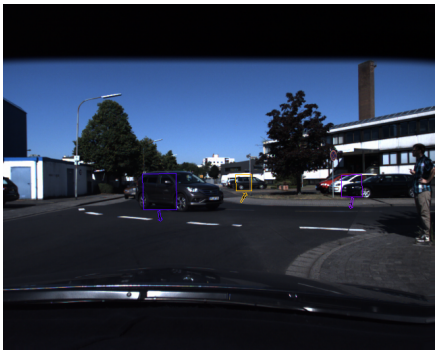
Figure 10.21: Run 1 turning mode. Mode probability graphs. Blue is the prior assumption.



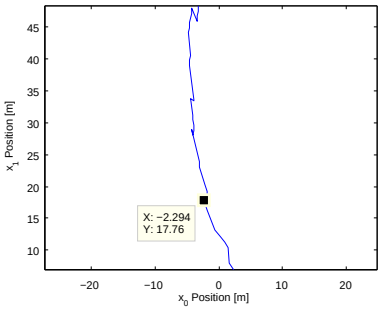
(a) Prior assumption is doing a turn. Turning was assumed the whole time.



(b) Prior assumption is driving straight. Turning was detected in time step 28.



(c) The camera image with ROI of the detection in time step 28.



(d) The relative trajectory with marked behavior detection position.

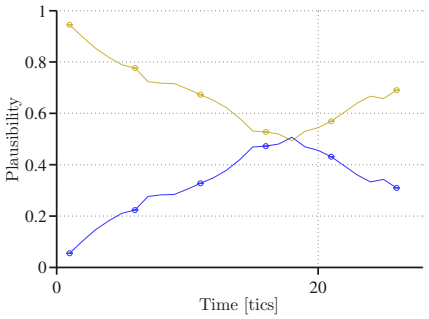
Figure 10.22: Run 2 turning mode. Mode probability graphs. Blue is the prior assumption.

shows that there is never a risk for false positive detections. That means this run would have been detected correctly for the whole time even with $\theta = 0$. However, this is a hypothetical consideration, since in other runs the trajectory and the noise need higher trajectories.

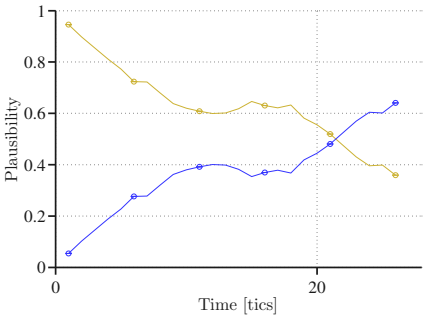
Straight driving mode

The straight driving trajectory was extrapolated four further data points (cf. bottom Fig. 10.23d), since the detection algorithm no longer detects the observed vehicle when the vehicle-front is leaving the camera image. Otherwise, the detection of vehicle O is lost before enough evidence for straight driving is accumulated.

When assuming turning as the correct mode, the straight driving was detected in time step 25. The behavior mode probability is seen in Fig. 10.23b. There is also no risk of a false positive detection for a turning behavior here (cf. Fig. 10.23a). This is the first time step without a vehicle detection by

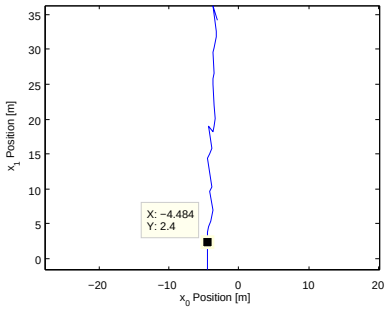


(a) Prior assumption is straight driving. Straight driving was assumed the whole time.



(b) Prior assumption is turning. Straight driving was detected in time step 25.

(c) The camera image with ROI of the detection in time step 25.



(d) The relative trajectory with marked behavior detection position.

Figure 10.23: Run 3 straight driving mode. Mode probability graphs. Green is the prior assumption.

the original vehicle detection algorithm. The detection occurs about when the ground-truth vehicle center crosses the white intersection margin (cf. Fig. 10.23c).

Reflection on the Behavior Detection Performance

Note that the camera pictures themselves suggest that the detection is very late. The reason for this impression is that the human observer looking at the image has better vehicle detection without much noise and extracts further features from the image (cf. Fig. 10.24a). The human driver would extract the vehicle position relative to the lane position directly from the optical image without the need of an ego-localization in a global map database that can lead to offset areas.

To summarize the reasons for this effect:

- There is high (systematic or non-independent) noise in the input vector. The algorithm accounts for the noise determined in the runs and

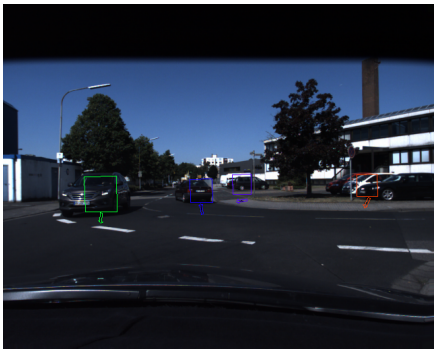
an additional security margin.

- The human driver/observer extracts the vehicle position and the lane position from the camera image. The relative position between the vehicle and the lane is more accurate than a positioning in a global map. Errors in the positioning lead to errors in the modeling (cf. Sec. 7.3.2)

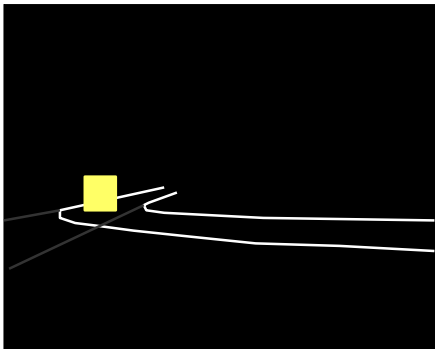
- The human driver/observer extracts the direction (and when observing the scene for longer than a blink, also the velocity) of the vehicle O. The detection algorithm used does not output a detected direction to the ICUBHF. Therefore, the sensed detection cannot be compared with the estimated detection as was done for the position dimension.

Therefore, the results looks moderate to humans. Considering the information input that the ICUBHF has, not much better results are possible. In order to understand this the reader should have a look at Fig. 10.24b. From the pure position sensor input at a certain time step even a human driver will not be able to make a reliable behavior estimation. Fig. 10.24a illustrates the information available to the human driver at the time step of the detection. The vehicle nose has already nearly left the turning lane, but a vehicle in the center of the ROI could also be in a turning behavior with its nose directed to the side. Even when observing the relative position estimates over time (cf. Fig. 10.20) the human driver is not able to make an early detection, since the high noise in the vehicle position detection and the offset noise in the map data positioning has to be considered.

The high noise is a fact that is already considered well by the grid representation. This means that the most effective improvement would be to consider vehicle directions in the behavior detection algorithm. In order to do this, a detection algorithm is needed, that provides reliable data on the vehicle velocity and direction. We will discuss this improvement next to other possible improvements and application areas in our conclusion.



(a) The picture illustrates the system input for the human reader from the camera perspective. The human driver is likely to detect the straight driving as true mode when seeing this camera image. It is the time step 20 from the third evaluation run. The green ROI of the observed oncoming vehicle does not give a clear evidence.



(b) The relative vehicle position is the only input from all possible observable vehicle states. The vehicle position is illustrated by a yellow ROI. The lane markings are illustrated in white. Note that the lane markings are here extracted from the camera image. Our ICUBHF uses the lane information from the lane-graph, which can have an offset.

Figure 10.24: The camera image and the ICUBHF input.

Chapter 11

Conclusion

We developed the theoretical concept of the BHF towards the ICUBHF and used it for behavior detection in a real-world intersection. This behavior detection can be used to anticipate dangerous collision situations by predicting the observed vehicle position estimate in a probabilistic manner. Such a system can be used in future ADAS to provide warnings to inattentive drivers or to execute automatic emergency brakings in situations with high uncertainty.

High uncertainties can either be the result of low sensor performance due to adverse conditions, such as poor visibility, or the result of high prediction uncertainty in the movement due to relatively long-term prediction intervals.

This work provides the fundamental research, shows the technical pitfalls and the strategic decisions necessary to further improve the behavior anticipation of other vehicles by Bayesian filtering and probabilistic comparative measures. Further system improvements and parameter tunings will be needed in order to move beyond the research stage. As a first step, the models can be learned or improved by further expert knowledge. A large data-set with several different intersections is necessary to make these improvements and to make sure that the parameters work for the majority of situations. In order to evaluate the ICUBHF in a large number of situations, the time-consuming and complex manual calibration to set the ego-position $\mathbf{l}_0^{Ego}$ and the ego-yaw ω_0^{Ego} needs to be replaced by a better automated localization technique (at least on the accuracy-level of the KITTI-benchmark) or by on-board lane-detection. In order to achieve a high number of evaluations with different noise in a given set of situations, we have already made use of the high-performance computing cluster (bwHPC) for an automated test bench. The manual calibration prevented us from doing an automated testing for a high number of different real-world situations.

We mentioned different ways to further improve the ICUBHF approach in this thesis. To summarize, the behavior detection and position tracking can be improved by

1. an adaptive grid representation,
2. a sensor fusion of different sensors or usage of multi-modal sensor distributions,
3. an improved vehicle representation,
4. an improvement to the attractor approach,
5. an improved lane grid incorporation,
6. a threshold bootstrapping,
7. the usage of the vehicle heading and velocity for behavior detection,
8. the incorporation of the ICUBHF into a data-association framework for multi-object tracking, and
9. an automatic generation of alternative behavior models based on the lane-graphs.

The adaptive grid representation could be made up of a grid with a coarser grid tesselation in more distant areas and a smaller size of the grid cells in the near-field in order to improve accuracy in the near-field (c.f. 3.2). A sensor fusion of different sensors in parallel allows a fusion of the sensor information within the grid cells in order to improve the accuracy of the estimate. Additionally, the sensors provided for this work do not export multi-modal sensor distributions. The usage of sensors that provide a multi-modal sensor distribution over the position space rather than a single position estimate in each time step will improve the MDBHF and ICUBHF. The vehicle representation of the tracked vehicles can be improved to use the probabilistic representation of the ICUBHF for an in-system computation of evasive maneuvers. At the moment the center of mass of the observed vehicles is tracked. A future system could model the vehicle dimensions. Using the position of the center of mass and the direction of the vehicle at this position the current or predicted occupied space of a certain vehicle can be computed probabilistically. The vehicle dimensions can also be used in the prediction models in order to ensure that the modeled trajectory will keep the entire vehicle within a certain lane. Furthermore, the driver's acceleration and steering profile can be learned in order to improve the prediction model. The attractor algorithm can be further improved to enhance the prediction model. Due to the incorporation of context information into the prediction model the uncertainty, and therefore the variance in the model, decreases. In a future work the information gain due to the context incorporation could be quantified by comparing the model to the real driver behavior. This can be used to quantify the variance of the prediction model when the context information is incorporated. The attractor

algorithm can be further improved by using a more advanced trajectory generation. In future approaches a collision detection between the trajectory and the lane border could be implemented in order to avoid less probable trajectories. The lane or context incorporation can be improved. In the current approach the ego-vehicle has to be manually calibrated within the lane position. On the one hand, self-positioning via GPS within the OSM graph is not accurate enough to improve tracking or detect the behavior. On the other hand, on-board lane-detection is not sufficient due to occlusion of the lane by other vehicles. A combination of both methods can be used to automatically position the ego-vehicle within the context, and either a probabilistic lane marking representation or a probabilistic ego-positioning could be implemented. For improved behavior detection, the threshold parameter used in the behavior detection could be automatically adapted in order to assure a certain false positive and false negative detection rate. To improve the behavior detection, the vehicle heading (and velocity) dimension could also be used for behavior detection. The real-world evaluation shows that this is the most promising improvement, meaning that the plausibility measure should not only compare the position estimates with the sensor input, but also the estimated heading with a sensed vehicle orientation. For this purpose a sensor needs to provide the heading of the observed vehicle. In our evaluation setting this means that a vehicle detection algorithm that works with camera images has to provide the orientation of the detected vehicle. The current heading detection algorithm delivers very inaccurate data. A future project is to develop or implement more accurate algorithms in the real-world evaluation setup. No change would be necessary in the grid representation, since the direction dimension is already represented by a single representative in each grid cell. However, such an enhanced comparison could benefit from a more advanced dimension direction, capable of appropriately representing distributions. This improvement enables early behavior detection, which is needed for the real-world application.

Beyond the discussed automotive application, a system such as the ICUBHF can be used in other fields that have to cope with high uncertainty in the sensor input, that have to cope with highly non-linear prediction models and/or when a prediction over a relatively large time interval is necessary, with the result that the non-linearity of the movement comes into effect. A limitation of this approach is the high computational effort needed when it comes to higher-dimensional problems. Some higher-dimensional problems need to be separated into independent dimensions or the grid resolution needs to be downscaled in order to allow computation. Regarding the vehicle tracking approach it has been shown that the ICUBHF is capable of representing two position dimensions. In order to track the velocity and direction some compromises were necessary, such as using a single representative for the velocity and direction dimensions. Considering these general conditions the ICUBHF can be applied in the following fields:

- for autonomous driving and ADAS to track vehicles, bicycle drivers and other dynamic, non-static and non-linear, goal-oriented moving objects,
- outdoor or indoor-tracking and behavior/goal-anticipation of robots or humans within locally bounded areas using multiple sensors, and
- self-positioning, e.g. via triangulation, since the probability grid allows a probabilistic triangulation through probability accumulation within the grid nodes.

Generally speaking, the ICUBHF can be used for state estimation and state prediction of systems such as humans, machines, mobile robots or robotic arms in low-dimensional spaces.

Despite the variety of possible application fields, the ICUBHF approach was developed to enable behavior and position tracking within the domain of ADAS.

The future work possibilities presented here improve the models and the design of the Bayesian network and should enable the ICUBHF to improve detection quality and detection time in order to provide a behavior detection that performs in a human-like manner, or even better. Such a system contributes to accident prevention measures such as emergency braking and enables an ADAS to act as a co-driver that actively guards the driver in inattentive situations or takes control, in complex situations.

Bibliography

[1] TORCS, the open racing car simulator, 2007. (citation on page 162)

[2] TORCS championship, 2010. (citation on page 80)

[3] A. Alin. Task-Specific Environment Representations for Driving Behavior Generation. Master's thesis, Universität Würzburg, 2010. (citations on pages 51, 53, and 57)

[4] A. Alin, M. Butz, and J. Fritsch. Vehicle with computing means for monitoring and predicting traffic participant objects, 2012. (citations on pages IX, 108, 111, and 115)

[5] A. Alin, M. V. Butz, and J. Fritsch. Tracking moving vehicles using an advanced grid-based bayesian filter approach. In *IEEE Intelligent Vehicles Symposium (IV '11)*, pages 466–472, Baden-Baden, June 2011. (citations on pages 67, 80, 85, and 88)

[6] A. Alin, M. V. Butz, and J. Fritsch. Incorporating environmental knowledge into Bayesian filtering using attractor functions. In *IEEE Intelligent Vehicles Symposium (IV '12)*, pages 476–481, Alcala de Henares, June 2012. (citations on pages 116, 125, 156, and 163)

[7] A. Alin, J. Fritsch, and M. Butz. Improved tracking and behavior anticipation by combining street map information with bayesian-filtering. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 2235–2242, Oct 2013. (citations on pages 145, 156, and 181)

[8] M. Althoff, O. Stursberg, and M. Buss. Erreichbarkeitsanalyse von Verkehrsteilnehmern zur Verbesserung von Fahrerassistenzsystemen. In *Proc. of 3. Tagung Aktive Sicherheit durch Fahrerassistenz*, 2008. (citations on pages 33, 109, and 169)

[9] S. O. Althoff, M. and M. Buss. Sicherheitsbewertung von fahrstrategien kognitiver automobile. *at - Automatisierungstechnik*, 56, no. 12:653–661, 2008. (citation on page 33)

[10] K. B. Anonsen and O. Hallingstad. Terrain aided underwater navigation using point mass and particle filters. In *Position Location and Navigation IEEE Symposium - PLANS*, 2006. (citation on page 24)

[11] K. Atkinson. *An Introduction to Numerical Analysis*. Wiley, 2 edition, 1989. (citations on pages VIII and 98)

[12] H. Badino, U. Franke, and R. Mester. Free space computation using stochastic occupancy grids and dynamic programming. In *Workshop on Dynamical Vision, ICCV*, Rio de Janeiro, Brazil, 10 2007. (citation on page 27)

[13] A. Barth and U. Franke. Where Will the Oncoming Vehicle be the Next Second? In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 1068–1073, Eindhoven, June 2008. Springer. (citation on page 126)

[14] A. Barth and U. Franke. Tracking oncoming and turning vehicles at intersections. In *Intelligent Transportation Systems, IEEE Conference on*, pages 861–868, Madeira Island, Portugal, 2010. (citations on pages 33 and 126)

[15] A. Barth, D. Pfeiffer, and U. Franke. Vehicle tracking at urban intersections using dense stereo. In *3rd Workshop on Behavior Monitoring and Interpretation, BMI*, pages 47–58, Genth, Belgium, 11 2009. (citation on page 33)

[16] I. Ben-Gal. *Encyclopedia of Statistics in Quality and Reliability*, chapter Bayesian Networks. John Wiley & Sons, 2007. (citation on page 5)

[17] J. Bentley. Modelling circular data using a mixture of von mises and uniform distributions. 2006. (citation on page 55)

[18] N. Bergman. *Recursive Bayesian Estimation Navigation and Tracking Applications*. PhD thesis, Linkoping University, 1999. (citation on page 24)

[19] N. Bergman and L. Ljung. Point-mass filter and cramer-rao bound for terrain-aided navigation. 1:565–570 vol.1, 1997. (citation on page 24)

[20] A. Berthelot, A. Tamke, T. Dang, and G. Breuel. Handling uncertainties in criticality assessment. In *Intelligent Vehicles Symposium*, pages 571–576, 2011. (citation on page 31)

[21] A. Berthelot, A. Tamke, T. Dang, and G. Breuel. A novel approach for the probabilistic computation of time-to-collision. In *Intelligent Vehicles Symposium*, pages 1173–1178, 2012. (citation on page 31)

[22] V. Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986. (citations on pages VI and 61)

[23] S. Brechtel, T. Gindele, and R. Dillmann. Recursive importance sampling for efficient grid-based occupancy filtering in dynamic environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3932–3938, 2010. (citation on page 27)

[24] S. Brechtel, T. Gindele, J. Vogelgesang, and R. Dillmann. Probabilistisches Belegtheitsfilter zur Schätzung dynamischer Umgebungen unter Verwendung multipler Bewegungsmodelle. In *Proceedings of the 21th Fachgespräch Autonome Mobile Systeme*, pages 49–56, 2009. (citation on page 27)

[25] R. S. Bucy and K. D. Senne. Digital synthesis of nonlinear filters. *Automatica*, 7:287–298, 1971. (citation on page 24)

[26] J. Burlet, T. D. Vu, and O. Aycard. Grid-based localization and online mapping with moving object detection and tracking. Technical report, INRIA Universit Joseph Fourier - Grenoble I Institut National Polytechnique de Grenoble (INPG), 2007. (citation on page 109)

[27] C. Chen, C. Tay, K. Mekhnacha, and C. Laugier. Dynamic environment modeling with gridmap: a multiple-object tracking application. In *Proc. of the Int. Conf. on Control, Automation, Robotics and Vision*, Singapour Singapore, 12 2006. (citations on pages 27, 28, 70, and 86)

[28] Z. Chen. Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. 2003. (citation on page 24)

[29] T. M. Cover and J. A. Thomas. *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, July 2006. (citation on page 144)

[30] R. Danescu, C. D. Pantilie, F. Oniga, and S. Nedevschi. Particle grid tracking system stereovision based obstacle perception in driving environments. *IEEE Intell. Transport. Syst. Mag.*, 4(1):6–20, 2012. (citations on pages 24 and 30)

[31] M. Darms, C. R. Baker , P. Rybski, and C. Urmson. Vehicle detection and tracking for the urban challenge. In M. M. . C. Stiller, editor, *5. Workshop Fahrerassistenzsysteme*, pages 57–67. Freundeskreis Mess- und Regelungstechnik Karlsruhe e.V. (FMRT), Karlsruhe, 2008. (citation on page 109)

[32] H. de Waard. *A new approach to distributed data fusion.* PhD thesis, Faculty of Science, University of Amsterdam, 2008. (citation on page 86)

[33] C. B. Do and S. Batzoglou. What is the expectation maximization algorithm? *Nature Biotechnology*, 26(8):897–899, Aug. 2008. (citation on page 11)

[34] S. Ehrenfeld and M. V. Butz. The modular modality frame model: continuous body state estimation and plausibility-weighted information fusion. *Biological cybernetics*, 107(1):61–82, 2013. (citations on pages 79, 141, 143, 145, and 146)

[35] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2:24–33, 2003. (citation on page 17)

[36] A. Geiger. The kitti vision benchmark suite. (citations on pages XIV and 176)

[37] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. 2012. (citation on page 176)

[38] T. Gindele, S. Brechtel, and R. Dillmann. A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments. In *IEEE Intelligent Transportation Systems Conference*, Madeira Island, Portugal, September 2010. (citation on page 120)

[39] T. Gindele, S. Brechtel, and R. Dillmann. Learning context sensitive behavior models from observations for predicting traffic situations. In *International Conference on Intelligent Transportation Systems, IEEE*, pages 2235–2242, 2013. (citations on pages 169 and 171)

[40] T. Gindele, S. Brechtel, J. Schröder, and R. Dillmann. Bayesian occupancy grid filter for dynamic environments using prior map knwoledge. In *Proceedings of the IEEE Vehicles Symposium*, Xi'an China, 06 2009. (citation on page 27)

[41] M. Gloderer and A. Hertle. Spline-based trajectory optimization for autonomous vehicles with ackerman drive. 2010. (citation on page 120)

[42] C. Glser, L. Buerkle, and F. Niewels. An inertial navigation system for inner-city adas. In *International Conference on Intelligent Transportation Systems, IEEE*, 2013. (citation on page 142)

[43] F. Gustafsson. *Adaptive Filtering and Change Detection*. Wiley, Oct. 2000. (citation on page 134)

[44] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund. Particle filters for positioning, navigation, and tracking. *Signal Processing, IEEE Transactions on*, 50(2):425–437, Feb 2002. (citation on page 24)

[45] S. C. Kramer and H. W. Sorenson. Recursive bayesian estimation using piece-wise constant approximations. *Automatica*, 24:789–801, 11 1988. (citation on page 24)

[46] T. Kuehnl. *Road terrain detection for Advanced Driver Assistance Systems*. PhD thesis, Bielefeld University, 2013. (citations on pages 70 and 175)

[47] T. Kuehnl, F. Kummert, and J. Fritsch. Spatial ray features for real-time ego-lane extraction. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 288–293. IEEE, 2012. (citations on pages 70 and 175)

[48] L. Le, A. Festag, R. Baldessari, and W. Zhang. V2x communication and intersection safety. Technical report, NEC Laboratories Europe, 2009. (citation on page 69)

[49] X. R. Li and V. P. Jilkov. A survey of maneuvering target tracking – part v: Multiple-model methods, 2003. (citations on pages XI, 136, 137, 138, 139, and 142)

[50] M. Liebner, C. Ruhhammer, F. Klanner, and C. Stiller. Generic driver intent inference based on parametric models. In *International Conference on Intelligent Transportation Systems, IEEE*, pages 268–275, 2013. (citations on pages 116, 118, and 170)

[51] J. Liu, B. Cai, J. Wang, and W. Shangguan. Gnss/ins-based vehicle lane-change estimation using imm and lane-level road map. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 148–153, Oct 2013. (citation on page 171)

[52] H. P. Moreton. *Minimum curvature variation curves, networks, and surfaces for fair free-form shape design*. PhD thesis, University of California, 1983. (citation on page 120)

[53] A. Mueller and H.-J. Wuensche. Object-Related-Navigation for Mobile Robots. In *Proceedings of IEEE Intelligent Vehicles Symposium*, pages 603–610, Alcal de Henares, Spain, June 2012. IEEE. (citations on pages IX and 109)

[54] K. Murphy. A brief introduction to graphical models and bayesian networks, 1998. (citation on page 11)

[55] K. Murphy. A brief introduction to graphical models and bayesian networks. *The Bayes Net Toolbox for Matlab, Computing Science and Statistics*, 33, 2001. (citation on page 5)

[56] H. Nyquist. Certain topics in telegraph transmission theory. *Trans. AIEE*, 47:617–644, 1928. (citation on page 97)

[57] A. Paz, N. Veeramisti, and P. Maheshwari. Life-cycle benefit - cost analysis of alternatives for accommodating heavy truck traffic in the las vegas roadway network. Technical report, 2010. (citation on page 70)

[58] D. S. Petrich, T. Dang, D. Kasper, G. Breuel, and C. Stiller. Map-based long term motion prediction for vehicles in traffic environments. In *International Conference on Intelligent Transportation Systems, IEEE*, pages 2166–2172, 2013. (citation on page 109)

[59] R. R. Pitre, V. P. Jilkov, and X. R. Li. A comparative study of multiple-model algorithms for maneuvering target tracking. *Proceedings of SPIE*, 5809:549–560, 2005. (citation on page 21)

[60] M. Platho, H.-M. Gro, and J. Eggert. Learning driving situations and behavior models from data. In *16th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC2013)*, The Hague, Netherlands, 2013. (citations on pages 116 and 170)

[61] X. Rong Li and V. Jilkov. Survey of maneuvering target tracking. part v. multiple-model methods. *Aerospace and Electronic Systems, IEEE Transactions on*, 41(4):1255–1321, 2005. (citations on pages VI, 21, and 22)

[62] S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. (citations on pages VI, 9, 14, 18, 19, and 21)

[63] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Book, MIT Press, Cambridge, USA, 2004. (citations on pages VI, 52, 53, and 62)

[64] I. Szottka. Particle filtering for lane-level map-mathing at road bifurcation. In *International Conference on Intelligent Transportation Systems, IEEE*, pages 154–159, 2013. (citations on pages XIV, 24, 169, and 170)

[65] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, USA, 2006. (citations on pages XVIII, 8, 11, 13, 17, 18, 20, 21, 22, 23, 24, 25, 42, 43, 52, 57, 73, and 99)

[66] C. W. Ueberhuber. *Computer-Numerik*. Number 2. Springer, 1995. (citation on page 99)

[67] Unknown author. OECD family database, 2012. (citation on page 1)

[68] Unknown authors. Open street map wiki - overpass api entry. (citation on page 177)

[69] C. Urmson. Realizing self-driving vehicles. Keynote talk at the IEEE Intelligent Vehicles Symposium (IV '12) in Madrid, 2012. (citation on page 175)

[70] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, J. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y. Woo-Seo, R. Simmons, S. Singh, J. Snider, A. Stentz, W. . Whittaker, and J. Ziglar. Tartan racing: A multi-modal approach to the darpa urban challenge. Technical report, Carnegie Mellon University, 2007. (citation on page 27)

[71] A. von Eichhorn, M. Werling, P. Zahn, and D. Schramm. Maneuver prediction at intersections using cost-to-go gradients. In *International Conference on Intelligent Transportation Systems, IEEE*, pages 112–117, 2013. (citations on pages 169 and 170)

[72] C. Zhang and J. Eggert. Tracking with multiple prediction models. In *ICANN (2)*, pages 855–864, 2009. (citations on pages 21, 142, 143, and 144)

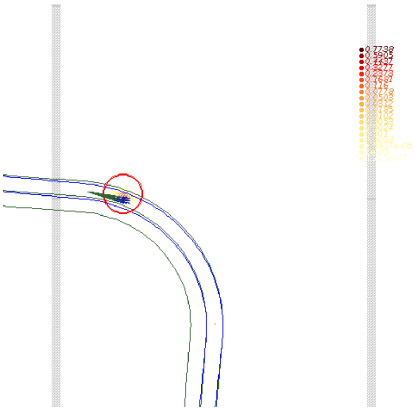
Part V

Appendix

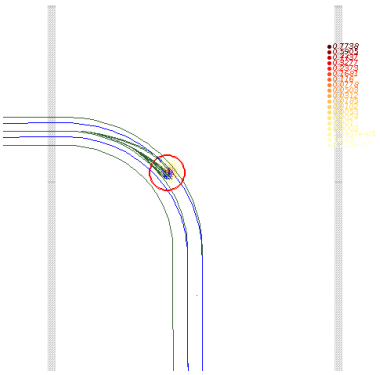
Chapter 12

Further example images from Evaluation

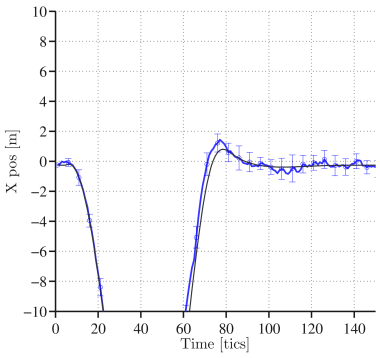
12.1 Attractor function



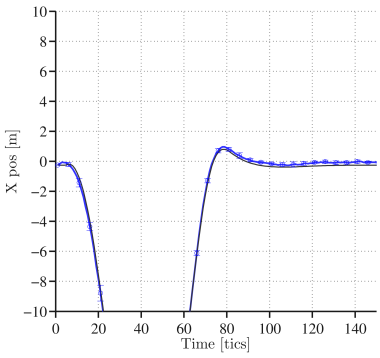
(a) The spline trajectories connect attractor positions with the departing positions. Attractor reachability constraint $d_{max} = 7$ m. (c.f. Eq.7.3)



(b) The spline trajectories connect attractor positions with the departing positions. Attractor reachability constraint $d_{max} = 30$ m. (c.f. Eq.7.3)



(c) The tracking result of the MDBHF.



(d) The tracking result of the ICUBHF.

Figure 12.1: The optimized spline used to improve the tracking of a vehicle in front of the ego-vehicle during a left curve. A larger d_{max} creates smoother trajectories. The second row illustrates the benefits of the ICUBHF in comparison with the MDBHF. When using the ICUBHF, the variance is smaller and the expectation value fits better to the ground-truth position.

12.2 More than two modes

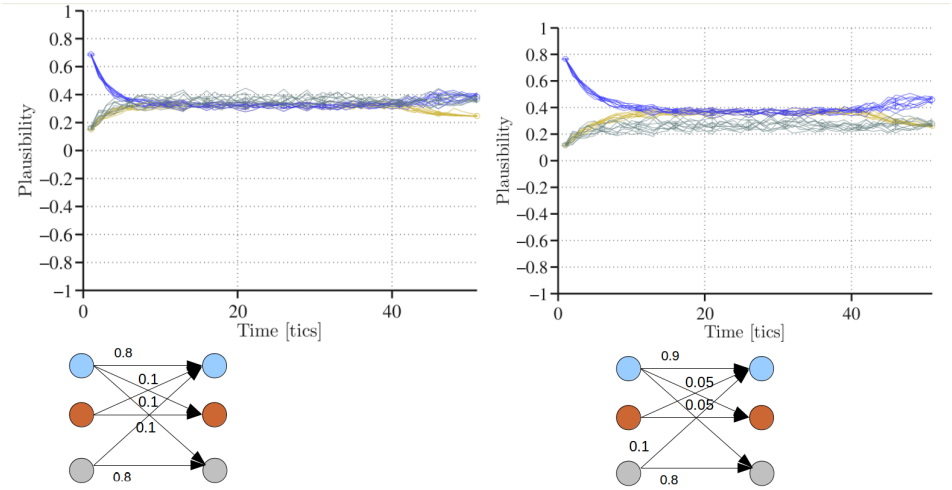


Figure 12.2: The figure shows the plausibility mode graphs in the Carmaker intersection scenario. Different settings were chosen for the mode transition probabilities. Their respective probabilities are annotated below the graphs. A turning mode, a straight-driving mode and a "maniac-driving" mode that uses the kinematic movement constraints only is used for this evaluation. With the transition parameter setting used in the right image, the maniac driving mode is more unlikely than is the left image. In the right image, the transition probability from a maniac-driving mode to another mode was set to 0.1, while the transition probability from another mode to the maniac driving mode was set to 0.05.