Holistic Temporal Situation Interpretation for Traffic Participant Prediction

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

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

genehmigte
DISSERTATION

von

Dipl.-Math. techn. Florian Kuhnt

aus Karlsruhe

Datum der mündlichen Prüfung:	6. Februar 2020
Referent:	Prof. Dr. J. Marius Zöllner
Korreferent:	Prof. Dr. Ralf Reussner

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Abteilung Technisch Kognitive Assistenzsysteme (TKS) am FZI Forschungszentrum Informatik. Unterschiedlichste Personen haben zum Erfolg maßgeblich beigetragen.

Prof. J. Marius Zöllner danke ich für das Vertrauen in meine Arbeit, zahlreiche Diskussionsbeiträge und Anregungen und den durchweg freundschaftlichen Umgang miteinander. Prof. Ralf Reussner danke ich für die Übernahme des Korreferats, sowie Prof. Christian Wressnegger und Prof. Jürgen Beyerer für die Mitwirkung als Prüfer. Außerdem einen Dank allen Professoren, die wertvolle Hinweise zu meiner Arbeit gegeben haben.

Allen Kollegen in den Abteilungen TKS und IDS am FZI danke ich für die angenehme Arbeitsatmosphäre und die Unterstützung auf unterschiedlichste Art. Insbesondere anfangs Thomas Schamm, Dennis Nienhüser, Ralf Kohlhaas, Marcus Strand, Thomas Gumpp und Tobias Bär für die schnelle Aufnahme in die Gruppe und das Kennenlernen effizienter Herangehensweisen. Während der hauptsächlichen Forschungszeit haben Sebastian Klemm, Jan Oberländer, Marc Zofka, Jens Doll und Ralf Kohlhaas durch intensive Diskussionen weit über die Arbeitszeit hinaus zum Gelingen dieser Arbeit beigetragen. Neue Impulse und gelungene Fortführungen meiner Arbeiten waren mit Christian Hubschneider, Dominik Petrich, Tobias Fleck, Stefan Orf und Karl Kurzer möglich.

Neben den wichtigen Abschlussarbeiten von Jens Schulz, Tobias Fleck, Patrick Bartler und Stefan Orf haben auch alle anderen von mir betreuten Studenten über ihre Anregungen und Diskussionen zur Schärfung des Dissertationsthemas beigetragen. Stefan Ulbrich danke ich für aufmunternde Gespräche und Hilfestellungen aus Sicht eines Postdoc in schwierigen Situationen. Christoph Reller hat überraschend unkompliziert und ausdauernd zur graphischen Darstellung von Forney-style Faktorgraphen beigetragen und mit Jörg Henß konnte ich mich über die Metamodellierung formaler Sprachen austauschen. Tobias Fleck, Karl Kurzer und Edgar Gerteisen danke ich für das abschließende Korrekturlesen und die richtigen vollendenden Impulse.

Ein großer Dank gilt meinen Eltern, die mich während meines Studiums und der Promotion immer unterstützt haben. Am meisten danke ich meiner Frau Nina für ihre pragmatischen Lösungen, ihr Verständnis und ihre besonders große Geduld.

Karlsruhe, Februar 2020 *Florian Kuhnt*

Zusammenfassung

Die Entwicklung autonomer Systeme für den Straßenverkehr ist in den letzten Jahren stark voran geschritten. Aus einzelnen Fahrerassistenzsystemen sind Komponenten gereift, die zunehmend Fahrfunktionen übernehmen können. Kombinierte Systeme ermöglichen bereits autonomes Fahren auf Autobahnen unter strukturierten Umgebungsbedingungen. Um sichere Verhalten und Trajektorien planen zu können, müssen Verkehrssituationen ausreichend verstanden werden, wofür nicht-beobachtbare Beziehungen aus sensoriell beobachtbaren Eigenschaften der Objekte abgeleitet werden müssen. Bisher wird hierfür eine lineare, aufeinander aufbauende Kette von Teilkomponenten verwendet. Zwar werden Sensor- und Modellunsicherheiten in den Teilkomponenten weitgehend berücksichtigt - an Modulgrenzen werden aber starke Vereinfachungen getroffen. So werden z.B. während einer sensorbasierten Lokalisierung auftretende Multimodalitäten nur mithilfe der hierbei zur Verfügung stehenden Informationen aus den Sensordaten analysiert und anschließend nur eine der Möglichkeiten an die weiter interpretierenden Module weitergegeben. Wünschenswert ist hier, auch das Situationswissen verwenden zu können, um die Multimodalitäten besser auflösen zu können. Um dies zu ermöglichen müssen die Komponenten Lokalisierung, statische und dynamische Umgebungserfassung bis hin zur Schätzung des komplexen Situationswissens einheitlich modelliert werden. Folgende Prinzipien sollen dadurch konsequent ermöglicht werden:

- Fusion verschiedenster Umgebungsbeobachtungen
- Nutzung von höherwertigem Wissen in Basisschätzungen
- Berücksichtigung temporaler Zusammenhänge

Ziel der Arbeit ist eine einheitliche Modellierung von situationsschätzenden Verfahren um die präzise Prädiktion von Verkehrsteilnehmer-Verhalten unter verschiedensten sensoriellen und technischen Voraussetzungen zu ermöglichen. Die Herausforderungen der Modellierung von Unsicherheiten und bedingter Abhängigkeiten, der steigenden Modellkomplexität, der Wahl der Inferenzmethodik und des gezielten Einsatzes von Lernverfahren sollen ganzheitlich adressiert werden.

In dieser Arbeit wird der Ansatz verfolgt, eine Modellierungssprache zu definieren, um darauf basierend ein einheitliches generisches Grundmodell für Verkehrssituationen aufzustellen, das anschließend für verschiedene Anwendungen unterschiedlich detailliert wird.

Um in der einheitlichen Modellierung Unsicherheiten berücksichtigen und diese über Beziehungen durch verschiedene Variablen und Zustandsräume transportieren zu können, wird als Basis ein Faktorgraph gewählt, aus dem verschiedene Modellierungsverfahren für bedingte Abhängigkeiten (Bayes'sche Netze), Korrelationen (Markov Random Fields) oder Glaubwürdigkeit (Evidence Theory) abgeleitet werden können.

Die steigende Modell-Komplexität, insbesondere durch die Betrachtung von Relationen zwischen sämtlicher Entitäten der Verkehrsszene, wird durch eine geeignete objektorientierte Modellierung bewältigt, die sowohl eine hierarchische Modellierung zulässt, als auch mehrfach instanziierbare Klassen, die untereinander in Relation gesetzt werden können.

Je nach Anwendung und Zustandsräumen sind unterschiedliche Inferenzverfahren notwendig. Diese sind teils aus etablierten Inferenzverfahren wie z.B. Kalman Filter ableitbar und lokal anwendbar, während für andere Teile speziell angepasste Verfahren angewendet werden.

Während durch Regeln begründbare Beziehungen, wie z.B. geometrische Abhängigkeiten oder Verkehrsregeln, durch Expertenwissen modelliert und parametrisiert werden können, ist dies z.B. für komplexe Verhaltensmodelle der Verkehrsteilnehmer im Allgemeinen nicht möglich. Die einheitliche Modellierung lässt zu, gezielt bestimmte Abhängigkeiten durch Techniken des maschinellen Lernens zu lernen.

Die generische Modellierungssprache vereint insgesamt viele gängige Prinzipien und Modelleigenschaften in einem Modell: Sie ist modularisierbar, hierarchisch aufgebaut mit Unterstützung für Klassen, Instanzen und Relationen zwischen einzelnen Entitäten. Abhängigkeiten können probabilistisch modelliert werden, um Prinzipien wie beobachtbare und versteckte Variablen und temporale Filterung umzusetzen. Hybride Zustandsräume mit verschiedenen Zustandsraumapproximationen (z.B. Gauss'sch, partikel- oder gitterbasiert) sowie angepasste Inferenzverfahren können je nach Anwendung, abhängig von der Struktur der Teilgraphen und der Konstellation der anliegenden Beobachtungen aus Sensorik und Hintergrundwissen und der zu schätzenden Variablen, passend gewählt und kombiniert werden. Die probabilistischen Abhängigkeitsmodelle werden entweder von Experten parametrisiert oder durch Maschinelles Lernen automatisiert gelernt. Die einheitliche Modellierung bietet eine Basis, um bestehende Prinzipien und Komponenten einzuordnen und kombinierbar zu machen und stellt gleichzeitig eine Plattform für neue durchgängige Schätzmöglichkeiten zur Verfügung. Beispielhaft wird dies an verschiedenen Anwendungen gezeigt.

Als Evaluierung wird die Anwendbarkeit der Sprache beispielhaft an verschiedenen Anwendungsszenarien gezeigt. Diese reichen von der Selbstlokalisierung anhand von Objekt-Bewegungsbeobachtungen, über die hierarchische Straßenlayout-Schätzung aus Kameradaten bis hin zur Vorhersage von Fahrzeugbewegungen unter Berücksichtigung von Relationen, insbesondere Interaktionen zwischen Verkehrsteilnehmern. Dabei werden die Möglichkeiten der Sprache konse-

quent auf die Anforderungen der jeweiligen Anwendung angewandt und ein Zusammenhang zu etablierten Prinzipien wie z.B. Bayes'sche Filter für die zeitliche Fortschreibung oder Markow Random Fields für geometrische Konstellationen hergestellt. Die Anwendungen wurden gewählt, um einerseits möglichst das gesamte Spektrum der zu bewältigenden Teilaufgaben des autonomen Fahrens, andererseits möglichst unterschiedliche Aspekte der Modellierungstechnik abzudecken.

Durch die Arbeit wird ein wichtiger wissenschaftlicher Beitrag sowohl auf Ebene der Gesamtsystemmodellierung (einschließlich Inferenz) als auch im Bereich der gewählten Anwendungen geleistet:

Beiträge zur Gesamtsystemmodellierung

- Erweiterung objektorientierter probabilistischer relationaler Modelle zur Anwendung als durchgängiges Modellierungsverfahren für modulare Schätzsysteme (Kapitel 3).
- Vereinheitlichung verschiedenster Schätzverfahren der Perzeption und des Szenenverstehens (Grundlagen in Kapitel 2, einheitliche Verwendung in Kapitel 6 - Kapitel 9).
- Objektorientierte Faktorgraphen zur einheitlichen Schätzung räumlicher, zeitlicher und semantischer Beziehungen in Verkehrsszenen (Kapitel 5 - Kapitel 9).

Beiträge zu Anwendungen im Bereich Automatisiertes Fahren

- Neuartiges Verfahren zur Selbstlokalisierung auf Straßenkarten anhand von interpretierten Umgebungsmessungen (Kapitel 7).
- Erweiterung eines bestehenden Ansatzes zur hierarchischen Straßenlayoutschätzung um temporale Fusion und eine geschickt gewählte lokale Zustandsraumrepräsentation (Kapitel 8).
- Schätzung von Interaktionen zwischen Verkehrsteilnehmern als konsequente objektorientierte Erweiterung des Interacting Multiple Model Filter-Prinzips (Kapitel 9).

Abstract

For a profound understanding of traffic situations including a prediction of traffic participants' future motion, behaviors and routes it is crucial to incorporate all available environmental observations. The presence of sensor noise and dependency uncertainties, the variety of available sensor data, the complexity of large traffic scenes and the large number of different estimation tasks with diverging requirements require a general method that gives a robust foundation for the development of estimation applications.

In this work, a general description language, called Object-Oriented Factor Graph Modeling Language (OOFGML), is proposed, that unifies formulation of estimation tasks from the application-oriented problem description via the choice of variable and probability distribution representation through to the inference method definition in implementation. The different language properties are discussed theoretically using abstract examples.

The derivation of explicit application examples is shown for the automated driving domain. A domain-specific ontology is defined which forms the basis for four exemplary applications covering the broad spectrum of estimation tasks in this domain: Basic temporal filtering, ego vehicle localization using advanced interpretations of perceived objects, road layout perception utilizing inter-object dependencies and finally highly integrated route, behavior and motion estimation to predict traffic participant's future actions. All applications are evaluated as proof of concept and provide an example of how their class of estimation tasks can be represented using the proposed language. The language serves as a common basis and opens a new field for further research towards holistic solutions for automated driving.

Contents

Acronyms

AADC Audi Autonomous Driving Cup. x, 163, 165, 168, 175, 180

ADAS Advanced Driver Assistance Systems. x, 1, 7, 163

CHM Compositional Hierarchical Model. x, 164, 165, 172

DSL domain specific language. x, 68, 69, 74, 89, 95, 96, 101, 104, 107, 109, 128, 143, 159, 165, 180, 187, 222

FFG Forney-style factor graph. x, 20, 24, 29, 44, 46, 47, 50, 57, 78, 212, 229, 230, 237

FOPL first-order probabilistic language. x, 9, 55–58, 60, 63, 70, 79, 139, 184, 201, 202, 223

HMM Hidden Markov Model. x, 41

IDM Intelligent Driver Model. x, 210, 211

IMM filter Interacting Multiple Model filter. x, 40, 45, 48–53, 59, 116, 204, 206, 212, 213, 221, 223, 224, 230, 237–239

MHT multiple hypothesis tracking. x, 53

OOBN object-oriented Bayesian network. x, 55, 56, 79

OOFGML Object-Oriented Factor Graph Modeling Language. v, x, 63, 68, 69, 72, 74, 77–79, 82–84, 86, 87, 89–92, 94, 95, 98, 101, 107, 109, 110, 130, 132, 139–143, 163, 169, 174–176, 183, 184, 196, 199, 201–203, 209, 213, 222, 224, 225, 231, 232, 234, 237

OPRM object-oriented probabilistic relational model. x, 203, 204, 223, 237

OPRML object-oriented probabilistic relational modelling language. x, 56, 61, 63, 70, 72, 77, 79, 89, 139, 202

PRM probabilistic relational model. x, 56

RANSAC Random Sampling Consensus. x, 164

RMSE root mean square error. x, 222

TDL Traffic Domain Language. x, 95–97, 101, 231

UML unified modeling language. x, 65, 228, 230

1 Introduction

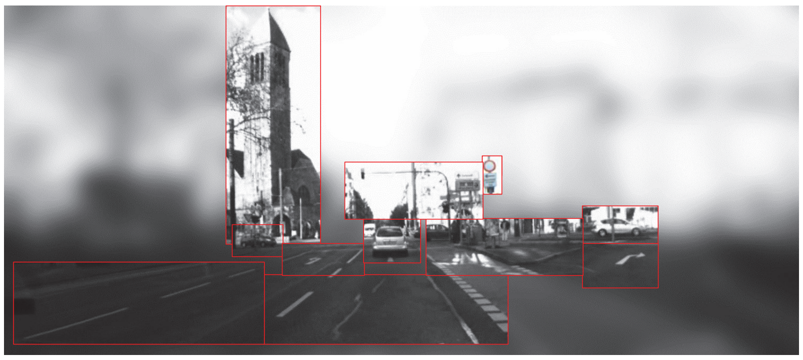
Self driving cars have always been part of visions about the future. Traveling in individual vehicles without the need to control them will allow relaxing, sleeping or focusing on other tasks. Energy draining daily commutes will be transformed into efficiently used time. Additionally, the automation of traffic (together with intelligent infrastructure) has the potential to improve safety as well as the time and energy efficiency on the road.

On the way to this goal, research in the field of automated driving has already come a long way: Since the first vehicle was able to autonomously follow lanes in the 1980s [43], many results have been integrated in production vehicles. Today's vehicles are equipped with a variety of Advanced Driver Assistance Systems (ADAS) that can be combined to allow handsfree driving on highways given good environmental conditions. But harsh environment conditions like bad weather or noncompliant behavior of other traffic participants impair the functionality and thus a human driver always has to supervise the system.

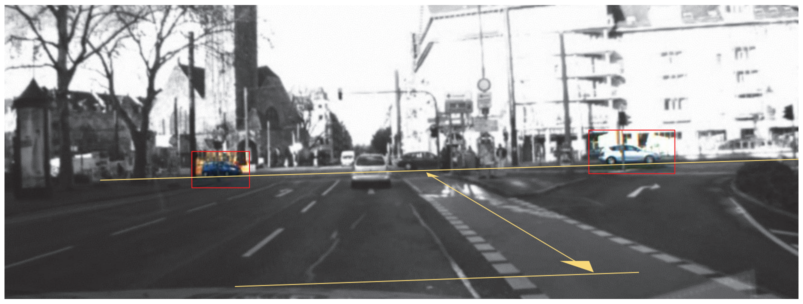
Past competitions such as the DARPA Urban Challenge or the Bertha Benz Challenge demonstrated autonomous driving in urban and rural environments. Nevertheless, much effort has to be put into preparing map data and not every special situation can be considered by developers. The capabilities of automated vehicles are still lagging behind their human counterparts: Humans are very good in perceiving and understanding the environment.

Three of the main tasks humans are outperforming machines are (1) creating a coherent scene understanding from multiple environment measurements, (2) estimating basic knowledge from scene understanding and (3) deriving advanced interpretations from temporal context (Fig. 1.1). To close the gap these principles have to be integrated consequently into the development of future autonomous driving systems. Instead of a sequential process consisting of independently developed modules an overall model including all estimation tasks equally has to be implemented (Fig. 1.2).

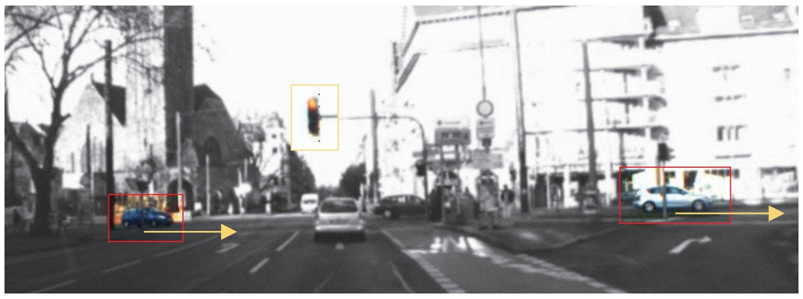
Following this hypothesis further challenges arise: The model has to generalize over many different vehicle types. It has to handle uncertainties in map and sensor data. It has to be scalable to traffic scenes including many vehicles interacting with each other. Complex behaviors resulting from human drivers have to be representable. Estimation has to be efficient and balanced between accuracy and runtime requirements, which are highly task dependent, e.g. roughly predicting the future during a complex driving situation or precisely calculating a



(a) Coherent scene understanding from multiple environment measurements.

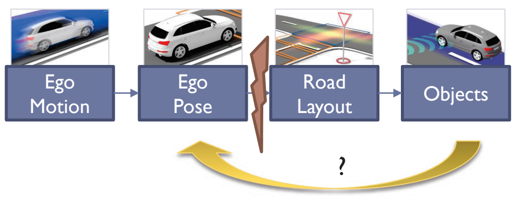


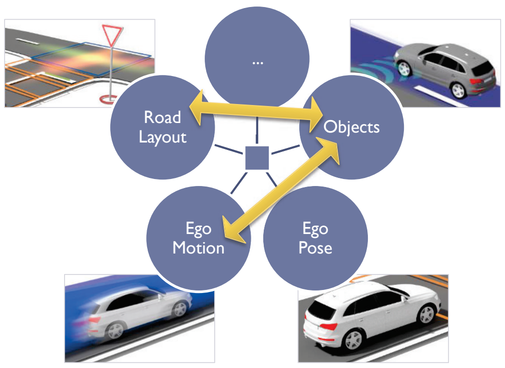
(b) Estimating basic knowledge from scene understanding, e.g. the ego vehicle's position relative to an intersection where crossing vehicles are observed.



(c) Advanced interpretations from temporal context, e.g. a red traffic light from crossing vehicles.

Figure 1.1: Three of the main tasks, where human perform better than autonomous driving systems.





(a) Sequential tool chain (b) Overall coherent model

Figure 1.2: Existing approaches make use of a sequential tool chain consisting of independently developed modules. This work proposes an overall coherent model considering all estimation tasks equally.

longterm map offline. The model has to be independent of state space representations including discrete and continous environment attributes. Many attributes only depend on a subset of other attributes and thus have to be handled in isolation. Development of future autonomous driving systems will always face these challenges.

The goal of this thesis is to contribute to the development process of future autonomous driving systems by proposing an overall probabilistic modeling language for estimation problems and a domain specific specialization to the traffic domain. The approach is applied to selected autonomous driving applications to show its feasibility and potential to advance towards human estimation capabilities.

1.1 Problem Statement

All sub tasks, such as estimating a current state, predicting future states or learning longterm generalizations, can be formulated as an estimation problem: Given a set of observable measurements the state of all scene objects have to be estimated.

Measurements origin from sensors including ego vehicle properties like ego motion and pose estimates and environment sensors including cameras and radar sensors that perceive environment objects like lanes, traffic signs and other traffic participants. Additionally information from longterm map data and other knowledge sources can be used as measurements. Unobservable properties of the scene like plans and trajectories of dynamic objects have to be derived from the observable measurements.

Due to the noise of the measurements the true configuration of the unobservable properties cannot be calculated in general. Instead a probability distribution over the space of possible solutions can be derived considering the uncertainties of the measurements. In general the precise distribution is not calculatable and has to be approximated depending on the focus of the application and the capabilities of the system and sensors. While the runtime is very important for realtime applications like state estimation and prediction, longterm learning and mapping applications can focus more on quality. Thus also the inference implementation depends on the application.

For a coherent modeling of all estimation tasks in the field of autonomous driving a common modeling approach is needed that raises all applications to a common level and enables application specific specialization at the same time. In particular, the following research questions are addressed:

- How can a generic modeling language for estimation problems be formulated?
- How can domain specific languages be created using the generic modeling language?
- What is a domain specific language for all estimation problems in the traffic domain?
- How can application specific models be derived from the traffic domain language?
- How can the model be used to realize a fusion of several environment measurements?
- How can local dependency models of unobservable human behavior be learned?

Therefore, this thesis introduces a generic language, applies it onto the traffic domain and derives models for different realistic applications. The applications use varying assumptions about the sensor and processing hardware capabilities.

1.2 Thesis Statement

The thesis statement is:

"Several components and modules for advanced scene understanding and prediction can be described in a unique, holistic and human-readable modeling language supporting the development process from problem formulation to inference implementation."

This thesis statement is substantiated by introducing a novel approach for holistic dependency modeling based on extensions of established models for probabilistic reasoning and evaluating this approach on exemplary applications spanning over a wide spectre of estimation problems.

1.3 Concept Overview

The basic idea of the concept of this thesis is to create a modeling language that supports the devolopment process of future autonomous driving systems.

The concept (Fig. 1.3) consists of the generic modeling language, a domain specific adaptation of the language and the actual applications. While the generic modeling language integrates all properties necessary to describe estimation tasks and implementations, the domain specific language uses these properties to describe the domain including entities and relations. This results in a probabilistic ontology. The focus of the domain specific language lies on formalizing the real world. Applications can use the domain specific language as basis, refine it and derive an approximation and inference method for the given task.

The generic modeling language combines four key features:

A factor graph is chosen to consider uncertainties and transport them by relations over several variables and state spaces. Several probabilistic graphical modeling approaches for conditional dependencies (Bayesian Networks), correlations (Markov Random Fields) or credibility (Dempster Shafer Evidence Theory) can be formulated by factor graphs.

A suitable implementation of object orientation overcomes the increasing model complexity, especially originating from considering relations between all entities of the scene. It allows hierarchical modeling as well as multiply instantiated classes with relations among them.

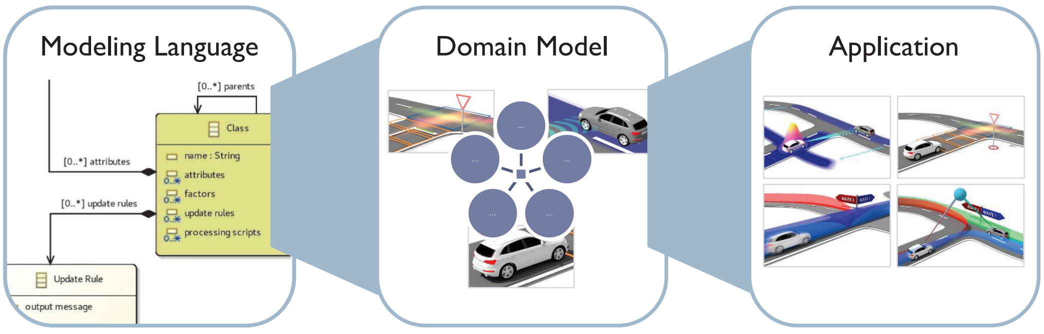
Different inference methods depending on the application and the state spaces are necessary. These are derived from established inference methods like e.g. Kalman filter and are locally applicable while for other parts specially adapted methods can be applied.

Whereas some dependencies can be justified by rules, like for example geometric relations or traffic rules, and therefore can be adjusted by expert knowledge, it is not possible to do this for complex behavior models. The generic modeling supports learning these dependencies from observations using machine learning techniques.

The generic modeling language forms the foundation for all these model properties. Using the domain specific language as a common basis for entities and relations every application can choose and define different property components to solve the specific task.

1.4 Contributions

This work contributes to the understanding of estimation problems in the field of robotic applications, especially in the traffic domain. New methods are intro-



CHAPTERS			
	3 Object-Oriented Factor Graph Modeling Language	2 Estimation Problems in Automated Driving	6 Basic Example: GNSS Tracking
	4 Language Properties	5 Traffic Domain Language	7 Ego Vehicle Localization
			8 Road Layout Estimation
			9 Route, Behavior and Trajectory Estimation

Figure 1.3: The concept consists of the generic modeling language, a domain specific adaptation of the language and the actual applications in the domain. The chapters in this work are correlated to these aspects of the concept.

duced for specific autonomous driving related estimation problems by leveraging the proposed generic modeling language.

Although generic probabilistic modeling languages exist, such as the Object Oriented Probabilistic Relational Modelling Language [64], they are limited in several aspects. These languages are extended by integrating various graphical modeling techniques and defining the inference methods inside the object oriented class structure for the first time. Such a language can be used throughout the whole development process from design stage to implementation or even at runtime.

Various estimation methods that have been developed over years and originate from different applications are now handled in one common language. Algorithms that had no connection before can thus be brought into one model, enabling analysis of the complete system in a coherent way. Perception and scene understanding tasks can be modeled in a unified manner but also still separated into specific modules. Applied to the traffic domain this helps to understand how spatial, temporal and semantic relations can be modeled and estimated in a unified way.

In the field of self localization on road maps a new estimation method is derived from the traffic domain language. So called map matching algorithms are a common way to roughly localize ego vehicles on road maps with simple sensors such as a GPS sensor. Advanced interpreted object information that can be provided by an integrated environment object detector, such as a stereo camera or radar, that is already used in current ADAS improves the localization inside the road boundaries and advances map matching to a lane-precise localization method.

Road layout perception is another task that has been studied in recent years to advance from single lane detection to estimating the whole road layout with varying number of lanes and intersections. An approach for hierarchical road estimation on single time frame basis [124] is extended by temporal fusion including the ego vehicle motion and the stationary road layout. A well-chosen grid based state representation overcomes the association problem in the fusion.

The estimation of interactions between traffic participants is one of the tasks in autonomous driving that has not been solved yet but has the potential to bring autonomous vehicles much closer to human performance. The object oriented structure of the proposed modeling language is used to describe traffic participants and interactions by entities and relations. Two different approaches are shown that consider single time step condensed state analysis and multi time step motion model estimation. The two methods show that considering interactions improves the prediction of traffic participant behaviors, on discrete route and behavior level as well as on continuous trajectory level.

The main contributions can be summarized in two groups:

Contributions to Holistic System Modeling

1. Extension of Object Oriented Probabilistic Relational Models to holistic modeling approach for modular estimation systems (Chapter 3).
2. Unification of various estimation approaches for perception and scene understanding (basics in Chapter 2, unified usage in Chapter 6 - Chapter 9).
3. Object Oriented Factor Graphs for unified estimation of spatial, temporal and semantic relations in traffic domain (Chapter 5 - Chapter 9).

Contributions to Autonomous Driving Applications

1. New approach for self localization on road maps using interpreted environment measurements (Chapter 7).
2. Extension of an existing approach for hierarchical road estimation by temporal fusion and a wisely chosen local state representation (Chapter 8).

3. Estimation of interactions between traffic participants as a consequent object oriented extension of the Interacting Multiple Model Filter principle (Chapter 9).

All in all the proposed approach opens a new field of research topics including language driven estimation system modeling, adaptive system implementation and learning on object-oriented factor graphs.

1.5 Document Outline

The rest of the document follows the following structure, also depicted in Fig. 1.3:

Chapter 2 introduces terms relevant to the traffic domain and autonomous driving and evaluates the related work considering estimation problems and probabilistic graphical models. In Chapter 3 the proposed unified modeling language is defined and its general capabilities are evaluated in Chapter 4. The modeling language is applied to the traffic domain in Chapter 5 introducing a domain specific ontology that is the basis for all applications. A basic example is given in detail in Chapter 6 to show how the adaptation to a specific application works before the language is used for typical more complex tasks in the autonomous driving domain such as ego vehicle localization (Chapter 7), road layout perception (Chapter 8) and traffic participant prediction (Chapter 9). Each of these applications serves as evaluation of the generic language and is additionally evaluated against existing approaches in the given application's domain. A conclusion and outlook is given in Chapter 10.

2 Estimation Problems in Automated Driving

This chapter forms the basis for subsequent chapters considering the automated driving domain on the one hand and probabilistic estimation problems on the other hand.

The very basic knowledge about math and probability theory can be found in cited references and is skipped in favor of a more detailed description of probabilistic graphical models and their relation to automated driving.

Starting with an explanation of terms in the traffic domain (Sec. 2.1) and automated driving components (Sec. 2.2), a problem formulation of the estimation problem using probability theory is given in refsecsec:estimation-problems which leads to the description of local dependencies by probabilistic graphical models (Sec. 2.4). After a detailed view on different aspects considering inference in probabilistic graphical models (Sec. 2.5) the factor graph representations of many well-known filter methods are discussed in detail in Sec. 2.6). A short introduction to first-order probabilistic languages (Sec. 2.7) is given before the seen estimation methods are correlated to their application in components for automated driving (Sec. 2.8). A summary about the key points learned in this chapter is given in Sec. 2.9.

2.1 Terms in Traffic Domain

This work handles a topic in the field of public road traffic. Elements that are created to help human drivers maneuvering safely on public road without colliding with other traffic participants have to be handled by machines. Although these elements might be known, their definition will be clarified here.

Fig. 2.1 gives an overview about the most relevant elements. The environment consists of static and dynamic elements, being the guiding infrastructure and the moving traffic participants respectively. The traffic participants can be *vehicles*, *bicycles*, *pedestrians* and other means of transportation. Vehicles can be divided in various subgroups, some of the important ones are *Cars* and *Trucks* to differ their different size and acceleration capabilities.

The most important infrastructure element is the *road*, the ground where vehicles drive on. It can be an urban or suburban blacktop road, a multilane highway

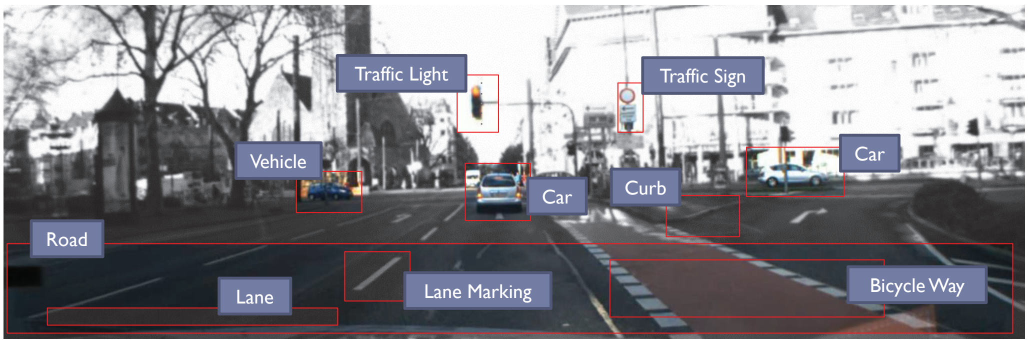


Figure 2.1: Exemplary terms that can be found in a traffic scene.

or a simple dirt road. Roads are usually separated from non-drivable area by a change of material (e.g. for example asphalt to grass), low and high curbs, or lane markings. Roads with intense traffic are divided into multiple *lanes* separating the driving directions and also giving a guideline for multiple vehicles driving in parallel in the same direction. The lanes are observable by different *lane markings* (dashed and solid), the road boundaries and sometimes only by the tracks of other vehicles or their movement (e.g. on snow covered roads). Lanes are a very convenient level of detail to guide vehicles and to create a topology map of the drivable area. Also other traffic participants like bicycles and pedestrians are moving on lane-similar elements like *sidewalks*, *pedestrian crossings* and *bicycle ways*.

Traffic signs introduce commandments and prohibitions to regulate the traffic participants' behaviors. They are usually mounted on poles and can be accompanied by symbols printed on the road surface. Variable signs (electronical and mechanical) can be quickly adjusted to accomodate to a changing traffic situation, e.g. lower speed limits on congested highways. *Traffic lights* are another way of dynamically regulating traffic flow. Especially at large, busy and occluded intersections with high crossing speeds, traffic lights are the only way to achieve a safe operation. There are special traffic lights for vehicles, bicycles, pedestrians and trams that have to be considered differently.

Since all these elements were created with a focus on human traffic participants, it is challenging for automated vehicles to perceive all of them. Especially estimating the state of traffic lights is one of the challenging tasks that cannot be circumvented by using a predefined map. A recent approach is introducing *Vehicle-To-X communication devices* into vehicles and infrastructure to directly communicate in a machine understandable way between infrastructure and vehicles (e.g. traffic light states) and between vehicles and vehicles (e.g. vehicle positions, plans).

In Germany, lane markings are white (yellow in construction sites) and there is a right-driving-commandment on roads with spatially separated driving directions (meaning drivers have to choose the right-most lane if there is no slower vehicle on these lanes).

2.2 Automated Driving Components

The goal when implementing automated vehicles is to master transportation tasks in public road traffic. This means driving from a starting point to a target point efficiently (appropriate time/costs) and without exposing anything to danger, neither the passengers of the vehicle itself nor the other traffic participants. Similar to a human the automated vehicle has to perceive the environment and has to react with proper reactions.

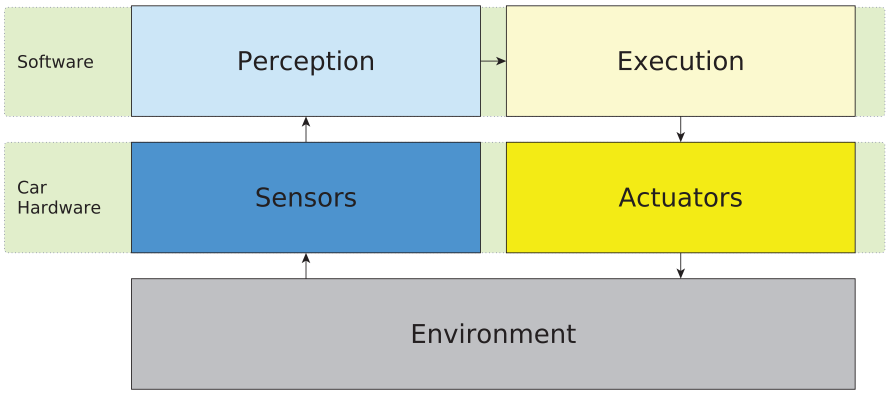


Figure 2.2: Generic concept of an agent observing an environment and acting on it.

In general this is described by the model of an intelligent agent [112] (Fig. 2.2). The agent observes its environment using sensors, understands the data and plans a decision that is then executed using its actuators. The action influences the environment and the effect can again be observed by the sensors. The perception and execution is also often called *cognition* what highlights the intelligence in the intelligent agent. It is quite common that somewhere in the internal process an environment model representing the real environment is created. Due to uncertainties in sensing principles and estimation models only an approximate estimation of the environment model is possible. These uncertainties and also the uncertainty in the effect of the executed actions make the whole estimation problem a very challenging task. These problem properties are already known for a long time. Different approaches try to handle them in different level of detail and modularize them into manageable subtasks [14].

The typical subtasks in the automated driving domain are explained at an exemplary implementation (Fig. 2.3).

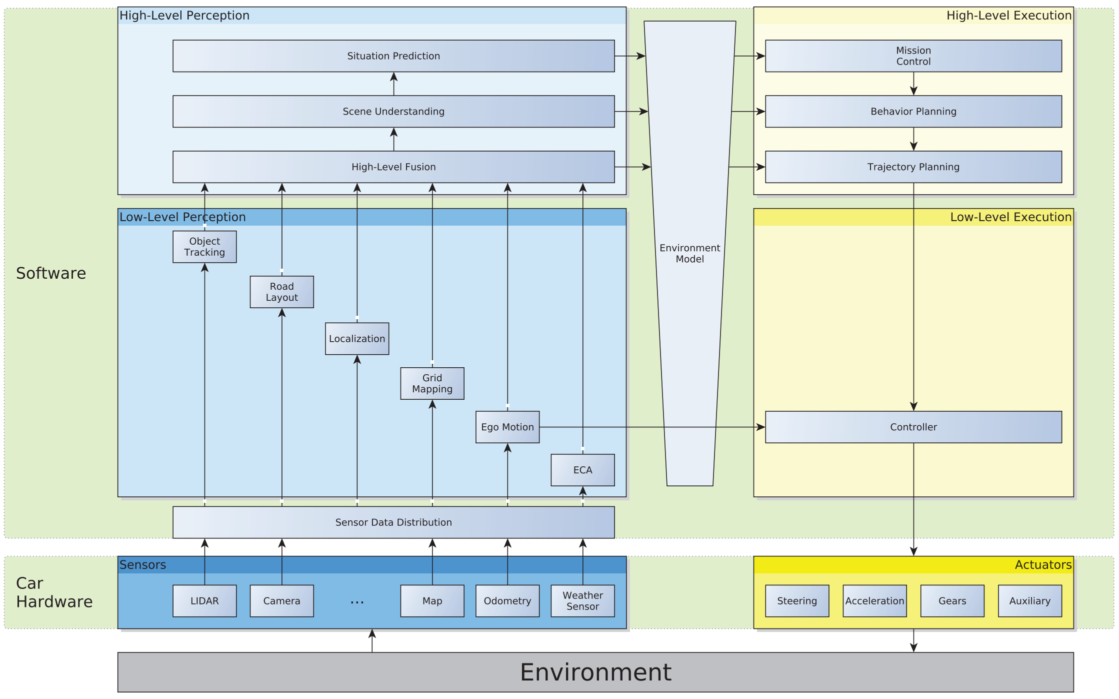


Figure 2.3: An exemplary implementation of the generic concept including all major estimation tasks.

2.2.1 Sensors

Various sensors can be used in modern automated vehicles. They are roughly grouped in internal state estimation sensors, environment perception sensors and knowledge sources[1]. The most important ones are listed here.

Internal State Estimation Sensors

Odometry sensors directly observe the position of actuators such as the steering and the wheel rotation. Usable information is the steering angle and the longitudinal velocity.

Inertial measurement units (IMU) measure the linear and rotary forces acting on a body, sometimes also the magnetic field surrounding it. From this raw data velocity and acceleration values in all dimensions can be derived.

Global Navigation Satellite System (GNSS) sensors use global navigation satellites (such as NAVSTAR GPS, GLONASS, Galileo and Beidou) to estimate the

[1]Knowledge sources are actually no real sensors in the sense of perceiving the environment by a physical measurement technique. But they deliver information to the system and are similarly affected by uncertainties like other sensors.

body's position on the earth surface. The sensor is actually an environment perception sensor that observes the distance to reachable satellites and derives the body's position from these distances. Because GNSS sensors are only usable to estimate the vehicle's position, they are listed here. GNSS sensors can be improved with differential GNSS comparing the measurements to a local static reference sensor with known position to calculate the changing athmospherical error and thus improve the position estimation.

There are many integrated solutions combining the above sensors, such as the ones from GeneSys (ADMA) and Oxford Technical Solutions Ltd. (OXTS).

Environment Perception Sensors

Weather sensors such as temperature sensors or rain sensors measure values about the current weather condition. They are helpful for automatic windshield activation but also for adjusting a driving behavior to bad weather conditions.

Camera sensors are one of the closest sensors to human perception. Due to their sensing principle and their usage in many other applications they are a very low-priced solution for achieving a detailed image of the surrounding environment. Their drawback is that depth information cannot be directly measured and machine learning techniques are usually needed to interpret the data.

Ultrasonic sensors are already widely distributed in current production vehicles. They can estimate free space in the closer surrounding around the vehicle and are already used for park assist systems.

Radio Detection and Ranging (Radar) sensors can measure distance and velocity of remote objects. They are robust against harsh weather conditions like snow and fog and are used in production vehicles to implement adaptive cruise control (ACC) and automated emergency braking (AEB) functions.

Light Detection and Ranging (Lidar) sensors can measure the distance and reflectance of objects using Laser beams. They have a wide opening angle and a high angular precision.

Photonic Mixing Devices (PMD) are active range measurement sensors that deliver a quite detailed dense range image. Currently they are not widely spread among applications in favor of Radar, Lidar and Stereo Camera sensors.

Stereo Camera sensors are delivering a dense depth image with a high angular resolution but usually a lower depth resolution than radar and lidar sensors. Using two camera sensors and an extensive, meanwhile quite efficient processing, a depth image is produced similarly to the human stereoscopic 3D vision capability.

Knowledge Sources

Map data can be seen as sensor data with a high precision according to the sensor principle but with possible correctness uncertainties originating from the possible environment change between recording and using the data. It can be used to simplify environment perception and interpretation especially for static environment elements.

Vehicle-to-X (V2X) communication sensors are communication devices that can receive messages from other vehicles or infrastructure (e.g. traffic lights). This way perception tasks like position estimation of vehicles or traffic light state estimation can be simplified but also human-like communication like informing about driving plans (e.g. indicators, gestures) can be realized.

General Web Access can be seen as a sensor that can gather general information from the internet. This can especially include data about weather and traffic congestion.

2.2.2 Low Level Perception

In the here presented exemplary system architecture the low level perception is handled in functional modules that build up on each other. Every functional module focuses on one estimation task and can handle and fuse different sensor data sources. The actual ordering is quite common but can be changed depending on the application. One goal of this thesis is to overcome this ordering by an overall probabilistic approach with equally represented estimation tasks.

To get an impression of the different estimation tasks, these functional modules are described shortly:

The **Environment Condition Assessment** (ECA) estimates the general condition of the environment outside and inside the vehicle. This can include weather estimations like rain or snow but also information about passenger activities and personal condition. This is usually not dependent on other functional modules but is a good basis for basic decisions in other modules.

Ego Motion Estimation is a very basic estimation of the ego vehicle's motion. In contrast to the later ego vehicle localization only a relative motion of the vehicle is estimated over time. This enables basic fusion of environment perception over time.

Grid Mapping produces a model-free representation of the vehicle's surrounding, mainly including occupied and unoccupied areas but can be extended to any advanced world property. This is usually one of the lower modules that is often also used for safety related tasks like trajectory surveillance and emergency braking. A basic temporal fusion using the ego motion stabilizes the estimation.

Localization (also known as Ego Vehicle Localization) fulfills the task of localizing the ego vehicle on a fixed world coordinate system. Basic approaches simply accumulate the ego motion and ignore the increasing error. Others use raw sensor data to match on a prerecorded sensor-dependent map or localize on a generic geometric traffic map. A correct localization is a critical requirement for subsequent estimation tasks. Thus, a common approach is to fuse several localization methods.

Road Layout Estimation estimates the static environment mainly including the road with lanes and markings but also traffic signs, traffic lights and all static obstacles can be included here. Usually they are separated in task-specific modules like traffic light recognition and traffic sign recognition. This task can easily be supported or even replaced by predefined map data.

Object Tracking is the most challenging basic estimation task. Moving objects have to be detected and observed as good as possible. Due to their movement, uncertainties per measurement have much more influence on the overall estimation. Motion models have to be implemented to compensate these uncertainties. Usually constant velocity or constant acceleration models are used here, whereas advanced models considering interactions with static and dynamic obstacles can improve the estimation in the high level perception.

2.2.3 High Level Perception

The high level perception modules try to understand the data and build up a coherent environment model containing all knowledge that can be derived from sensory input. The environment model can afterwards be used for all decision and planning tasks. It can be represented as a model-free, usually grid based model or as a model-based model with geometric objects. In general high level perception modules include the following three tasks:

High-Level Fusion combines the results of all functional modules to one environment model. Spatial inconsistencies can be solved here.

Scene Understanding is the task of bringing objects into relation: Vehicles drive on roads, traffic signs belong to specific lanes and vehicles react to the road geometry or other vehicles. These relations build up an entity-relationship model and describe the meaning of the traffic scene in a machine-readable, often also human-readable, way. The module returns the most advanced interpretation to the currently present time slice in the environment model.

Situation Prediction adds a temporal interpretation to the currently present traffic scene. This enriches a traffic scene (with only little temporal interpretation) to a traffic situation (with more temporal context). The result is a temporal extension of the vehicle movements including possible geometric trajectories and discrete behaviors. In contrast to the later trajectory planning, not a single optimal trajectory but a probability distribution over all relevant possible trajectories is desired.

2.2.4 High Level Execution

In the high level execution layer all tasks are gathered that deliver long-term decisions.

A **Mission Control** decides about driving tasks. It is the most high-level decision module choosing and scheduling tasks like driving to a target location or parking for charging a battery. Also the usage of different task-specific components is triggered and their success is checked here. It is usually implemented as a state machine allowing deterministic behavior and manageable verification.

Behavior Planning handles the chain of needed behaviors to fulfill a driving task. It mainly reduces the large field of possible basic actions to a few ones that have been checked in a wider temporal context to bring the vehicle closer to the goal. The output is one or more possible maneuver chains that act as a guidance for trajectory planning.

Trajectory Planning plans a drivable geometric trajectory that is safe and comfortable for the next few seconds. In general this is an optimization problem considering hard constraints (like physical boundaries) and weak optimization criterias (like comfort parameters). The result is a geometric trajectory that considers the evolvement of the traffic situation sufficiently. Since the trajectory plan also influences the other traffic participants it has to be considered to couple this module more directly with the Situation Prediction.

2.2.5 Low Level Execution

After deciding about long term driving plans in the high level execution layer the task of the low level execution layer is to formalize these decisions into direct actuator inputs.

Controllers are used to send high frequency updates to the actuators according to the desired trajectory and the observed ego motion. The result is a vehicle behavior that executes the desired trajectory as close as possible.

Other actuators like gears, lights and windshield wiper are controlled similarly.

2.2.6 Actuators

Similarly to the sensors that observe the environment, the actuators influence the environment.

Steering and Acceleration actuators like steering wheel and gas and brake are the most relevant actuators for automated vehicles. These directly control the vehicle's lateral and longitudinal motion. They are implemented as direct input to the motor and steering system or as additional motors acting on the steering wheel and pedals what brings additional delays and actuator uncertainties.

Gears have to be chosen, at least for driving forward and backward in maneuver situations.

Visual and acoustic communication to passengers and other traffic participants can be undertaken via lights, displays and the horn.

V2X communication senders are necessary to explicitly communicate to other intelligent vehicles and infrastructure.

2.3 Large Estimation Problems

As already mentioned in Sec. 2.2, the whole estimation process in an intelligent agent underlies many uncertainties. Let X be the set of all attributes of the scene. Typically the attributes of interest $Y \subset X$ cannot be directly observed. This is often expressed as Y being a *hidden*, *unobservable*, *latent* or *intensional* variable. [2]

There are properties of the environment that can be observed by sensors, resulting in measurements $Z \subset X$, often called *observable* or *extensional* variables. These measurements underly sensor uncertainties that result from physical sensor principles. Additionally the measured attributes are often not directly the attributes of interest Y but are correlated to Y. If this correlation is a geometric, logical or physical dependency (e.g. wheel is mounted on car) the dependency can be described with relatively high certainty, while a rather psychological dependency (traffic participant is reacting on movement of others) introduces further uncertainties.

The general estimation problem can be described as estimating the state of the hidden variable Y given the observable measurements Z:

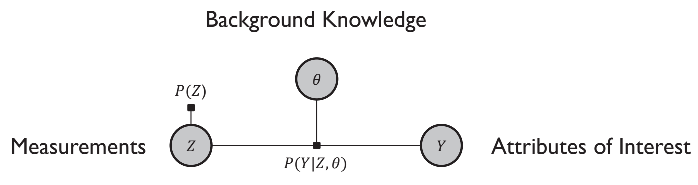
$$Y \sim P(Y|Z) \tag{2.1}$$

[2] The letters X, Y and Z are differently used in different existing research. In filtering works X is often used for hidden variables and Z for observable ones [112] [35] [40], but not always, e.g. [28] [123]. Since in this work a more general perspective is desired, X is used as the set of all variables, Y for hidden ones, Z for observable, and θ for parameters, resulting in $X = Y \cup Z \cup \theta$.

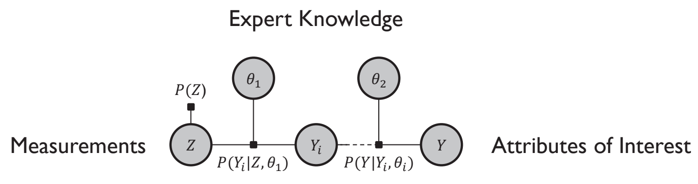
To solve this problem a proper model for $P(Y|Z) = P(Y|Z,\theta)$ with model parameters θ has to be found. This is a challenging task: Considering the problem of autonomous driving Z includes all attributes of all sensor measurements, e.g. several cameras, lidars, radars, that can easily be a multiple when considering several automated vehicles and V2X communication. Y includes all environment objects, properties of them and hidden variables like driving plans of traffic participants. There are different states of Y and Z at every different time. Thus, the direct formulation of $P(Y|Z,\theta)$ is unfeasible.

There are two approaches for handling this challenge (Fig. 2.4):

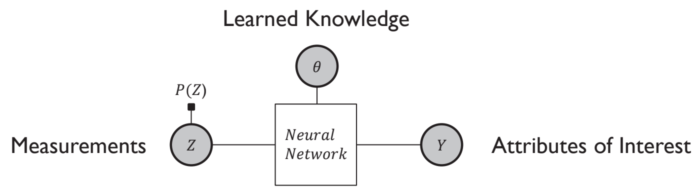
1. The overall problem can be factorized into smaller subproblems that can then be parametrized using expert knowledge related to a specific subtopic.
2. Machine learning can be used to parametrize a very large generic model using (labeled) observations from experienced situations.



(a) Problem formulation



(b) Approach by factorization



(c) Approach by machine learning

Figure 2.4: The two approaches b), c) of handling a complex overall estimation problem a). Adapted from [3].

As argued in [3] both approaches have advantages and disadvantages:

Factorizing into subproblems has the advantage of having intermediate representations (sensor features, object lists ...) which help evaluating and analyzing system failures. The subproblems can be parametrized more easily by human experts. The drawback is that a lot of effort has to be undertaken to define the factorization and the model parametrizations. Special cases have to be considered during modeling and the system is less adaptive.

Extensive machine learning on the other side allows very automatic parametrization of the model. Little to no expert knowledge is required. Associations can be incorporated that would not be possible using expert knowledge. A drawback is that safety and stability cannot be guaranteed and it is hard to get insights into the internals of large machine learned models such as deep neural networks.

The goal is to find a new modeling language that allows the combination of both approaches in a defined, easily manageable environment. From another perspective the same language allows to describe the common modular approach (Sec. 2.2) in a holistic way based on probability theory becoming a formalized factorization of the automated driving problem. One of the elementary bases for describing factorizations is the existing work on probabilistic grapical models.

2.4 Dependency Formalization by Probabilistic Graphical Models

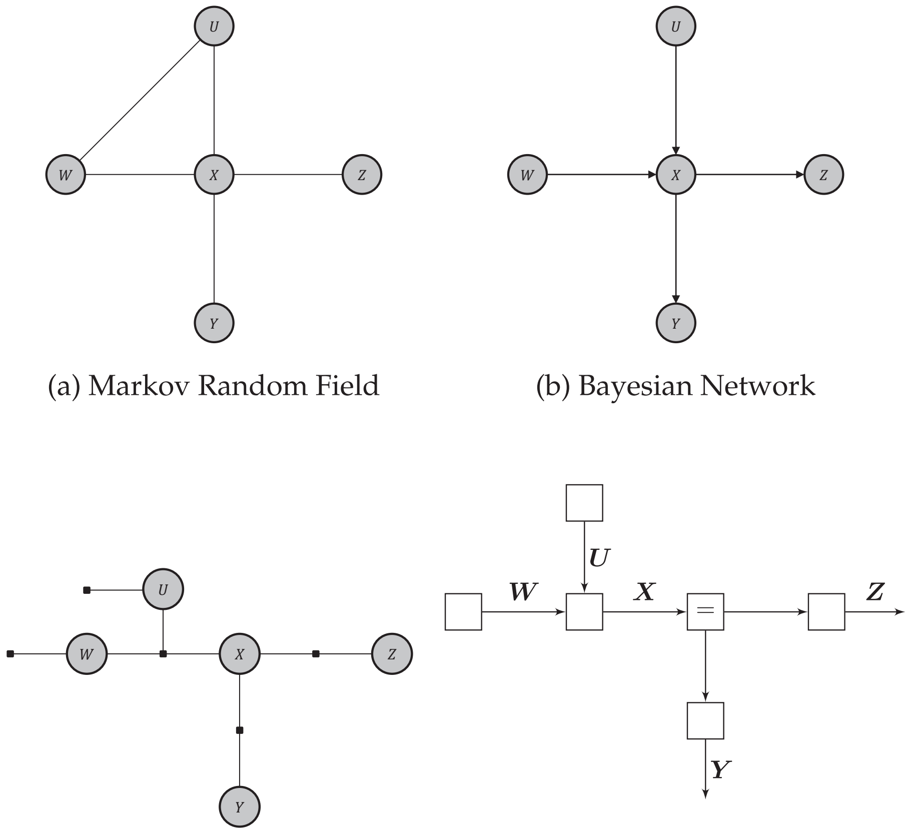
The goal of probabilistic graphical models is to formalize the dependencies and independencies between aspects (variables) of estimation problems. In general this is solved by using a graph structure consisting of nodes and edges (Fig. 2.5).

The nodes (usually) represent variables whereas the edges represent dependencies. The less edges are available the less dependencies exist between variables. This reduces model complexity and thus increases inference speed. The edges in the graph do not have to completely represent the real world dependencies. Instead a trade-off has to be made between reducing the graph structure using independence assumptions and representing the real world possible dependencies as close as possible. Also relevant dependencies can automatically be estimated by machine learning, so called graph structure learning.

The graph describes a factorization of the global distribution over all variables X, the so called *joint probabilistic distribution* $P(X)$, e.g.

$$P(X) \propto \prod f(X_i) \tag{2.2}$$

where X_i is a subset of the variables in the graph and the local functions $f(X_i)$ represent the dependencies modeled in the graph. There are different approaches how the dependencies are exactly modeled which will be described in the following sections. Some of them do not ensure that the product of the local functions is normalized to a correct probability distribution. Since often the exact probability



(a) Markov Random Field (b) Bayesian Network

(c) Bipartite factor graph (d) Forney-style factor graph (FFG)

Figure 2.5: Four different graph representations. Adapted from [86].

is not of interest, the normalization can be neglected what is expressed by usage of the $\propto$ symbol.

More information about probabilistic graphical models in general can be found in [95], [96], [75], [28] and [123]. A brief review of graph theory in general is given in [123].

2.4.1 Correlations: Markov Random Fields

Markov Random Fields (also known as *Markov Networks* or simply *Undirected Graphical Models*) base on an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes (vertices) $\mathcal{V}$ and undirected edges $\mathcal{E}$. Every node $i \in \mathcal{V}$ represents a probabilistic variable X_i and every edge $e \in \mathcal{E}$ represents a (undirected) correlation between two variables (Fig. 2.5a). The dependencies are formulated by non-negative potential functions ψ defined on the cliques[3] of $\mathcal{G}$. The joint potential function of the Markov Random Field given a graph $\mathcal{G}$ and a set of cliques $\mathcal{C}$ can be factorized over the cliques:

$$\psi(X) = \prod_{c \in \mathcal{C}} \psi_c(X_c) \tag{2.3}$$

This joint potential function is not normalized in general, thus it has to be normalized to be interpreted as a joint probability distribution:

$$P(X) = \frac{1}{Z} \psi(X) \tag{2.4}$$

with

$$Z = \sum_{X} \prod_{c \in \mathcal{C}} \psi_c(X_c) \tag{2.5}$$

The normalization function is hard to calculate and one of the drawbacks of Markov Random Fields. However, for most evaluations the normalization can be neglected and thus:

$$P(X) \propto \prod_{c \in \mathcal{C}} \psi_c(X_c) \tag{2.6}$$

The potential functions model correlations between the involved random variables. There is no conditional dependency describing that one variable is caused by another variable. On the one hand, this allows very generic formalization of correlations, on the other hand, causal dependencies (and independencies) cannot be modeled.

[3]A clique is a fully connected subset of nodes for which all members are neighbours [121]

2.4.2 Causalities: Bayesian Networks

A Bayesian Network (also known as belief network, generative model, causal model or simply *directed graphical model*) is a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes (vertices) $\mathcal{V}$ and directed edges $\mathcal{E}$. Every node $i \in \mathcal{V}$ represents a probabilistic variable X_i and every edge $e \in \mathcal{E}$ represents a (directed) causality between two variables (Fig. 2.5b). Because of these causalities no cycles are allowed in the directed graph.

The dependencies are formulated as local conditional probability distributions for each variable given its parents[4]:

$$P(X_i|X_{\Gamma(i)}) \tag{2.7}$$

The joint probability distribution of a Bayesian Network can be described by a factorization using the local conditional probability distributions:

$$P(X) = \prod_{i \in \mathcal{V}} P(X_i|X_{\Gamma(i)}) \tag{2.8}$$

In contrast to Markov Random Fields this joint probability distribution is already normalized since all the local functions are normalized conditional probability distributions.

The local conditional probability distributions describe causalities among small subsets of the variables. These causalities can easily be parametrized by a human expert. Additionally, the usage of conditional probability distributions allows detailed independence analysis using techniques like d-separation, see [28].

2.4.3 Credibility: Dempster Shafer Evidence Theory

Besides modeling correlations (Markow Random Fields) and causalities (Bayesian Networks), the consideration of credibility can be seen as a third way of modeling dependencies. Although *Dempster Shafer Evidence Theory* (also known as *belief theory*) is not directly connected to probabilistic graphical models, it is straight forward to include it here as another way of describing dependencies.

Assume Ω representing the set of possible outcomes of a discrete random variable X with $a_i \in \Omega$ being discrete, disjoint elements. While classic Bayesian probability theory assigns one probability measure to every event a_i ($P(a_i) \in [0, 1]$), evidence theory uses two measures, *belief* $Bel(a_i)$ and *plausibility* $Pl(a_i)$, with $Bel(a_i) \leq P(a_i) \leq Pl(a_i)$. The region between $Bel(a_i)$ and $Pl(a_i)$ can be seen as additional

[4]The set of parents $\Gamma(j)$ of a node $j \in \mathcal{V}$ is given by all nodes $i \in \mathcal{V}$ that are connected to j with an edge leading to j: $\Gamma(j) := \{i \in \mathcal{V}|(i, j) \in \mathcal{E}\}$

uncertainty measure. If there is none of this uncertainty, $Bel(a_i) = P(a_i) = Pl(a_i)$ and evidence theory is reduced to classic probability theory.

Every information source is represented by a basic belief assignment (BBA), that maps the power set 2^Ω to the interval $[0, 1]$:

$$m : 2^\Omega \to [0, 1] \tag{2.9}$$

It has to fulfill the conditions

$$m(\emptyset) = 0 \tag{2.10}$$

$$\text{and} \sum_{A \in 2^\Omega} m(A) = 1. \tag{2.11}$$

Belief and plausibility can then be derived by these equations:

$$Bel(A) := \sum_{A_K \in A} m(A_K) \tag{2.12}$$

$$Pl(A) := \sum_{A \cap A_K \neq \emptyset} m(A_K) \tag{2.13}$$

Two BBAs can be combined using the Dempster Rule of combination, that is equivalent to the Bayes Rule for binary BBAs. This allows handling of evidence theory based probabilities similarly to Bayesian Networks.

Additionally, a so called *pignistic probability transform* allows the transformation of evidence theory output to probability theory and works exist that formulate an *inverse pignistic probability transform* to achieve the opposite direction [122]. Because evidence theory needs more computation efforts than classic probability, it is usually only applied to small estimation problems not comparable to the size of Bayesian networks. But the ability to transform between both approaches allows integrating evidence theory in larger Bayesian networks. Furthermore, it was shown that evidence theory can be mapped to a Bayesian networks architecture [120].

2.4.4 Factor Graphs

A Factor Graph is a bipartite graph $\mathcal{H} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ with two types of nodes (vertices), so called variable nodws $\mathcal{V}$ and factor nodes $\mathcal{F}$ and edges $\mathcal{E}$ connecting variables to factors (Fig. 2.5c). Every node $i \in \mathcal{V}$ represents a probabilistic variable X_i and every edge $e \in \mathcal{E}$ connects a variable to a factor $f_j \in \mathcal{F}$. Let

$X_{f_j} := \{X_i | (X_i, f_j) \in \mathcal{E}\}$ denote the corresponding set of random variables for each $f_j \in \mathcal{F}$.

Then a factor graph can be used to describe a joint distribution as a normalized product of local potential functions corresponding to the factors in the graph:

$$P(X) \propto \prod_{f \in \mathcal{F}} \psi_f(X_f) \tag{2.14}$$

Markov Random Fields (Sec. 2.4.1) are a special case of factor graphs where the factors $\psi_f(x_f)$ correspond to potential functions over the cliques [5]. Bayesian Networks (Sec. 2.4.2) are a special case where the factors are local conditional probability distributions. Because of this and the ability of describing Dempster Shafer Evidence Theory in Bayesian Networks (Sec. 2.4.3) factor graphs are a helpful generalization of all three. All inference methods on the previous probabilistic graphical models can be mapped to methods on factor graphs. In fact, generic inference libraries (e.g. libDAI [93], GTSAM [39], iSAM [68] and g2o [80]) use factor graphs as a network-independent implementation basis.

Additionally to the bipartite graph depiction, there exists also an advanced graph modeling method, so-called Forney-style factor graphs (FFGs) [86] [110] [77] (also called normal graphs [49]) (Fig. 2.5d). If probability distributions of the variables are approximated by single values (and thus neglecting all uncertainties) these graphs directly lead to signal processing diagrams [87]. Thus, the FFG representation emphasizes the connection between deterministic and probabilistic signal processing. More details about different state space approximations will be given in Sec. 2.5.1.

The FFG representation can be derived from the bipartite representation by

- using edges instead of nodes for variables (connecting max. 2 factors)
- and introducing equality-factors (and additional edges) where a variable was connected to more than two factors.

A detailed description of FFGs is given in [110]. There it is also mentioned that edges can be drawn with arrows. This allows expressing causalities like in Bayesian networks or highlighting the forward direction like in signal processing.

Although FFGs emphasize the compatibility with standard block diagrams and will also have some advantages for inference rules (Sec. 2.5.2), they have the drawback of introducing additional equality factors. Also in probabilistic processing the bipartite graph representation is more popular. In this work the bipartite graph representation is used for dependency modeling and the FFG representation is used when it comes to representations close to system modeling block diagrams (e.g. Sec. 2.6). It is important to keep in mind that both representations can be transformed into each other.

[5] Even the normalizing constant $1/Z$ can be interpreted as a factor over the empty set

2.5 Inference with Probabilistic Graphical Models

As seen in Sec. 2.4, all different dependencies can be modeled in a generic factor graph. Such a factor graph can be used for different tasks including the main inference and learning tasks introduced in Sec. 2.3. It is assumed that the random variables X in the graph can be split into the three sets observable variables Z, unobservable variables Y and parameters θ. Then the tasks can be formulated as follows:

- **Evaluation:** The probability p_x of a given configuration x of the joint probability distribution can be evaluated using the factorization [40]:
$$p_x = P(X = x) \tag{2.15}$$
- **Sampling:** Samples (configurations) $\hat{x}^{(i)}$ can be drawn from the joint probability distribution:
$$\hat{x}^{(i)} \sim P(X) \tag{2.16}$$
This can also be seen as using the graph for simulation [40].
- **Inference of posterior distribution:** The posterior marginal distribution of unobserved variables Y can be estimated given the observations Z [123]:
$$P(Y|Z, \theta) \tag{2.17}$$
- **Inference of most likely configuration:** Instead of the whole posterior distribution it can be enough to know the maximum a posterior (MAP) configuration $\hat{Y}$ [123] [40]:
$$\hat{Y} = \underset{Y}{\operatorname{argmax}}\, P(Y|Z, \theta) \tag{2.18}$$
- **Parameter Learning:** The parameters of the model can be learned given a (usually large) set of observations Z and hyperparameters λ [123]:
$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(\theta|Z, \lambda) \tag{2.19}$$

The calculation of the MAP estimates $\hat{Y}$ and $\hat{\theta}$ can be seen as solving an optimization problem [40]. There are many algorithms for solving the inference and learning tasks (such as Loopy Belief Propagation [79], Fractional Belief Propagation [131] and Generalized Belief Propagation [134]), that can be formulated on the generic factor graph since all dependencies are already formulated without defining the actual variable state spaces. For machine learning usually the technique of Expectation Maximization [36] is used. Although machine learning is a qualified research area on its own the actual techniques are based on inference methods and thus the focus in this thesis will be on inference in general. Evaluation and sampling techniques are used within some algorithms to increase performance on specific variable properties. Thus a short look is taken on state space representations (Sec. 2.5.1) before the message passing algorithm (Sec. 2.5.2) is introduced that is a generalized basis for many independently developed inference methods. Sec. 2.6 shows how well-known filter methods can be derived from this algorithm.

2.5.1 State Space Representations

The state space of real world attributes can either be discrete or continuous. **Discrete state spaces** can directly be handled by assigning a probability value to each discrete element. Assuming a discrete variable X_D has S values denoted by integers $1, \dots, S$, the probability distribution $P(X_D)$ is a mapping from these values to $[0, 1]$:

$$P(X_D) : \{1, \dots, S\} \rightarrow [0, 1] \tag{2.20}$$

The parameters of this mapping can be expressed as a vector p utilizing the value integer as index:

$$P(X_D) = \mathcal{D}_{[p]}(X_D) = p_{X_D} \tag{2.21}$$

$\mathcal{D}_{[p]}(X_D)$ is introduced as discrete probability distribution function with parameters p similarly to the normal distribution function $\mathcal{N}_{[\mu,\Sigma]}(X_C)$ later in this section. The parameter vector p is a probability table with S elements. Local potential functions can also be described by probability tables:

$$\psi_f(X_D, Y_D) : \{1, \dots, S_X\} \times \{1, \dots, S_Y\} \rightarrow [0, 1] \tag{2.22}$$

This results in $S_X * S_Y$ entries in the table or S_i^N entries for general potential functions with N variables. The table can be expressed in a matrix (or tensor) that allows efficient calculations during inference (Sec. 2.5.4). As long as the variable spaces consist of a reasonable number of discrete elements and the local potential functions include only a small number of variables, this is easily computable.

Continuous state spaces are not easily computable in general. A continuous function has to assign probability values to all the elements:

$$P(X_C) : \mathbb{R} \rightarrow [0, 1] \tag{2.23}$$

General continuous distributions cannot be described by a closed form function. They can be approximated using assumptions that either match the input data or neglect properties that do not have to be considered. Typical approximations can be classified in the following classes [86] [87]:

- **Single values** neglect all uncertainties and reduce the distribution to one single value $\hat{x}$ with probability 1:

  $$P(X_C) \sim \delta(X_C, \hat{x}) \tag{2.24}$$

  This reflects temporary or final "hard decisions" on the variable X_C.

- **Quantization** (also called histogram) uses a fixed grid of M quantization levels to collect probability values in bins:

  $$P(X_C) \sim \{\hat{x}^{(0)}, \dots, \hat{x}^{(M)}\} \to [0,1] \tag{2.25}$$

  Often the grid is equidistant with a fixed step size h, e.g. $\hat{x}^{(i)} = \hat{x}_0 + i * h$

- **Function value and derivative at a single point** is a linearized approximation around a single point $\hat{x}$ with function value $f_{\hat{x}}$ and derivative $f'_{\hat{x}}$:

  $$P(X_C) \sim f_{\hat{x}} + f'_{\hat{x}} \cdot (X_C - \hat{x}) \tag{2.26}$$

  This is only valid in the vicinity of $\hat{x}$. It is a good approximation for gradient methods where a walking direction from a current guess $\hat{x}$ is iteratively needed.

- **Gaussian distributions** (or Normal distributions) describe the probability distribution by n-dimensional mean vector μ and covariance matrix Σ:

  $$P(X_C) \sim \mathcal{N}_{[\mu,\Sigma]}(X_C) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} e^{\left(-\frac{1}{2}(X_C-\mu)^\top \Sigma^{-1}(X_C-\mu)\right)} \tag{2.27}$$

  For many unimodal distributions the Gaussian distribution is a sufficient approximation. As soon as the distributions become asymmetric or multimodal other approximations have to be used.

- **Lists of samples** (particles) approximate complex distributions by a weighted sum over M simpler distributions, e.g. single values $\hat{x}^{(i)}$:

  $$P(X_C) \sim \sum_{i=1}^{M} w^{(i)} \delta(X_C, \hat{x}^{(i)}) \tag{2.28}$$

  This approximation is used in the known particle filter approach. The usage of Gaussian distributions is known as Gaussian Mixtures:

  $$P(X_C) \sim \sum_{i=1}^{M} w^{(i)} \mathcal{N}_{[\hat{x}^{(i)}),\Sigma^{(i)}]}(X_C) \tag{2.29}$$

The approximations can also be classified into *parametric* and *nonparametric* [121]: Single values, Gaussian distributions and derivatives are parametric while quantization and lists of samples are nonparametric, although Gaussian mixtures have a parametric aspect.

Please note that the shown exemplary equations might be seen based on 1-dimensional spaces and it might be assumed that the given continuous variable X_C is 1-dimensional. All equations are also valid for n-dimensional multivariate distributions necessary for multi-dimensional X_C. In potential functions no difference exists between $\psi_f(X_C)$ with a multidimensional X_C and $\psi_f(X_1, \dots, X_N)$ if

$\dim X_C = \sum \dim X_i$. Actually this is the key idea behind factorization (Sec. 2.4): If subdimensions of a (large) random variable are independent, these can be split to separate random variables and some of the dependencies between these sub variables can be removed.

Additionally to the here shown representations, compound distributions have to be used to describe potential functions that integrate very different variables, especially discrete and continuous spaces.

2.5.2 Message Passing

While factor graphs are also useable for *evaluation* and *sampling* [40], their most powerful application is *inference* and *learning*. For solving these problems the structure of the underlying factor graph is of key importance and can be utilized in different algorithms. While generic graph structures including loops and a mixture of discrete and continuous variables are only solvable by approximate methods, tree-structured graphs can be solved by the message passing algorithm with linear complexity.

The message passing algorithm [90] (also known as sum-product algorithm [79] [86] or belief propagation [101] in Bayesian Networks) is also the basis for advanced algorithms solving graphs with loops. Therefore a more detailed look at the idea of the message passing algorithm follows.

Let $N(v) \subset \mathcal{F}$ be the set of neighbor factors connected to variable node $v \in \mathcal{V}$ and $N(f) \subset \mathcal{V}$ be the set of neighbor variables connected to factor $f \in \mathcal{F}$. Then the message passing algorithm consists of two message calculation instructions: One for edges from variable nodes to factor nodes:

$$m_{v \to f}(X) = \begin{cases} 1 & N(v) \setminus \{f\} = \emptyset \\ m_{f_2 \to v}(X) & N(v) \setminus \{f\} = \{f_2\} \\ \bigotimes\limits_{f' \in N(v) \setminus \{f\}} m_{f' \to v}(X) & else \end{cases} \tag{2.30}$$

And one for edges from factor nodes to variable nodes. The message from a factor $f = \{v, w_1, \dots, w_n\}$ to the variable v is the function

$$m_{f\to v}(X) = \begin{cases} \psi_f(X) & deg(f) = 1 \\ \bigoplus\limits_{w_1} \cdots \bigoplus\limits_{w_n} \left(\psi_f\left(X_v, \ldots, X_{w_n}\right) \bigotimes\limits_{w\in N(f)\setminus\{v\}} m_{w\to f}(w) \right) & else \end{cases} \tag{2.31}$$

If all messages coming from all neighbor factors of a variable are given, the marginal distribution of this variable can be calculated, via

$$Bel(v) = \bigotimes_{f\in N(v)} m_{f\to v}(v) \tag{2.32}$$

It was shown that message calculation is an efficient intermediate calculation step for marginal calculations [79]: If multiple variables' marginals have to be calculated, the messages are the calculations that would have been necessary multiple times otherwise. This can also be seen as a consequent utilization of the distributive law [130] [19].

In the case of Forney-style factor graphs (FFGs) (Sec. 2.4.4) with variables connecting max. 2 factors, the message passing equations (eq. 2.30 and eq. 2.31) can be simplified to a single rule: Neglecting the empty set case, eq. 2.30 simplifies to

$$m_{v\to f}(X) = m_{f_2\to v}(X) \tag{2.33}$$

and thus eq. 2.31 can directly be described as

$$m_{f\to v}(X) = \bigoplus_{w_1} \cdots \bigoplus_{w_n} \left(\psi_f\left(X_v, \ldots, X_{w_n}\right) \bigotimes_{w\in N(f)\setminus\{v\}} m_{f_w\to w}(w) \right) \tag{2.34}$$

where f_w is the factor that provides the message along edge w (corresponding to variable X_w) to f.

In both graph representations the implementation of the abstract operators $\bigotimes$ and $\bigoplus$ depend on the estimation task (Sec. 2.5) and the state space representations (Sec. 2.5.1).

The abstract product operator $\bigotimes$ usually corresponds to the product $\prod$. The abstract addition operator $\bigoplus$ corresponds to a sum over all discrete elements for discrete variables X_D and to an integral for continuous variables X_C. Using these operators, the result of equ. 2.32 equals to the marginal distribution. The resulting algorithms are called *sum-product algorithm* for discrete variables and *integral-product algorithm* for continuous variables.

If instead of the whole marginal distribution only the joint maximum of the distribution is searched, this can be achieved by replacing the abstract addition operator $\bigoplus$ by a $\max$ operator, leading to the *max-product algorithm* [28]. If the logarithmic domain is used, the abstract product operator $\bigotimes$ switches to the sum-operator $\sum$ and yields the so-called *max-sum (or min-sum) algorithm* [87].

2.5.3 Message Passing Schedules

The message passing algorithm is exact if all messages needed are welldefined. This is possible with a generic message passing schedule if the factor graph is tree structured: The algorithm can start by calculating the messages at the leaf variables and factors using the first cases in equ. 2.30 and equ. 2.31. Subsequently all other messages can be calculated until both messages are calculated per edge that connects a variable to a factor. It was shown that equ. 2.32 delivers the exact result then, and does this in linear complexity [79].

If a factor graph is not tree-structured there are several options. First of all, additional independence assumptions can be made or nodes can be "clustered"[6] differently [79] [35] to achieve a tree structure and being able to apply the exact message passing algorithm again. If this is not possible or wanted, the message passing algorithm can be started on a factor graph with initial message values and run iteratively until it converges. It was shown [97] [89] that it converges with high probability and leads to an approximate solution of the searched marginal distribution (or MAP). This is also known as loopy belief propagation.

If prior detailed knowledge about variables' functionalities and importance is available, a middle course between loopy optimization and independence introduction can be taken: The factor graph can be defined with all dependencies but for inference a *message passing schedule* is defined that takes the application's specialties into account, for example by inferring only from past states to future states, as largely utilized in [62].

Another simplification depends on the state space approximations. If the variables are approximated by a list of samples (particles), the messages consist of a sum of all single particle messages (comp. eq. 2.28). The accuracy and inference effort depend on the number of particles. A straight forward inference method is to follow a central message passing schedule and calculate every message by iterating over all particles, leading to a full distribution approximation per variable (breadth-first message passing). Depending on the application it can be worth selecting a node with high certainty (e.g a lane estimation close to the sensor in lane detection application), selecting one sample of that distribution and passing this single message through the whole graph to get one of the most probable solutions without estimating all variables' distributions in detail (depth-first message passing) [124] (Fig. 2.6).

[6] *Clustering* means combining variables in bigger variables which is the inverse of *factorization* explained in Sec. 2.3.

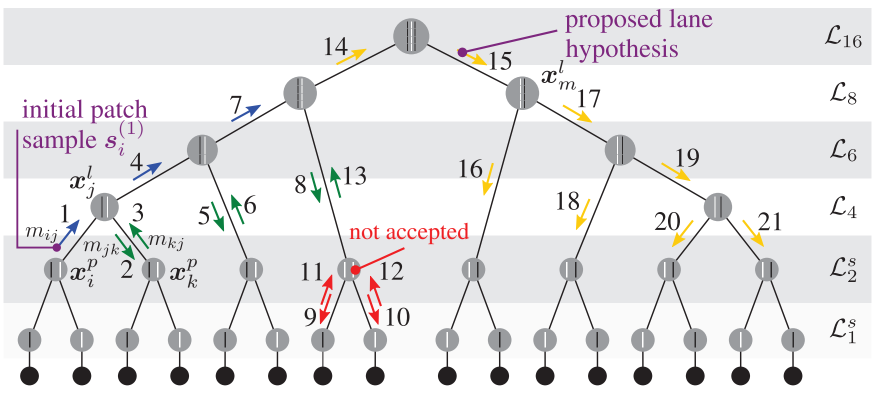


Figure 2.6: Depth-first message passing in a lane detection application reducing much of the computation effort. [124]

2.5.4 Message Approximations

Having discussed about the generic message passing rule and the message passing schedules, a closer look into the actual calculations has to be taken.

While most implementations of $\bigoplus$ including sum and min/max operators are calculable, the integral version considering continuous variables is often not calculable. This does not have to origin from a hardly modelable probability distribution of the input variables but can also origin from the factor function ψ_f itself.

The state space representations of Sec. 2.5.1 can also be used to approximate the messages, resulting in simplified message update rules of eq. 2.31. Although discrete variables are not affected by the integral-product rule complexity they can also be simplified to matrix/vector calculations. Available message representations for discrete and continuous variables are:

- **Discrete values**:
  message: vector of element probabilities p
  sum-product rule:
  If a local potential function on a factor f is applied to the messages, the resulting discrete probability distribution can again be expressed as a vector:

  $$m_{f\to v}(X_v) \propto \mathcal{D}_{[p_f]}(X_v) \tag{2.35}$$

  p_f can be calculated by update rules utilizing the matrix formulation of the potential function in f (Table 2.1).

- **Single values**:
  message: single value $\hat{x}$
  integral-product rule approximation:

  $$m_{f\to v}(X_v) \propto \psi_f(X_v, \hat{x}_1, \ldots, \hat{x}_N) \tag{2.36}$$

- **Quantization**:
  message: set of values $\{p_i\}_M$ corresponding to quantization levels $\{\hat{x}^{(i)}\}_M$
  integral-product rule approximation:

$$m_{f\to v}(X_v) \propto \sum_{i_1} \cdots \sum_{i_N} \left(\psi_f(X_v, \hat{x}_1^{(i_1)}, \ldots, \hat{x}_N^{(i_N)}) \cdot m_{X_1\to f}(\hat{x}_1^{(i_1)}) \cdots m_{X_N\to f}(\hat{x}_N^{(i_N)}) \right) \tag{2.37}$$

- **Function value and derivative at a single point**:
  This message approximation is mainly used for parameter estimation using e.g. expectation maximization, depending largely on the properties of the factor function $\psi_{f,\theta}$ with parameters θ. This is discussed in detail in [37] and [35].

- **Gaussian distributions**:
  message: n-dimensional mean vector μ and covariance matrix Σ
  integral-product rule approximation:
  Gaussian distributions are beneficial with linear operations: If the factor function ψ_f is a linear operation and the input messages are Gaussian, the resulting message is a Gaussian again:

$$m_{f\to v}(X_v) \propto \mathcal{N}_{[\mu_f, \Sigma_f]}(X_v) \tag{2.38}$$

  μ_f and Σ_f can be calculated by update rules depending on the operation in f. An overview of many update rules is given in [77] and Table 2.2 and Table 2.3.

- **Lists of samples**:
  message: set of weights and corresponding sample parameters, e.g. single values $\{w^{(i)}, \hat{x}^{(i)}\}_M$
  integral-product rule approximation:

$$m_{f\to v}(X) \propto \sum_{i_1} \cdots \sum_{i_N} \left(\psi_f(X_v, \hat{x}_1^{(i_1)}, \ldots, \hat{x}_N^{(i_N)}) \cdot w_1^{(i_1)} \cdots w_N^{(i_N)} \right) \tag{2.39}$$

  Some exemplary update rules for sample-based messages are given in Table 2.4.

The different message approximations can be used on the same underlying factor graph. Once modeled dependencies can be approximated by different message approximations leading to different inference algorithms with different runtime and accuracy properties. Different message approximations can even be applied locally on the same factor graph. Therefore messages have to be converted from one representation to another. Additional variables and factors can be integrated that represent different state space approximations of the same variable and conversions between these new variables. The conversion factors can be seen as equality factors, that do not introduce additional uncertainties[7] but only convert messages to a different approximation type. Some exemplary conversion rules for variable composition and decomposition are depicted in Table 2.5.

In combination with message passing schedules it is even possible to infer the two messages over a single edge using two different approximations. This can especially be helpful if factor functions are not easily invertable. It results in a high flexibility for approximating dependencies on a per message basis.

It can be summarized that probabilistic graphical models can be used to describe real world dependencies (and independencies) as well as possible approximations up to signal processing block diagrams. Generic factor graphs can be modified (stretched and clustered) to emphasize independence assumptions and message passing schedules. Also established inference methods like Kalman or Particle Filters can be described using the probabilistic graphical methods as discussed in Sec. 2.6.

[7]besides the uncertainties induced by state space approximations

Table 2.1: Message update rules for vector-based discrete messages.

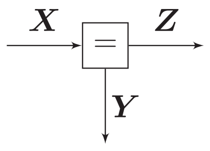
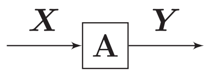
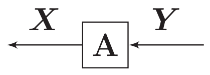
Node	**Update rule**	
	$\boldsymbol{z} = \boldsymbol{x} \times \boldsymbol{y}$	(2.40)
	$\boldsymbol{y} = \mathrm{A}^{\mathsf{T}} \boldsymbol{x}$	(2.41)
	$\boldsymbol{y} = \mathrm{A} \boldsymbol{x}$	(2.42)

Table 2.2: Message update rules for gaussian distribution based continuous messages. Simplified version of [85] and [77]. The rules that make use of A^{-1} are only valid if A is invertible.

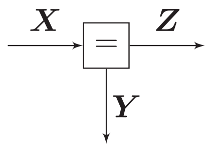
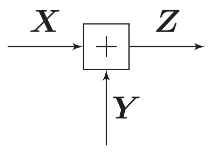
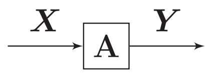
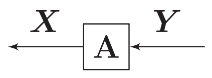
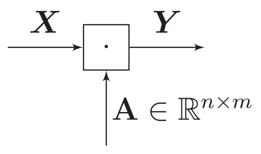
Node	**Update rule**
	$\mu_Z = (\Sigma_X^{-1} + \Sigma_Y^{-1})^{\mathsf{T}}(\Sigma_X^{-1}\mu_X + \Sigma_Y^{-1}\mu_Y)$ (2.43) $\Sigma_Z = \Sigma_X(\Sigma_X + \Sigma_Y)^{\mathsf{T}}\Sigma_Y$ (2.44)
	$\mu_Z = \mu_X + \mu_Y$ (2.45) $\Sigma_Z = \Sigma_X + \Sigma_Y$ (2.46)
	$\mu_Y = A\mu_X$ (2.47) $\Sigma_Y = A\Sigma_X A^H$ (2.48)
	$\mu_X = A^{-1}\mu_Y$ (2.49) $\Sigma_X = A^{-1}\Sigma_Y A^{-\mathsf{T}}$ (2.50)
	$\mu_Y = A\mu_X$ (2.51) $\Sigma_Y = A\Sigma_X A^H$ (2.52)

Table 2.3: Second Part: Message update rules for gaussian distribution based continuous messages. Simplified version of [85] and [77]. The rules that make use of A^{-1} are only valid if A is invertible.

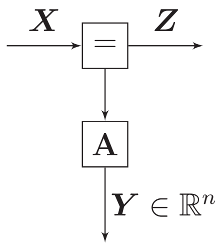
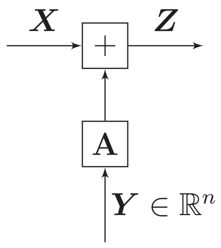
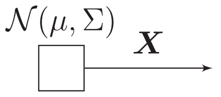
Node	Update rule
	$\mu_Z = \mu_X + \Sigma_X A^H G(\mu_Y - A\mu_X)$ (2.53) $\Sigma_Z = \Sigma_X - \Sigma_X A^H G A \Sigma_X$ (2.54) with $G := (A\Sigma_X A^H + \Sigma_Y)^{-1}$ (2.55)
	$\mu_Z = \mu_X + A\mu_Y$ (2.56) $\Sigma_Z = \Sigma_X - A\Sigma_Y A^{\mathsf{T}}$ (2.57)
	$\mu_X = \mu$ (2.58) $\Sigma_X = \Sigma$ (2.59)

Table 2.4: Message update rules for sample-based continuous messages. Adapted from [77].

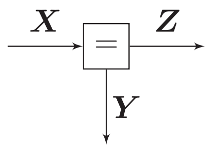
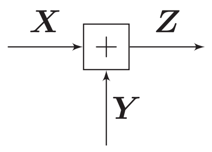
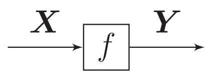
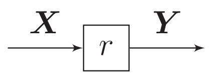
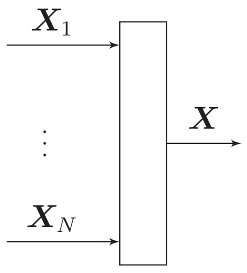
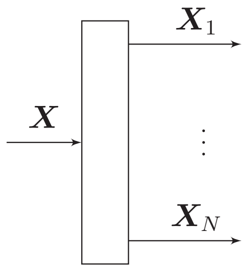
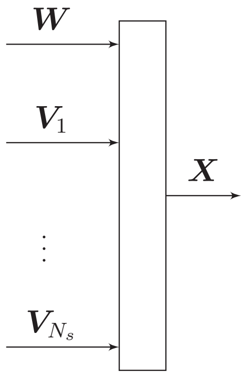
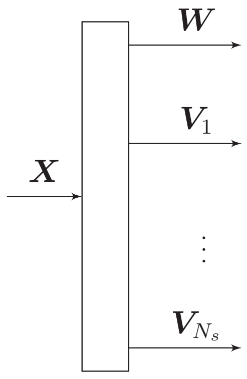
Node	**Update rule**
	$\hat{z}^{(i)} = \hat{x}^{(i)}$ (2.60) $w_Z^{(i)} = w_X^{(i)} m_Y(\hat{z}^{(i)})$ (2.61) where $m_Y(\hat{z}^{(i)})$ is the message from Y evaluated at $\hat{z}^{(i)}$
	$\hat{z}^{(i)} = \hat{x}^{(i)} + \hat{y}$ (2.62) $w_Z^{(i)} = w_X^{(i)}$ (2.63) with $\hat{y}$ being a sampled value of the message from Y
	$\hat{y}^{(i)} = f(\hat{x}^{(i)})$ (2.64) $w_Y^{(i)} = w_X^{(i)}$ (2.65)
 resample	$\hat{y}^{(i)} = \hat{x}$ (2.66) $w_Y^{(i)} = 1$ (2.67) with $\hat{x}$ being a sampled value of the message $m_X(X) = \sum_{i=1}^{M} w_X^{(i)} \delta(X, \hat{x}^{(i)})$

Table 2.5: Some exemplary message update rules for converting between message types. m_X is the message at the edge connecting the factor to variable X.

Node	Update rule
 $\boldsymbol{X}_1$ $\vdots$ $\boldsymbol{X}_N$ $\boldsymbol{X}$ composition	$X := (X_1, \dots, X_N)$ (2.68) $m_X = (m_{X_1}, \dots, m_{X_N})$ (2.69)
 $\boldsymbol{X}$ $\boldsymbol{X}_1$ $\vdots$ $\boldsymbol{X}_N$ decomposition	$X := (X_1, \dots, X_N)$ (2.70) $m_{X_i} = (m_X)_i$ (2.71)
 $\boldsymbol{W}$ $\boldsymbol{V}_1$ $\vdots$ $\boldsymbol{V}_{N_s}$ $\boldsymbol{X}$ gaussian mixture composition	$m_X := \{w^{(i)}, \hat{x}^{(i)}, \Sigma^{(i)}\}_{N_s}$ (2.72) $m_{V_i} := (\hat{v}_i, \Sigma_{V_i})$ (2.73) $m_W \in [0,1]^{N_s}$ (2.74) $m_X = \{w_i, \hat{v}_i, \Sigma_{V_i}\}_{N_s}$ (2.75)
 $\boldsymbol{X}$ $\boldsymbol{W}$ $\boldsymbol{V}_1$ $\vdots$ $\boldsymbol{V}_{N_s}$ gaussian mixture decomposition	$m_X := \{w^{(i)}, \hat{x}^{(i)}, \Sigma^{(i)}\}_{N_s}$ (2.76) $m_{V_i} := (\hat{v}_i, \Sigma_{V_i})$ (2.77) $m_W \in [0,1]^{N_s}$ (2.78) $m_{V_i} = (\hat{x}^{(i)}, \Sigma^{(i)})$ (2.79) $m_W = (w^{(1)}, \dots, w^{(N_s)})$ (2.80)

2.6 Filter Methods as Graphical Models

Reconcile that as described in Sec. 2.3 the general estimation problem can be formulated as estimating the state of the hidden variable Y given the observable measurements Z (eq. 2.1). One of the reasons why this can hardly be modeled is because of temporal dependencies. Y and Z include all observations and states over all time. Everything could be dependent on everything. This is not manageable in general, especially for real time applications. This section focuses on known filtering methods that tackle the problem of temporal dependencies by a suitable factorization of the joint probability distribution $P(X)$.

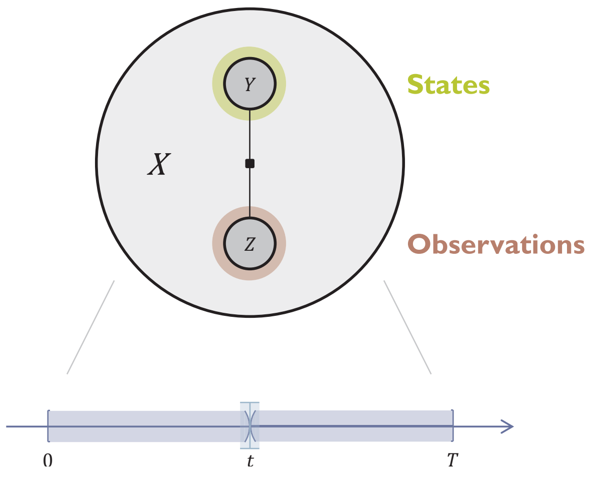


Figure 2.7: All attributes X consist of observations Z and hidden states Y. They can be split into different parts in time.

Assuming a variable X contains values for all continuous time in $[0, T]$ the variable can be split into three parts around one point t in time (Fig. 2.7):

$$X = X_{[0,t)} \cup X_t \cup X_{(t,T]} \tag{2.81}$$

Applying this separation to observations Z and hidden variables Y, the inference tasks that can be solved using general Probabilistic Graphical Models (Sec. 2.5, eq. 2.17 and 2.18) can be detailed considering temporal problems (Fig. 2.8):

- **Filtering:** The current hidden state Y_t given all observations to date $Z_{[0,t]}$ can be estimated:

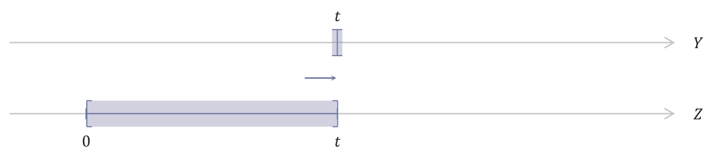
$$P(Y_t | Z_{[0,t]}) \tag{2.82}$$

- **Prediction:** Future states Y_{t+k} (and observations) can be predicted given all past observations $Z_{[0,t]}$:

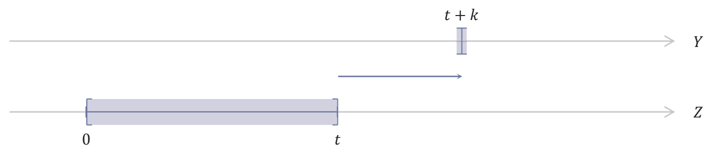
$$P(Y_{t+k} | Z_{[0,t]}) \qquad k > 0 \tag{2.83}$$

- **Smoothing:** Given observations over a long period $Z_{[0,T]}$ a state within that period Y_k can be estimated more precisely than when filtering from earlier observations only:
$$P(Y_k|Z_{[0,T]}) \qquad 0 \leq k \leq T \tag{2.84}$$

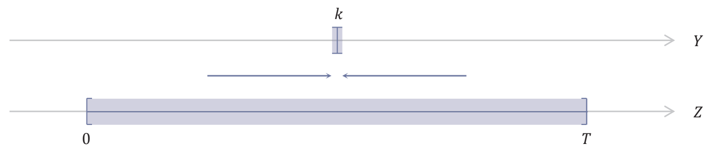
- **Most likely explanation:** Given an interval of observations $Z_{[t_1,t_2]}$ a sequence of states $\hat{Y}_{[t_1,t_2]}$ can be estimated that is most likely to have generated these observations:
$$\underset{Y_{[t_1,t_2]}}{\operatorname{argmax}} P(Y_{[t_1,t_2]}|Z_{[t_1,t_2]}) \tag{2.85}$$



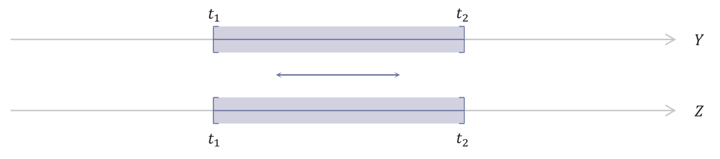
(a) Filtering



(b) Prediction



(c) Smoothing



(d) Most likely explanation

Figure 2.8: The inference tasks *Filtering*, *Prediction*, *Smoothing* and *Most Likely Explanation*. The arrows symbolize the information flow. Adapted from [77].

The tasks parameter learning, sampling and evaluation, that can be applied to general Probabilistic Graphical Models can of course also be applied to temporal models and are a great help here as well.

Now first the theory of state space models (Sec. 2.6.1) is introduced and then different well-known filters are analyzed. Since research around filter modeling methods is a wide field focus will be on a representative subset and especially their relation to factor graphs. For detailed information on the filter principles see the respective referenced works. An overview over many other filters is given in [118].

2.6.1 State Space Models for Sequential Estimation Problems

Although there exist works on modeling continuous time [30] [66], usually discrete time is assumed, meaning that observations Y_t only arrive at discrete, often equidistant time steps.

A state space model [28] [95] (also known as Bayesian Filter [113] and Dynamic Bayesian Network[8] [95], [38], [112], [96], [56], [75]) assumes that the observed data Z is sequential (e.g. $Z = (Z_0, \dots, Z_T)$) and the hidden states Y describe a stochastic process (e.g. $Y = (Y_0, \dots, Y_T)$). From now on t is an integer to realize the discrete time handling. It describes one step in a sequence that can be a temporal sequence like in a dynamic process or a logic sequence like in DNA strands or characters in words. The group of variables X corresponding to the discrete sequence steps 0 to t is denoted by $X_{0:t}$.

It can be assumed that the problem is first-order Markov, meaning the current state only depends on the preceding state: $P(Y_t|Y_{0:t-1}) = P(Y_t|Y_{t-1})$. This sounds like a strong limitation but larger sequential dependencies can be compressed in the current state, for example by setting $\tilde{Y}_t = (Y_t, Y_{t-1})$ [95] or by calculating higher order values like a velocity or a meaningful behavior from two (or more) position states. It is also assumed that all models are the same for all time (time-invariant or homogeneous). This is also a minor limitation since the models can have parameters to adjust their behavior. These parameters can be modeled as part of the state variable Y_t. This is used for example in the Interacting Multiple Model filter (IMM filter) (Sec. 2.6.5).

All models that have to be defined for a state space model are

- the prior $P(Y_0)$,
- a state transition (or process) function $P(Y_t|Y_{t-1})$
- and an observation (or measurement) function $P(Z_t|Y_t)$.[9]

[8]The term *Dynamic Bayesian Network* is not used since the term *dynamic* is misleading. As already stated in [95] the term *dynamic* is meant to highlight that a dynamic system is modeled not that the network changes over time. Since in this work also dynamically changing probabilistic graphical models are considered, the term *dynamic* is not used in such a central way.

[9]$P(Z_t|Y_t)$ describes how observations are "generated" from state variables, making the state space model a "generative model".

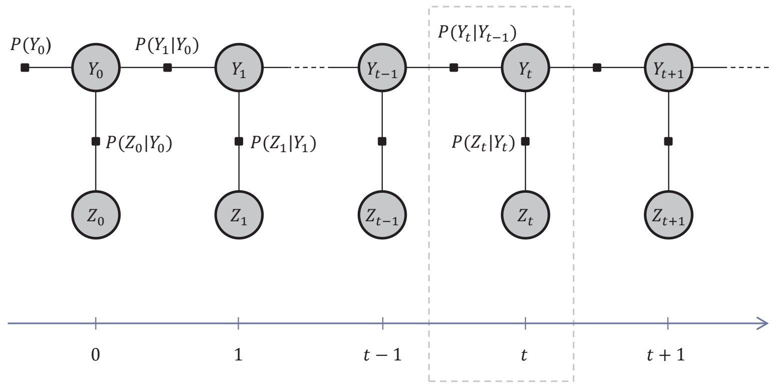


Figure 2.9: State Space Model as a Factor Graph. Adapted from [28]

The joint probability distribution of the whole model can be factorized using these functions (Fig. 2.9):

$$P(Z_0, \ldots, Z_T, Y_0, \ldots, Y_T) = P(Y_0) \left(\prod_{t=1}^{T} P(Y_t|Y_{t-1}) \right) \prod_{t=0}^{T} P(Z_t|Y_t) \tag{2.86}$$

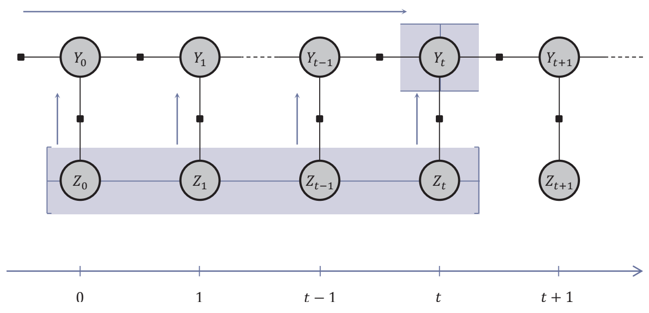
Note that the observations and states can consist of several (discrete and continuous) sub variables $Y_t^1, \ldots Y_t^i$ and $Z_t^1, \ldots Z_t^j$ and thus the probability functions can also be factorized to several sub models describing dependencies (and independencies) between these sub variables. This leads to a larger probabilistic graphical model describing the dependencies inside one time slice and between two time slices.

The estimation tasks introduced in Fig. 2.8 can be described using the factor graph representation of the state space model as depicted in Fig. 2.10. Depending on the given state spaces, the chosen approximations and the complexity of the probabilistic graphical model, several popular traditional Bayesian Filters can be derived (Sec. 2.6.2 - Sec. 2.6.6).

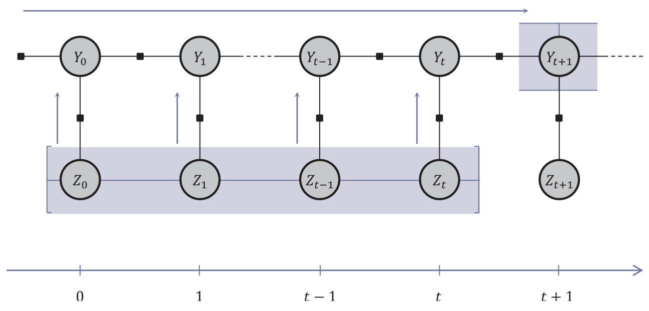
2.6.2 Hidden Markov Model

A Hidden Markov Model (HMM) [28] [112] [95] (also known as Discrete Bayes Filter [118]) is a state space model using discrete probability distributions for the hidden states Y. A typical situation is that the observations Z have a higher dimensionality. They are often modeled as a discrete distribution as well but also continuous distributions, e.g. approximated as Gaussian or Gaussian mixtures, are possible.

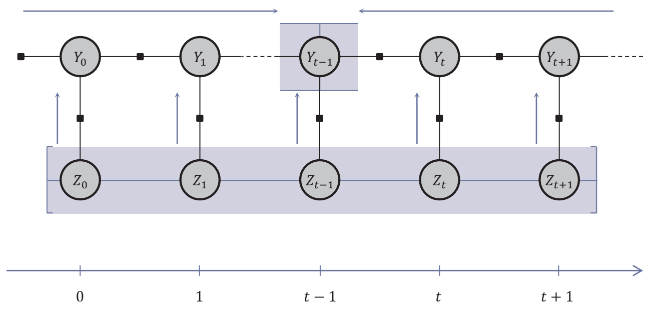
Let Y and Z be discrete. Then all messages can be expressed using the vector-based formulation (Sec. 2.5.4) and matrix-based potential functions. The transition model can be expressed by a probability table matrix T, the observation



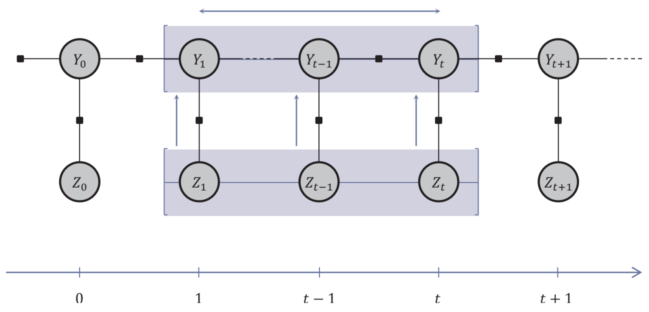
(a) Filtering



(b) Prediction



(c) Smoothing



(d) Most likely explanation

Figure 2.10: The inference tasks *Filtering*, *Prediction*, *Smoothing* and *Most Likely Explanation* depicted using the factor graph represenation of a state space model. The arrows symbolize the information flow in the factor graph.

model by O and the prior by a vector π, meaning

$$P(y_0) = \pi_{y_0} \tag{2.87}$$
$$P(y_t|y_{t-1}) = T_{y_{t-1},y_t} \tag{2.88}$$
$$P(z_t|y_t) = O_{y_t,z_t} \tag{2.89}$$

Following the update rules from Table 2.1 the forward message f and backward message b can compactly be written as

$$\boldsymbol{f}_{0:t} \propto (O\boldsymbol{z}_t) \times (T^\top \boldsymbol{f}_{0:t-1}) \tag{2.90}$$
$$\text{and} \quad \boldsymbol{b}_{t+1:T} \propto T\left((O\boldsymbol{z}_{t+1}) \times \boldsymbol{b}_{t+2:T}\right) \tag{2.91}$$

These messages can be used to perform filtering, smoothing and infering the most likely explanation. For prediction, the observation terms $(O\boldsymbol{z}_t)$ have to be omitted.

2.6.3 Kalman Filter

A Kalman Filter [69] [77] (also called Linear State Space Model [110] [135], Linear Dynamical System [28], Kalman Filter Model [95]) is a state space model using Gaussian message approximations and linear transition and observation functions. There are many extensions to the basic Kalman Filter, e.g. Extended Kalman Filter or Unscented Kalman Filter. This section will focus on the basic Kalman Filter to show the principle. For an overview about different advanced filters see [118].

For Kalman Filters the prior, transition function and observation function can be described using the Gaussian distributions

$$P(y_0) = \mathcal{N}_{[\mu_0,\Sigma_0]}(y_0) \tag{2.92}$$
$$P(y_t|y_{t-1}) = \mathcal{N}_{[Fy_{t-1},\Sigma_F]}(y_t) \tag{2.93}$$
$$P(z_t|y_t) = \mathcal{N}_{[Hy_t,\Sigma_H]}(z_t) \tag{2.94}$$

where F and H are transformation matrices describing the linear transition and observation functions and Σ_F and Σ_H describe the uncertainty (noise) in these processes. This is equivalent to the traditional formulation of noisy linear equations given by

$$y_0 = \mu_0 + u \tag{2.95}$$
$$y_t = Fy_{t-1} + w_t \tag{2.96}$$
$$z_t = Hy_t + v_t \tag{2.97}$$

with Gaussian noise implemented in the noise terms

$$u \sim \mathcal{N}_{[0,\Sigma_0]}(u) \tag{2.98}$$
$$w \sim \mathcal{N}_{[0,\Sigma_F]}(w) \tag{2.99}$$
$$v \sim \mathcal{N}_{[0,\Sigma_H]}(v) \tag{2.100}$$

Eq. 2.95-2.97 can be interpreted as detailed formulation of messages in the graph of the general state space model (Fig. 2.9). Thus the general graph can be detailed for linear state space modeling (Fig. 2.11).

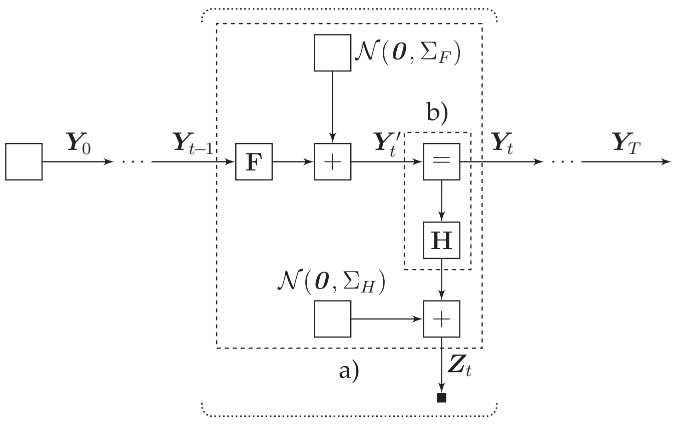


Figure 2.11: Kalman Filter as a probabilistic graphical model in FFG notation. Adapted from [110]

For inference on the linear state space model graph the update rules from Sec. 2.5.4 can be used (Table 2.2). Since H is generally very large and not easily invertible, the backward multiplication rule is not applicable. Instead, the composite rule can be applied on the combination of the equality factor and H. This update rule is actually the original idea of the Kalman Filter.

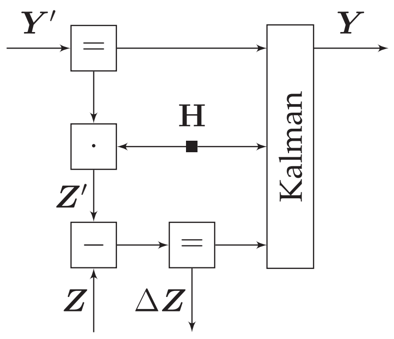


Figure 2.12: Detailed view on the composite update rule. This corresponds to part b) in Fig. 2.11.

The composite update rule can be visualized as a graph (Fig. 2.12) introducing some intermediate variables, well-known from traditional Kalman Filters [69]

[112] [118]: The *predicted state* Y^- is projected into the measurement space, resulting in a *predicted observation* Z^-. The forward matrix multiplication rule can be applied:

$$\mu_{Z^-} = H\mu_{Y^-} \tag{2.101}$$
$$\Sigma_{Z^-} = H\Sigma_{Y^-}H^\mathsf{T} \tag{2.102}$$

The predicted observation is compared to the actual observation Z leading to the *prediction error* ΔZ (also called *innovation*).

$$\mu_{\Delta Z} = \mu_Z - \mu_{Z^-} \tag{2.103}$$
$$\Sigma_{\Delta Z} = \Sigma_Z + \Sigma_{Z^-}. \tag{2.104}$$
$$\tag{2.105}$$

Finally, the predicted state Y^- is updated to the new state estimate Y using the innovation and a weighting factor, called *Kalman gain* K.

$$\mu_Y = \mu_{Y^-} + K\mu_{\Delta Z} \tag{2.106}$$
$$\Sigma_Y = \Sigma_{Y^-} - KH\Sigma_{Y^-} = (I - KH)\Sigma_{Y^-} \tag{2.107}$$
$$\text{with Kalman gain} \quad K := \Sigma_{Y^-}H^\mathsf{T}\Sigma_{\Delta Z}^{-1} \tag{2.108}$$

Combing all forward messages to one big update rule leads to the compact Kalman equations given in [112]:

$$\mu_{Y_{t+1}} = F\mu_{Y_t} + K_{t+1}(\mu_{Z_{t+1}} - HF\mu_{Y_t}) \tag{2.109}$$
$$\Sigma_{Y_{t+1}} = (I - K_{t+1}H)(F\Sigma_{Y_t}F^\top + \Sigma_F) \tag{2.110}$$
$$\text{with} \quad K_{t+1} := (F\Sigma_{Y_t}F^\top + \Sigma_F)H^\top(H(F\Sigma_{Y_t}F^\top + \Sigma_F)H^\top + \Sigma_H)^{-1} \tag{2.111}$$

This is the compact forward pass update rule, that can be implemented without knowledge about any graphical representation. In fact, this is the common way for restricted, easily manageable applications, such as the basic example application in Chapter 6. The graphical representation (Fig. 2.11, Fig. 2.12) splits the overall Kalman method into single simple update rules that represent different aspects of the Kalman filter. With this knowledge the Kalman filter principles can be combined in new ways leading to other well known filters (e.g. IMM filter (Sec. 2.6.5)) or completely new filters as already described in [86]. The introduced explicit depiction of the composite rule allows modeling the IMM filter (Sec. 2.6.5) as a factor graph because it uses the prediction error ΔZ for estimating a likelihood for each transition model.

2.6.4 Particle Filter

There are several sampling methods including Gibbs sampling, importance sampling and markov-chain monte-carlo methods. [35] and [77] give a good overview about how they can be represented as message passing in factor graphs. This section will focus on the well-known particle filter.

Particle Filtering (also known as sequential Monte-Carlo integration [35] [44]) is a forward-only message passing method utilizing several sampling methods. The basic idea is that the complex continuous states Y are approximated by samples (particles) and the integral-product rule is approximated by eq. 2.39. The particle representation allows multimodal non-gaussian distributions in Y and non-linear transition models $P(Y_t|Y_{t-1})$. Transition models just have to define how a single particle is predicted to the next time step. This can also include a sampled uncertainty that can be based on any probability distribution. Additionally, because the importance sampling method is used for observation model evaluation, the observation model can be applied straight forward on each sample resulting in the same freedom as for the transition model.

While these are many advantages over parametric methods using limited models like Kalman filtering, other problems occur like degeneracy and sample impoverishment [77]. One way to handle degeneracy is to include a resampling step to reorder samples around probable regions in the distribution and reset the weights. This leads to a famous particle filter implementation called sampling importance resampling (SIR) filter. For an overview about several other particle filter implementations, see [21] and [45].

The SIR filter can be described in FFG notation (Fig. 2.13). For particle filters the prior, transition function and observation function can be described using non-linear generic distributions $\mathcal{P}_{[\theta]}(X)$:

$$P(y_0) = \mathcal{P}_{[\theta_0]}(y_0) \tag{2.112}$$
$$P(y_t|y_{t-1}) = \mathcal{P}_{[f(y_{t-1}),\theta_F]}(y_t) \tag{2.113}$$
$$P(z_t|y_t) = \mathcal{P}_{[h(y_t),\theta_H]}(z_t) \tag{2.114}$$

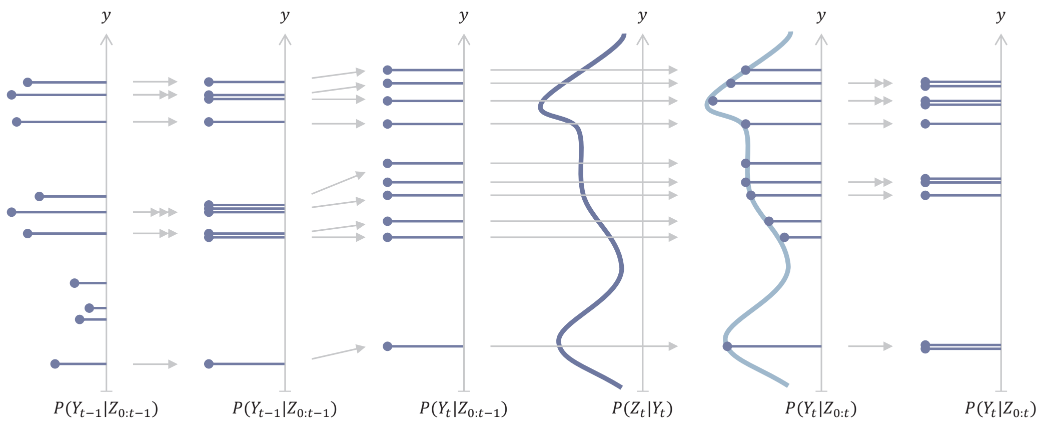
with f and h being non-linear deterministic, not necessarily invertible functions. This can also be formulated as noisy non-linear equations given by

$$y_1 = u \tag{2.115}$$
$$y_t = f(y_{t-1}) + w_t \tag{2.116}$$
$$z_t = h(y_t) + v_t \tag{2.117}$$

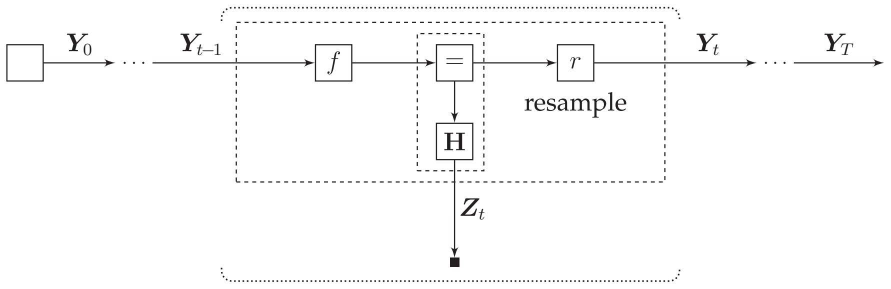
with generic noise implemented in the noise terms

$$u \sim \mathcal{P}_{[\theta_0]}(u) \tag{2.118}$$
$$w \sim \mathcal{P}_{[\theta_F]}(w) \tag{2.119}$$
$$v \sim \mathcal{P}_{[\theta_H]}(v) \tag{2.120}$$

Eq. 2.115-2.117 can be interpreted as detailed formulation of messages in the graph of the general state space model (Fig. 2.9). Thus the general graph can be detailed for sample-based state space modeling (Fig. 2.13).



(a) Examplary samples during forward pass.



(b) FFG representation.

Figure 2.13: FFG representation and examplary samples of the SIR particle filter.

For inference on the state space model graph representing the SIR filter the update rules from Sec. 2.5.4 can be used (Table 2.4). After the initial sampling of particles $(\hat{y}_0^{(i)}, w_{Y_0}^{(i)})$ from $P(y_0)$ the update cycle of each time step consists of [112]:

1. Forward propagation of every sample using the transition model $P(y_t|y_{t-1})$ consisting of the deterministic transition function $f()$ and subsequent noise addition.

2. Weighting of each sample by the likelihood $P(z_t|y_t)$ that the observation results from it. Therefore $P(z_t|y_t)$ has to be evaluated at $\hat{y}_t^{(i)}$ which can be seen as evaluating the particle by the subgraph including the deterministic observation function $h()$ and the generic observation noise.

3. Resampling of the resulting distribution to prevent degeneracy using the resampling factor. Different resampling methods are possible.

A big advantage of the particle filter lies in the weighting step evaluating the observation weight of the given state sample. The observation functions can consist of large sub graphs, evaluating several sensor models. An extensive usage of the particle evaluation is part of the depth-first message passing introduced in [124].

2.6.5 Interacting Multiple Model Filter

The Interacting Multiple Model filter (IMM filter) [29] [118] is an efficient implementation of switching model filters [25] (also known as switching state space models [55], switching Kalman filter model [95]). There are two classes of switching model filters (Fig. 2.14): Switching transition models and switching observation models. The IMM filter belongs to the class of switching transition models. With the switch, different behavior modes can be realized resulting in differently implemented transition models.

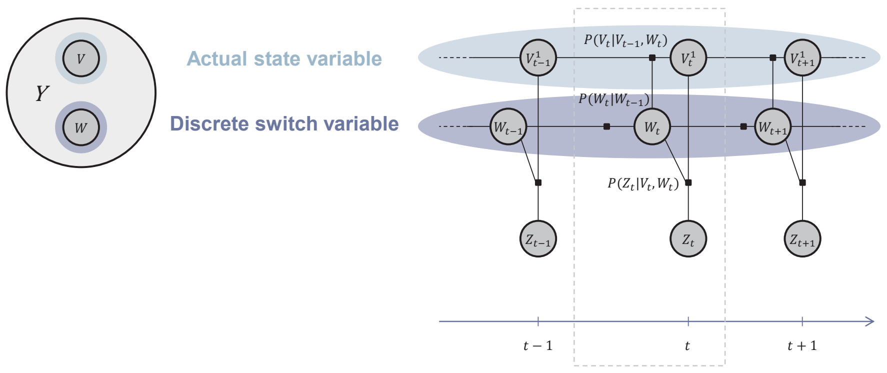
The switch is realized by a discrete variable that holds a probability for each behavior mode and a transition model for these discrete behavior mode distributions. Therefore, the state space Y is split into the continuous actual state V and the discrete mode distribution W:

$$Y = (V, W) \tag{2.121}$$

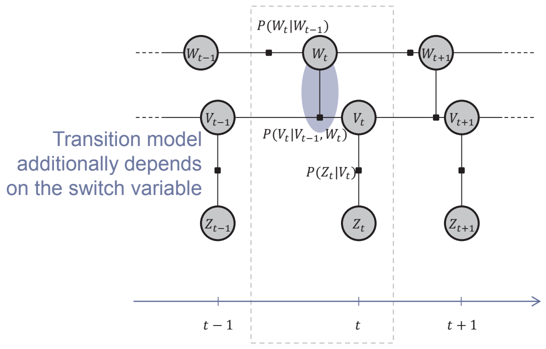
$$P(y_t|y_{t-1}) = P(v_t, w_t|v_{t-1}, w_{t-1}) \tag{2.122}$$

The complete transition model factors to a mode-dependent transition model on the continuous part and a discrete transition model for the mode distribution:

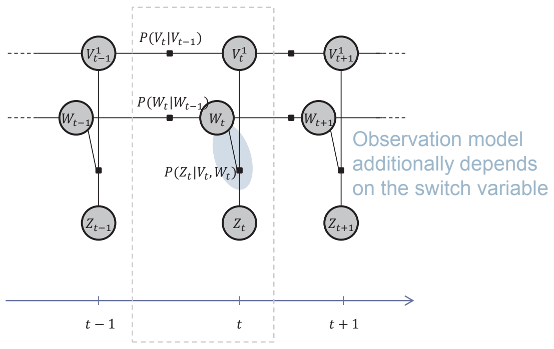
$$P(y_t|y_{t-1}) = P(v_t|v_{t-1}, w_t)P(w_t|w_{t-1}) \tag{2.123}$$



(a) Switching model filters



(b) Switching transition models



(c) Switching observation models

Figure 2.14: Switching model filters simultaneously estimate the actual state V and a discrete switching variable W as part of Y. Switching transition models have a different meaning than switching observation models. Adapted from [95].

The two models can be described as

$$P(v_t|v_{t-1}, w_t) = P_{w_t}(v_t|v_{t-1}) \tag{2.124}$$

$$= \mathcal{P}_{[v_{t-1}, \theta_{w_t}]}(v_t) \tag{2.125}$$

$$P(w_t|w_{t-1}) = T_{w_{t-1}, w_t} \tag{2.126}$$

where P_{w_t} is a continuous transition model belonging to behavior mode w_t and T is a transition model for the mode distribution. With this description the switching transition model filters can be seen as a stack of continuous state filters (often Kalman filters (Sec. 2.6.3)) with a discrete state filter (see Sec. 2.6.2) deciding about the probabilities of the continuous state transitions.

Given a sequence of observations $z_{0:t}$, estimating the continuous state distribution $P(V_t|z_{0:t})$ and the mode distribution $P(W_t|z_{0:t})$ is desired. The mode distribution describes how well the corresponding transition model matches the current observation sequence. This likelihood can be derived from the prediction error ΔZ that was already needed for inference in Kalman filtering (Sec. 2.6.3).

The invention of the IMM filter origins from the problem that exact inference would result in a belief state V_t at time t with $O(N_W^t)$ mixture components, where N_W is the number of discrete behavior modes. The reason is that the true chain of behavior modes can switch at every time step resulting in N_W^t different behavior mode chains that all have to be estimated. Several approximating filters (GPB1, GPB2, etc. [25] [55]) use different techniques to reduce the estimate size, where the IMM filter has an outstanding balance between accuracy and computation effort.

Now, the main idea of the IMM filter is to let the (possibly expensive) filtering step of the continuous filters run only once per model but let the different behavior mode chain interaction act as much as possible. Therefore, the N_W different state estimates from the previous time step are first predicted using the mode transitions defined in T and then combined to one estimate per subsequent transition model. This step is called *interaction* (between the different state estimates).

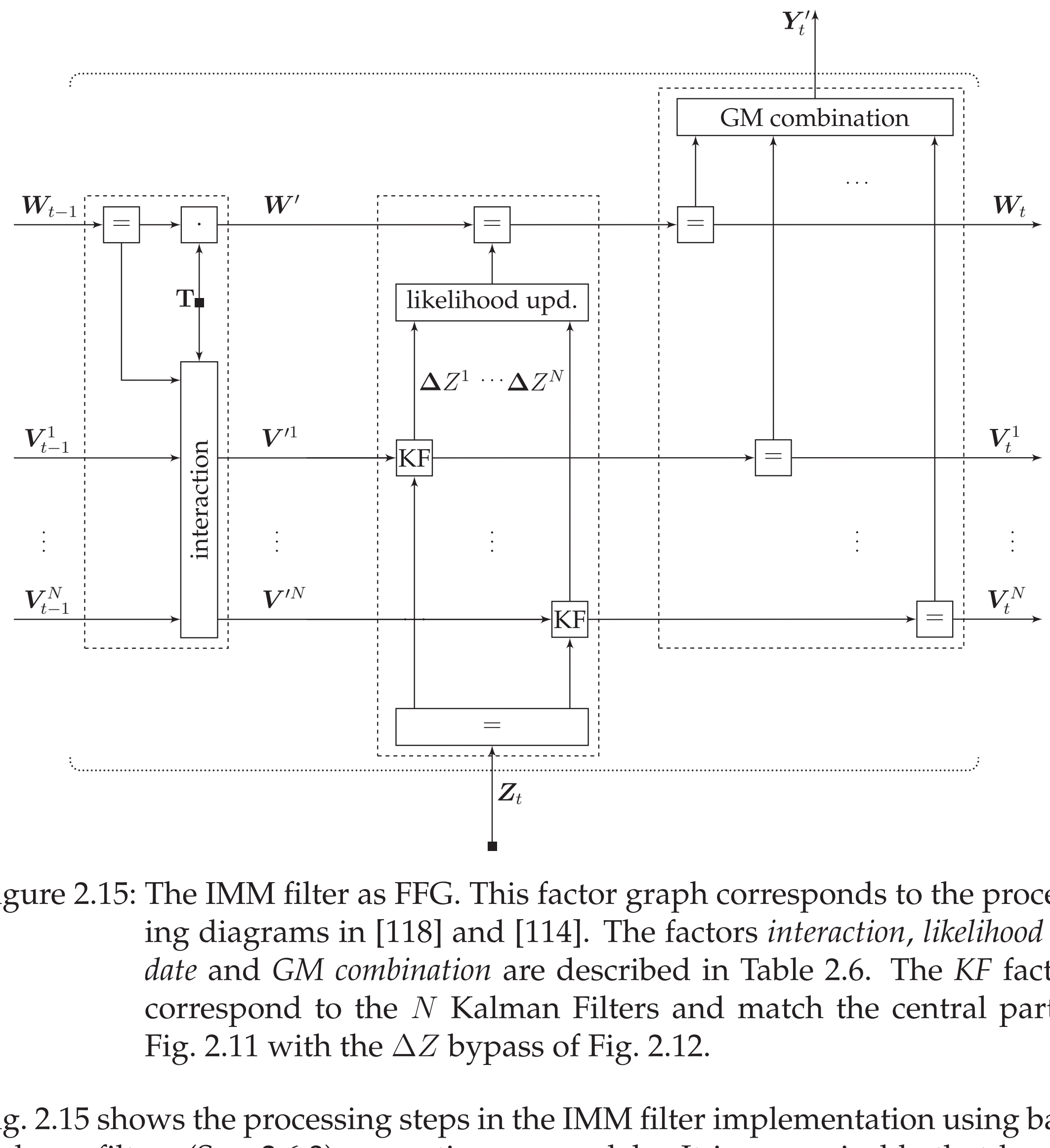


Figure 2.15: The IMM filter as FFG. This factor graph corresponds to the processing diagrams in [118] and [114]. The factors *interaction*, *likelihood update* and *GM combination* are described in Table 2.6. The *KF* factors correspond to the N Kalman Filters and match the central part in Fig. 2.11 with the ΔZ bypass of Fig. 2.12.

Fig. 2.15 shows the processing steps in the IMM filter implementation using basic Kalman filters (Sec. 2.6.3) as continuous models. It is recognizable that known components are used like the transition and observation update of the discrete part and one Kalman filter per mode. Additional update rules specific to the IMM filter are given in Table 2.6.[10]

The IMM filter can also be interpreted as an intermediate state space approximation between Kalman filter (Sec. 2.6.3) and particle filter (Sec. 2.6.4): The gaussian mode approximations V_i together with the mode weights W can be seen as a Gaussian mixture representation of Y, where each component has a meaning, namely corresponding to one transition model behavior. This means, the IMM filter has the advantage of being able to estimate multi-modal state distributions (in contrast to the Kalman filter) while keeping the mode representation simple and with a discrete meaning (in contrast to the particle filter).

2.6.6 Multi Target Tracking

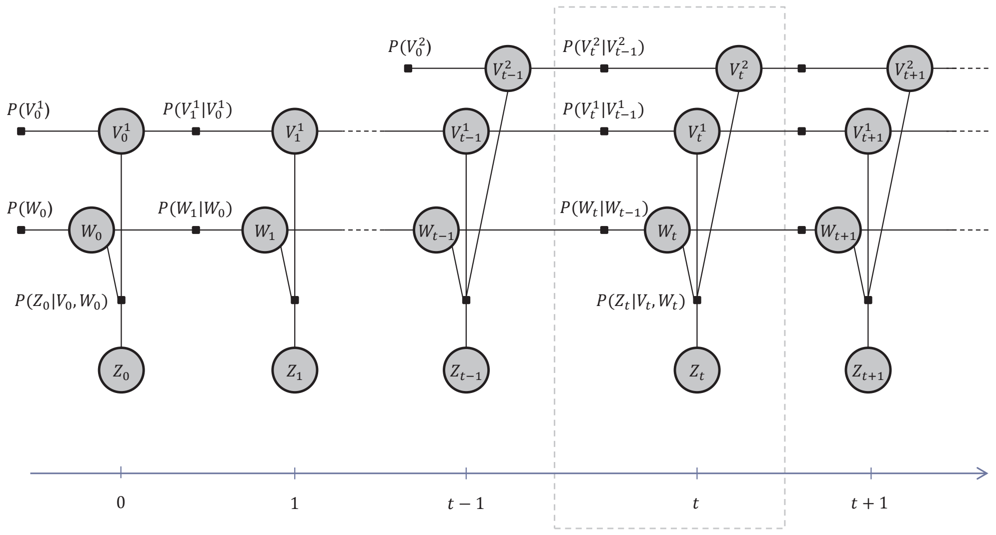


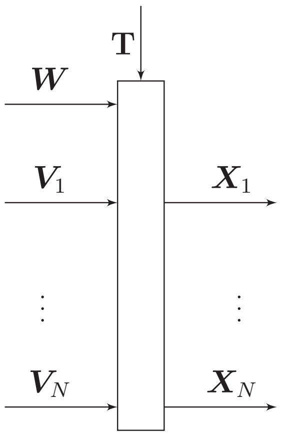
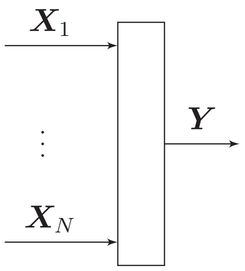
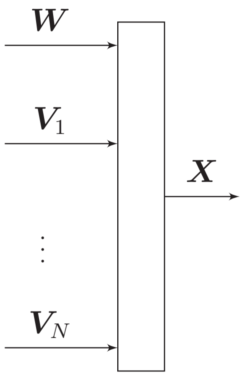
Figure 2.16: Multi Target Tracking with switching observation models representing the data association problem. A track for a new object V^2 is created at $t-1$.

Multi target tracking is the process of estimating the state of not only one but multiple objects (so called targets). In contrast to the previous sections, multiple filters have to run in parallel, each estimating the state of one real object. The challenge is that observations are usually not associated to the objects and thus another uncertainty lies in the relationship about which observation was the result of which object.

This can be seen as another class of switching model filters. While the switching transition models correspond to different behaviors of a single target (Sec. 2.6.5) the switching observation models change the relationship between states and

[10]The update rules correspond to the IMM filter algorithm presented in [24], neglecting the normalization factors.

Table 2.6: Message update rules specific to the IMM filter.

Node	Update rule
 interaction	$$\mu_{X_j} = \sum_{i=1}^{N} T_{ji}^{\mathsf{T}} w_i \mu_{V_i} \quad (2.127)$$ $$\Sigma_{X_j} = \sum_{i=1}^{N} T_{ji}^{\mathsf{T}} w_i (\Sigma_{V_i} + (\mu_{V_i} - \mu_{X_j})(\mu_{V_i} - \mu_{X_j})^{\mathsf{T}}) \quad (2.128)$$
 likelihood update	$$y_i = \mathcal{N}_{[0,\Sigma_{X_i}]}(\mu_{X_i}) \quad (2.129)$$
 GM combination	$$\mu_X = \sum_{i=1}^{N} w_i \mu_{V_i} \quad (2.130)$$ $$\Sigma_X = \sum_{i=1}^{N} w_i (\Sigma_{V_i} + (\mu_{V_i} - \mu_X)(\mu_{V_i} - \mu_X)^{\mathsf{T}}) \quad (2.131)$$

observations (Fig. 2.14). The later can be used for considering sensor failures (used in fault diagnosis systems [95]) or for the association problem in multi target tracking. A multi target tracking system can be described using a discrete switching variable influencing the observation models (Fig. 2.16).

Therefore, the state space Y is split into the continuous state V holding the states of all targets $V^{(i)}$ and the discrete observation switch W:

$$Y = (V, W) \tag{2.132}$$

$$V = (V^{(1)}, \dots, V^{(N)}) \tag{2.133}$$

$$P(y_t|y_{t-1}) = P(v_t^{(1)}, \dots, v_t^{(N)}, w_t|v_{t-1}^{(1)}, \dots, v_{t-1}^{(N)}, w_{t-1}) \tag{2.134}$$

The complete transition model factors to single independent transition models on the continuous part and a discrete transition model for the observation switch distribution:

$$P(y_t|y_{t-1}) = P(v_t^{(1)}|v_{t-1}^{(1)}) \cdots P(v_t^{(N)}|v_{t-1}^{(N)}) \cdot P(w_t|w_{t-1}) \tag{2.135}$$

The continuous part transition models $P(v_t^{(i)}|v_{t-1}^{(i)})$ are not dependent on the observation switch but switching transition models can easily be integrated here by implementing one IMM filter per target. Often $P(w_t|w_{t-1})$ is neglected if there is no temporal dependency like in data association. If sensor failures are considered, it can be used to model the temporal permanence of failures.

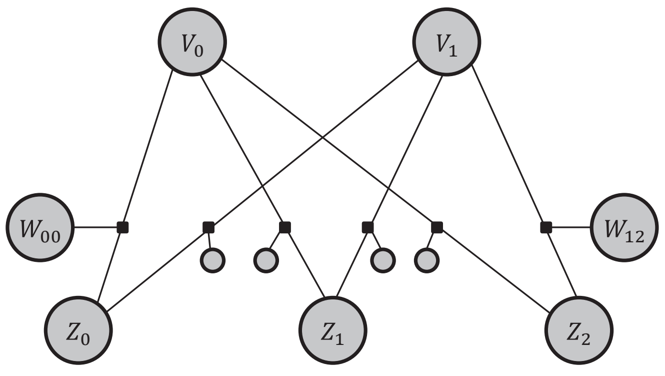
In contrast to the switching transition model (Sec. 2.6.5) the observation model is conditioned on W:

$$P(z_t|y_t) = P(z_t|v_t, w_t) \tag{2.136}$$

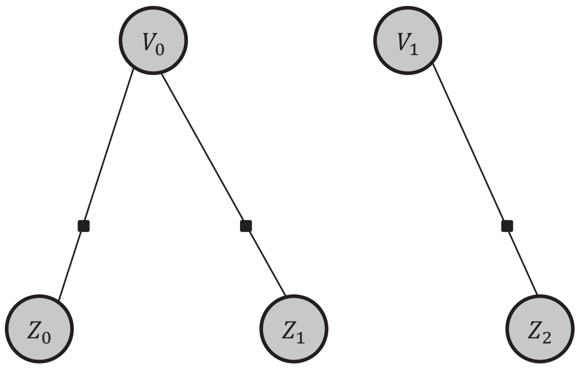
$$= P_{w_t}(z_t|v_t^{(w_t)}) \tag{2.137}$$

$$= \mathcal{P}_{[v_t^{(w_t)}, \theta_{w_t}]}(z_t) \tag{2.138}$$

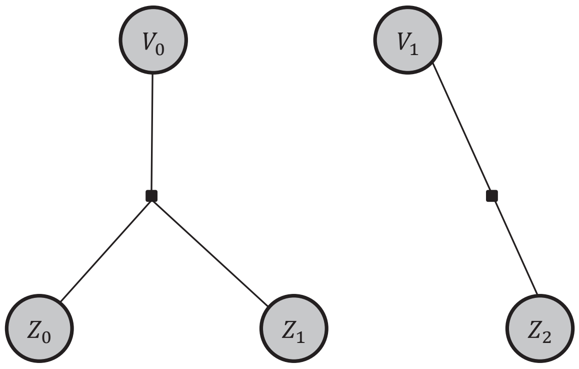
If associations are known $P(z_t|v_t, w_t)$ acts as a multiplexer with switch w_t deciding which state $v_t^{(i)}$ is "passed through" to the observations. If they are not known, the most probable association has to be estimated additionally to the actual target states. The general problem considering all possible associations (called multiple hypothesis tracking (MHT) [23] [32]) explodes quickly, even in the single target case [95]. Several approximating algorithms have been developed, like global nearest neighbour, probabilistic data association [23], joint probabilistic data association [50], auction algorithm, hungarian algorithm .... Their common idea is to assess the pairwise (mahalanobis) distance between the predicted states $P(v_t^{(w_t)}|z_{0:t-1})$ and the possible observations z_t to estimate the likelihood of possible (joint) assignments. A detailed consideration of efficient probabilistic data association using message passing is given in recent work [90] including multi-sensor setup and set-type methods.



(a) Pairwise simple calculations.



(b) Reduced space of relationships.



(c) Clusters for joint estimations.

Figure 2.17: Gating and clustering: pairwise simple calculations (a) help removing unlikely relationships resulting in a reduced amount of relations to estimate (b). For joint estimations, clusters can be created (c) from the reduced space of relationships.

One simple key tool, that reduces complexity largely is *gating* and *clustering* (Fig. 2.17): Before estimating the precise likelihood of joint assignments, very unlikely associations are discarded by a more efficient preliminary calculation, e.g. a distance measure. Afterwards groups of associations can be created that have to be estimated jointly. Thus, the expensive calculations are focused on a sub group of associations that are likely to have an impact. These methods are worth to consider on a generic graphical model as depicted in Fig. 2.17 to transfer them to other problem formulations.

Up to now, a known, fixed number of targets was assumed. In reality, the number of targets is often varying and not known and has to be estimated as well. There are established methods for multi target tracking using heuristics to create new filters and delete obsolete ones. Another approach is to use set-type filters that will not be considered in this thesis but can be mapped to factor graphs as well [90]. In general, an unknown number of targets means that the state space has to grow and shrink dynamically. Attributes not being fixed is called "first order" models as opposed to "propositional" models. Existing first order probabilistic languages are discussed in Sec. 2.7.

2.7 First-Order Probabilistic Languages

First-order probabilistic languages (FOPLs) have been developed to reason about uncertainties concerning the logic of scenes (derived from first-order logic). A good overview about the core concepts is given in [104] [114] and a more recent survey about different approaches can be found in [65]. The goal is to implement a structure that allows handling of uncertainties considering existence, number and relationships of real world objects. Several languages achieve this goal in different detail.

While there are also rule based and procedural based FOPLs, this section focuses on entity-relation based ones [65]. These follow the principles of entity-relationship models describing worlds by entities refering to real world object types, attributes describing these object types and relationships between the entities, e.g. spatial, temporal or semantic. The entities and relationships are formulated on class level and then instantiated multiple times to describe a specific world. The probabilistic extension of the entity-relation based FOPLs means that additionally probabilistic dependencies are defined on class level and then utilized to describe possible instantiations. The searched probability distribution is the overall distribution over all possible worlds (configurations). Analogously to the previous sections, observations help to reduce the uncertainty and find the true world with correct relationships and attributes of objects.

The most advanced FOPLs are built up on Bayesian Networks (Sec. 2.4.2). First, classes and instances have to be introduced via object-orientation. There are two main approaches of so-called object-oriented Bayesian networks (OOBNs) that extend Bayesian networks by classes and instances: one by Koller and Pfeffer [76] and one by Bangsø [22]. They have in common, that Bayesian network fragments are defined on class level and instantiated and connected multiple times to form a large Bayesian network. Thus, repeating network fragments have to be defined only once. When learning, OOBNs restrict reoccuring conditional probabilities to be the same in all instances of a class and to be learned as one generic model in the class definition. The main difference between the two approaches is that Bangsø has defined the class interface more clearly, such that every class knows its probabilistic model completely and model analysis can be performed already on class level. Also the class hierarchy allowing inheritence of properties from parent classes profits from this.

Based on the two OOBN definitions, two different entity-relation based FOPLs have been developed: probabilistic relational models (PRMs) [54] and the object-oriented probabilistic relational modelling language (OPRML) [64]. The languages use reference attributes to reference other instances, similar to foreign keys in relational databases. Relationships can be modeled by either referencing another instance in an attribute with a meaningful name or by using a separate entity representing the relationship and referencing two or more instances that belong to this relationship. From an object-oriented perspective this means that any relationship between instances can be modeled instead of only *part-of* relationships as in OOBNs. Since the OPRML is based on the OOBN definition of Bangsø, it inherits its advantages. The language can model existence, number and reference uncertainties and in [64] it is also described how the language can be used to model state space models based on discrete state spaces.

The OPRML is defined as a formal language [108] [109] using a syntax definition and its visual representation is based on the concept of a frame based representation system [92] consisting of *frames* and *slots*. This system is comparable to object-orientation in programming languages where *frames* correspond to *classes* and *slots* to their *members*.

Although the OPRML includes already many features that are necessary for a generic modeling language for estimation problems it lacks some key features:

- It is based on Bayesian Networks,
  lacking the generality of factor graphs (Sec. 2.4.4).
- It is used with discrete variable representations only. An extension to continuous variables is needed.
- Inference methods are not integrated in the object-oriented formulation.

Chapter 3 defines a new language that is largely inspired by the OPRML but overcomes these limitations.

2.8 Automated Driving as an Estimation Problem

As described in Sec. 2.2, automated driving can be seen as implementing an intelligent agent that observes the environment, interprets the situation, makes a decision and acts on the environment in a closed loop. The overall task has been split into groups of components, where the software groups are:

- Low level perception
- High level perception
- High level execution
- Low level execution

The filters described in Sec. 2.6 have their origins in the perception groups: *Object Tracking* needs temporal filtering using simple or multiple transition models representing the motion of dynamic objects. *High level fusion* has to fuse different sensor data respecting different observation models. And *situation prediction* can be realized using the prediction capability of state space models.

Several approaches have been developed to accomplish *scene understanding*. First-order probabilistic languages as described in Sec. 2.7 are promising to describe relationships between entities, especially semantic ones that bring geometric relations to a more abstract, human understandable level [115].

In general, advanced works in all perception tasks use factor graphs (Sec. 2.4.4) and detailed discussions on inference methods (Sec. 2.5) to describe their approach in a generic way, such as *(ego) localization* approaches by static environment observations [40] [62] and *road layout estimation* by multiple (stereo) camera features [52] [124].

To achieve an extensive perception a holistic approach is obvious. Factor graphs seem to be a common denominator but will be a challenge to handle and keep modular if graphs grow.

Following the observations about Forney-style factor graphs (Sec. 2.4.4) and single value messages representations (Sec. 2.5.1), even basic signal processing can be represented in factor graphs. Thus, they are also applicable to components of the execution group like *controllers*. Without going too much into detail, *planning* components can be seen as a (FFG)-factor having the probabilistic representation of the perceived environment as input (including as much uncertainties as possible) and a (maximum a-posterior) decision as output.

It is a tendency of recent years, that hard decisions are delayed as much as possible to later modules in the tool chain: Where hard decisions were made already in low level perception, now uncertainties including multi modalities are considered more and more and kept available up to the decision making planning modules to make them aware of all uncertainties. Seeing the whole automated driving application as one large estimation problem can serve the description and progress of this process.

2.9 Summary

This chapter has formed the basis for the subsequent chapters. The key results of this chapter are:

- The main terms in the traffic domain are defined.
- The main components needed for automated driving are introduced following a generic system architecture based on the concept of an intelligent agent.
- Uncertainties have to be considered and handling large estimation problems is challenging. Possible solutions consist of factorizing the overall problem into sub problems or learning hardly adjustable parameters from observations.
- Probabilistic graphical models can describe correlations, causalities and credibility. Factor graphs are the most generic description.
- Inference on probabilistic graphical models is a wide field. Message approximation and message passing schedules are key aspects that have to be considered.
- Well-known filter algorithms can be depicted using factor graphs. They visualize the different possibilities in action and confirm the potential of factor graphs as a generic description method. Several aspects, such as Kalman gain, particle resampling, switching model filters and association gating are incorporated.
- First-order probabilistic languages are promising extensions to handle object-orientation and relational aspects but are still at an early stage and not yet widely used in automated driving applications.
- Probabilistic graphical models are already used in current automated driving components and will receive more attention towards an improved overall scene understanding. A holistic modeling language including probabilistic estimation aspects and object-orientation is needed.

2.9.1 Contributions

Although the main scientific contributions are focus of the subsequent chapters, some minor contributions are already presented:

- **Generic system architecture for automated driving** (Sec. 2.2)
  (previously published in [14] and more detailed in [13] [8] [18] [2])

- **Modeling large estimation problems**
  via probabilistic graphical models (Sec. 2.3)
  (previously published in [3])

- **Message update rules for discrete messages, sample-based messages and conversions** (Sec. 2.5.4)
  (inspired by the linear update rules of [85])

- **Detailed factor graph description**
  of the Kalman composite update rule (Sec. 2.6.3)
  (the Kalman composite update rule itself is part of [85])

- **Factor graph description of the IMM filter** (Sec. 2.6.5)

- **Basic generic factor graph description of gating and clustering** (Sec. 2.6.6)

2.9.2 Conclusion

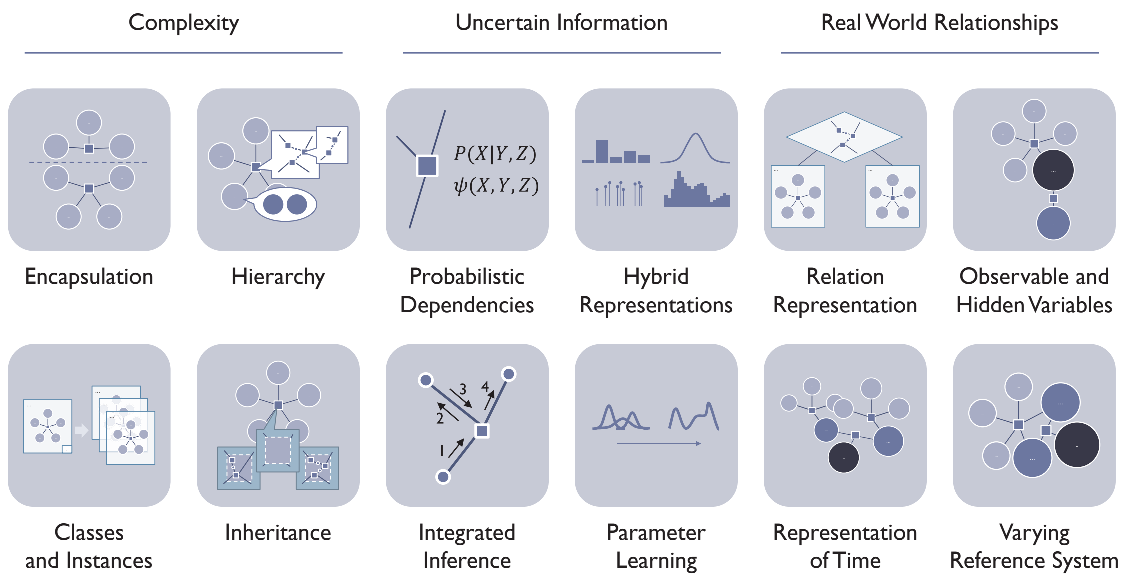


Figure 2.18: Twelve properties can be derived as requirements for a generic modeling language. Detailed analysis is given in Chapter 4.

It becomes clear that a generic description language for holistic system modeling has a key role to achieve a profound scene understanding that is the basis for traffic participant prediction. This work proposes such a language and applies it to various automated driving applications. Derived from the preceiding sections a suitable language must fulfill the following properties (Fig. 2.18):[11]

- **Encapsulation:** The overall system has to be separatable into single components (Sec. 2.2). A suitable language supports this by a sufficient encapsulation of class attributes.
- **Hierarchy:** Variables have to be defined by detailed sub sets of variables. Factors by detailed sub graphs of the probabilistic graphical model (Sec. 2.6.5).
- **Classes and Instances:** Filters and other principles should be defined/parametrized on abstract level and instantiated multiple times according to a given scenario (Sec. 2.6.6).
- **Inheritance:** Kalman and Particle filter are two sub classes of state space models (Sec. 2.6). And a simplified method for gating can have the same base class as the detailed multi hypothesis tracking (Sec. 2.6.6). Inheritance from a common base class allows the definition of common properties and principles on abstract level which results in different views on a specific implementation according to the perspective of the instance that is using it.
- **Probabilistic Dependencies:** It is crucial to model dependencies probabilistically to represent the uncertainties given in automated driving applications (Sec. 2.3). The dependencies can be modeled in the sense of correlation, causality or credibility corresponding to Markov Random Fields, Bayesian Networks and Dempster Shafer Evidence Theory (Sec. 2.4).
- **Hybrid Probability Distribution Representations:** Different representations of probability distributions have to be usable in the same graph since they all have their advantages and disadvantages (Sec. 2.5.1, Sec. 2.5.4).
- **Integrated Inference:** Inference methods (update rules and schedules) have to be definable and combinable in the overall modeling language (Sec. 2.5).
- **Parameter Learning:** Hardly definable parameters have to be able to be learned from observations (Sec. 2.3). Learning of parameters has to be possible in local areas (sub sets of overall estimation variables) in larger probabilistic graphical models that consist of many other dependencies that are defined by expert knowledge.
- **Relation Representation:** Key feature of FOPLs (Sec. 2.7) is to model relative attributes by relations between objects. These relations have to be supported by a holistic language.
- **Observable and Hidden Variables:** Aspects of estimation problems can be separated into observable and hidden variables (Sec. 2.3). These variables have to be represented in a generic way to switch easily according to the given estimation task (Sec. 2.5).

[11]Just a short overview is given here. A detailed formulation of the requirements of each property follows in Chapter 4.

- **Representation of Time:** Temporal dependencies are evident in most estimation tasks (Sec. 2.2). A generic language has to support the representation of time and has to allow mapping of this property to filtering principles (Sec. 2.6).
- **Varying Reference Systems:** Measurements are usually relative to the sensor and the variables of interest are often in a global coordinate system (Sec. 2.3). Different views on the same situation and its attributes must be possible.

The OPRML fulfills already some of these requirements. It will be used in the following as a basis but needs extensions to include all properties.

3 Object-Oriented Factor Graph Modeling Language

This chapter defines the central unified modeling language, called Object-Oriented Factor Graph Modeling Language (OOFGML). It is independent of the actual estimation domain, thus all statements here will be given without reference to the target domain, the traffic domain. The chapter focuses on the actual language definition (Sec. 3.1) and the core usage idea including the method for deriving a domain specialization (Sec. 3.2) and instantiating for given observations (Sec. 3.3). Available implementation opportunities will be given showing the big applicability of the language (Sec. 3.4). A definition of useful special terms (Sec. 3.5) is given before the findings are summarized in Sec. 3.6. The language's capabilities will be explained and evaluated on examples in chapter 4.

3.1 Abstract OOFGML Syntax

The OOFGML is a modeling language that is comparable to an abstract object-oriented programming language extended by probabilistic dependency handling. Language definitions are based on formal language theory [108] [109] and while there are different ways to define what *words* are valid according to the language's syntax a common way is to describe the syntax by a set of rules, called formal grammar. A basic meta model is given in Fig. 3.1.[1]

Object-Oriented programming languages as well as FOPL (Sec. 2.7) are defined as formal languages. The OOFGML can be seen as an extension of both of them: An abstract object-oriented programming language extended by probabilistic dependency handling or a knowledge representation language extended by inference and processing methods. The structure of the following syntax definition is inspired by the OPRML syntax definition in [64].

[1]The correctness of the definition according to formal language theory and the usage of consistency checks and other formal language tools is not focus of this work. Instead this work tries to show what components for such a language are required by the domain's applications. Correct definition of the language including completeness and utilization of existing language definitions is focus of future research.

The language consists of:

- A set of **classes**
$\mathbf{C} = \{C_1, C_2, ..., C_{n_C}\}$,
e.g. *Vehicle*, *Car* and *Truck*

- A partial ordering $\lhd$ over $\mathbf{C}$, which defines the **class hierarchy**,
e.g. *Car* is a subclass of *Vehicle*

- A set of **attributes**
$\Lambda_C = \{\lambda_1, \lambda_2, ..., \lambda_{n_\Lambda}\} \; \forall \, C \in \mathbf{C}$
These represent **class properties with and without uncertainties**, e.g. the *position* of a *vehicle*.

- A set of **factors**
$F_C = \{f_1, f_2, ..., f_{n_F}\} \; \forall \, C \in \mathbf{C}$
These describe **dependencies between attributes**.

- A set of **update rules**
$U_C = \{u_1, u_2, \ldots, u_{n_U}\} \; \forall \, C \in \mathbf{C}$
These describe **how messages are updated during inference**.

- A set of **processing scripts**
$\Gamma_C = \{\gamma_1, \gamma_2, ..., \gamma_{n_\Gamma}\} \; \forall \, C \in \mathbf{C}$
These describe **instantiation rules and inference schedules** for handling the processing in applications.

3.1.1 Class Hierarchy

The class hierarchy defines an inheritence hierarchy on the classes. Class properties like attributes, factors, update rules and processing scripts are inherited from parent classes. They can be defined partially in one class and refined in a subclass, for example a parent class can define the involved attributes and a subclass can add a definition of the dependency formulation. Attributes and factors that refer to other classes can be refined from refering to a super class to refering to a subclass of this super class.

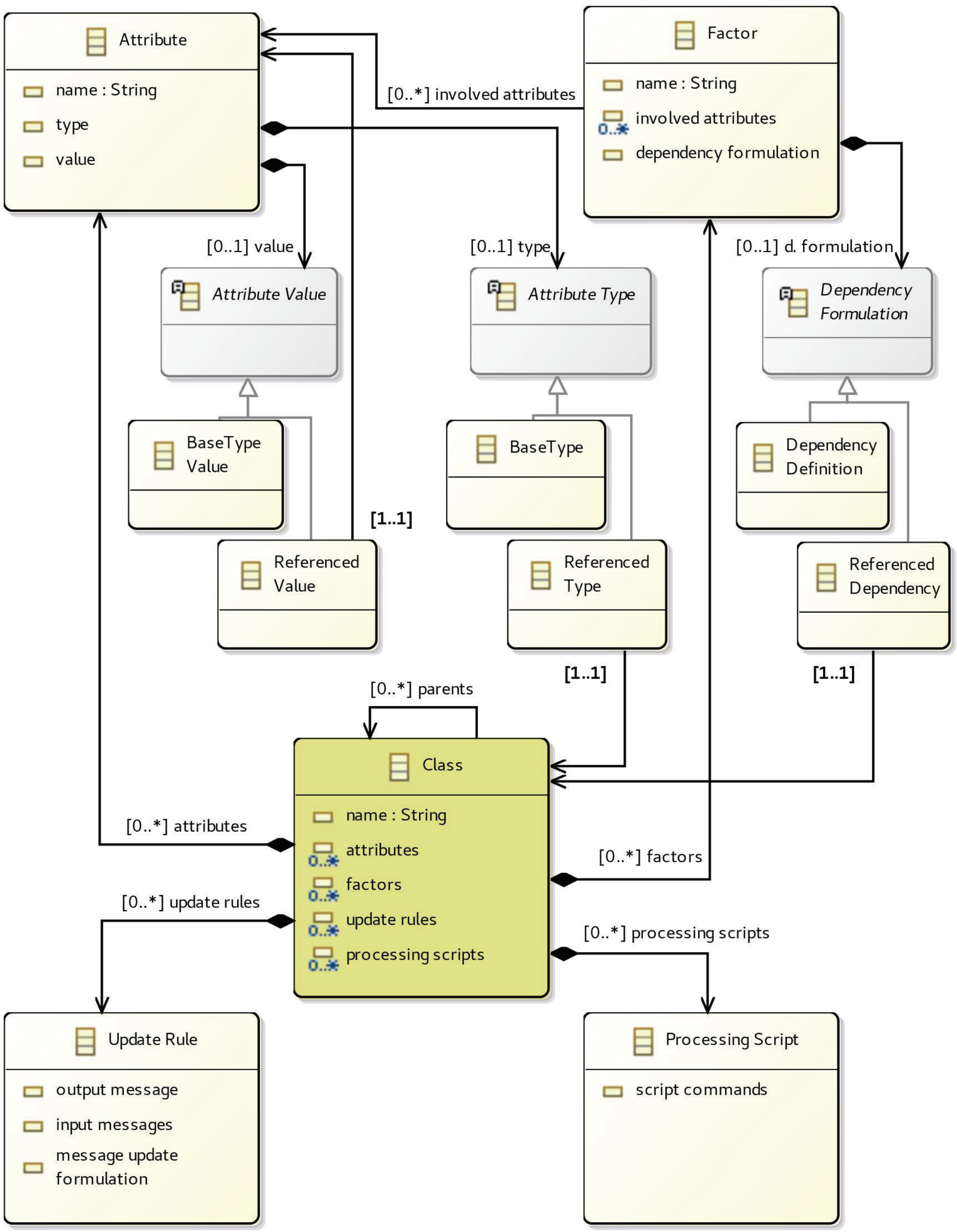


Figure 3.1: Meta model depicting the abstract syntax of the proposed language. The central definition of a *class* can be matched to existing generic languages such as the unified modeling language (UML).

3.1.2 Attributes

Each attribute λ consists of:

- A **name** identifying the attribute
- A **type** describing the range of values the attribute can take. The type is
  - a base type, or
  - a reference to another class.
- An optional **value** of the attribute. This can be
  - a base type value, or
  - a reference to another attribute.

Since the type can be defined as a reference to another class, the class hierarchy on the referenced classes can be used to derive a hierarchy of subtypes, for example special state representations. The **base types** represent a basic type like integer, floats or sets for discrete distributions. If the value is set to a base type value, the attribute becomes a constant. If the value references another attribute the attribute can also be seen as an alias for the referenced attribute. Both cases set a fixed value for the given attribute on class level. In general values will be set during instantiation (Sec. 3.3) including references to one ore many instances of the class referenced by the type.

3.1.3 Factors

Each factor f consists of:

- A **name** identifying the factor
- A set of **involved attributes**
  $\Lambda_f = \{\lambda_1, \lambda_2, ..., \lambda_{n_{\Lambda_f}}\}$ describing the attributes that are involved in the dependency formulation.
- A **dependency formulation** (or potential function)
  ψ_f describing the actual dependency between the involved attributes. This can be
  - a function definition including all involved attributes, or
  - a reference to another class
- An optional **reference to another factor** with the same dependency formulation
  describing that this factor is referencing the same instance that the other factor references.

Since the type can be defined as a reference to another class, the class hierarchy on the referenced classes can be used to derive a hierarchy of subtypes, for example special dependency factorizations. The factors with a direct function definition are also called **base factors** since they do not refer to another class but directly represent a dependency formulation, e.g *same value* or *fixed distance to*. If Ω is the set of all possible configurations of the involved attributes Λ_f, every factor has a function ψ_f from Ω to V, where V is a weighting space, e.g the set of positive real numbers $\mathbb{R}^+$:

$$\psi_f : \Omega \to \mathbb{R}^+ \tag{3.1}$$

The factor functions are used to describe matching configurations by local functions. The global function describes the fitness of all global configurations and an optimization on that function can be used to find the globally best fitting configuration. This is the generalized basis for all estimation problems.

A factor with involved attributes λ_f can be interpreted similarly to a factor in a factor graph. It defines **edges** between the factor and all involved attributes. Each edge defines two **messages**, one for each inference direction:

$$m_1 = (f, \lambda) \tag{3.2}$$
$$m_2 = (\lambda, f) \tag{3.3}$$

3.1.4 Update Rules

Each update rule u consists of:

- An **output message**
  $O_u = m_o$ describing the message that is calculated.
- A set of **input messages**
  $I_u = \{m_1, m_2, ..., m_{n_{I_u}}\}$ describing the attributes that are used as input messages.
- A **message update formulation**
  φ_u describing the actual update rule. This can be
  - a function definition including the input messages, or
  - a vector referencing other update rules, describing what update rules have to be calculated to achieve the given update rule.

Note that the vector of references can also be used with a single reference to define an alias or reference to an alternative graph representation for the given update rule.

3.2 From Language to Application

An expert can define the language for a specific domain or application. While the term Object-Oriented Factor Graph Modeling Language (OOFGML) refers to the overall language description, the domain specific language (DSL) is the application of the OOFGML to a specific domain or application including the definition of classes with attributes and factors specific to this domain (Sec. 1.3).

The capability of the OOFGML to model classes and references to other classes via attributes can be used to model entities and relations. Thus, every DSL can be seen as a definition of potential entities and relations specific to the domain, also known as a probabilistic ontology [64].

Ontologies are a high-level representation that describe domain specific properties. They are as precise as necessary and as generic as possible. Ontologies of different domains can be compared to each other and can be merged to compound domains. The same way, a generic domain language can be specialized and refined into a very specific domain language that describes, for example, a very specific application.

The ontology with the hierarchy on entities can not only be used to represent the domain specific properties of real world entities but also to represent inference methods specific to the application. These methods are directly incorporated in the same class hierarchy (as processing scripts and dependency formulations) and allow a very generic handling of inference methods throughout different applications.

The specialization and generalization is one of the key features of the OOFGML allowing to fulfill the generality goal of this thesis.

3.3 Model Instantiation and Inference

A possible configuration of the domain described in the DSL is called *world*. Its syntax consists of:

- A set of **instances**
  $\mathbf{I} = \{I_1, I_2, ..., I_{n_I}\}$,
  e.g. *Car 1*, *Car 2* and *Truck 1*
- For each class $C \in \mathbf{C}$, a subset $I_C \subset \mathbf{I}$, where I_C represents the set of instances of class C
- One or many instances $I_V \in \mathbf{I}$ as value for each attribute $I.\lambda \,\forall\, I \in \mathbf{I}$, written $I.\lambda = I_V$ or $I.\lambda = (I_V, I_W, \dots)$
- A value $x \,\forall\, I.\lambda \,\forall\, I \in \mathbf{I}$, where $C.\lambda$ is a base type, written $I.\lambda = x$

With the factors of the DSL a weight for every possible world can be calculated. The world with the largest weight is the most probable world. The weighted set of all worlds describes the probability distribution over all possible worlds.

In estimation problems (Sec. 2.3) the goal is often to find the most probable world given some observations. But also the probability distribution over all possible worlds can be of interest to get an estimate of how certain the system can estimate the observed situation.

To infer these worlds from observations, steps have to be taken to get from the DSL to the worlds:

1. Receive data from sensors
2. Create instances of the classes matching the probably observed environment and the inference methods to be used
3. Set instance references of attributes and factors
4. Set evidences from sensor data
5. Run inference methods
6. Extract possible worlds

An obvious inference approach is to run these steps from top to bottom but several adaptations are reasonable, for example creating an initial large set of instances with approximate inference methods resulting in a rough result that enables precise inference on a smaller set of instances afterwards, so-called *gating* in multi target tracking applications (Sec. 2.6.6). Such inference methods are used in the rather complex interaction estimation application in Chapter 9.

To achieve maximal flexibility, these steps can be described in the processing scripts, separated in and spread throughout the classes, allowing to share methods and principles with different applications. The ontology aspect of the DSL gives this functionality additional support. More details are described in detail in Sec. 4.2.3 and Chapter 6.

3.4 Implementation Opportunities

So far, the generic language is mainly a formalization of estimation problems including inference methods to solve them. The actual (efficient) implementation is not directly predetermined. Instead different opportunities are offered:

- **Full OOFGML implementation**
  The full specification of the language could be implemented and provided to application developers. This is a challenging approach and open for future research. A possible approach can be an offline modeling tool during algorithm design time with code generators that could then automatically

derive proper implementations. Or the full language could be implemented for online usage allowing dynamic algorithm switches and scheduling during runtime. Both approaches are not focus of this thesis.

- **FOPL implementation**
  An existing first-order probabilistic language (FOPL) (Sec. 2.7) can be seen as a subgroup of the language and thus an (existing) implementation for such a FOPL can be used. The OPRML can be seen as a subgroup using discrete state spaces and Bayesian network methods. It will be used in Chapter 9 for estimating interactions between traffic participants.
- **Specialized implementation**
  If the requirements on the application are clear after modeling it using the language a specialized implementation can be chosen that implements the exact specified algorithm. Also dynamic parts can be incorporated without implementing a full, generic class and instance system as in the previous options. This method is chosen in Chapter 7 for localizing an ego vehicle using the observed surrounding object constellation.
- **Established filter implementation**
  Another possibility is that the actual inference method can be streamlined to an existing established filter principle (Sec. 2.6). Then an implementation of the filter can directly be used. This method is considered in Chapter 6 at a very basic example of estimating an ego vehicle pose using absolute position measurements.
- **Modular implementation**
  If independence assumptions are introduced separating large parts of the modeled class structure the implementation can be accomplished in independent modules that communicate over a manageable number of interfaces. The different modules can then be implemented in very different ways including all of the previous methods. This is used at an exceptional scale in Chapter 8 for estimating the road layout in front of an ego vehicle.

The possibility of using different matched implementations underlines the large applicability of the proposed language.

3.5 Special Terms and Notations

After defining the language and its main usage methods some term definitions are added that are not necessarily required for language definition but can be introduced using the main definition and will help discussing the language in the subsequent chapters.

The terms class, sub class, parent class and instance have already been used and are defined just as in object-oriented programming languages. Factors describe dependencies between involved attributes λ_f. The attributes are also called

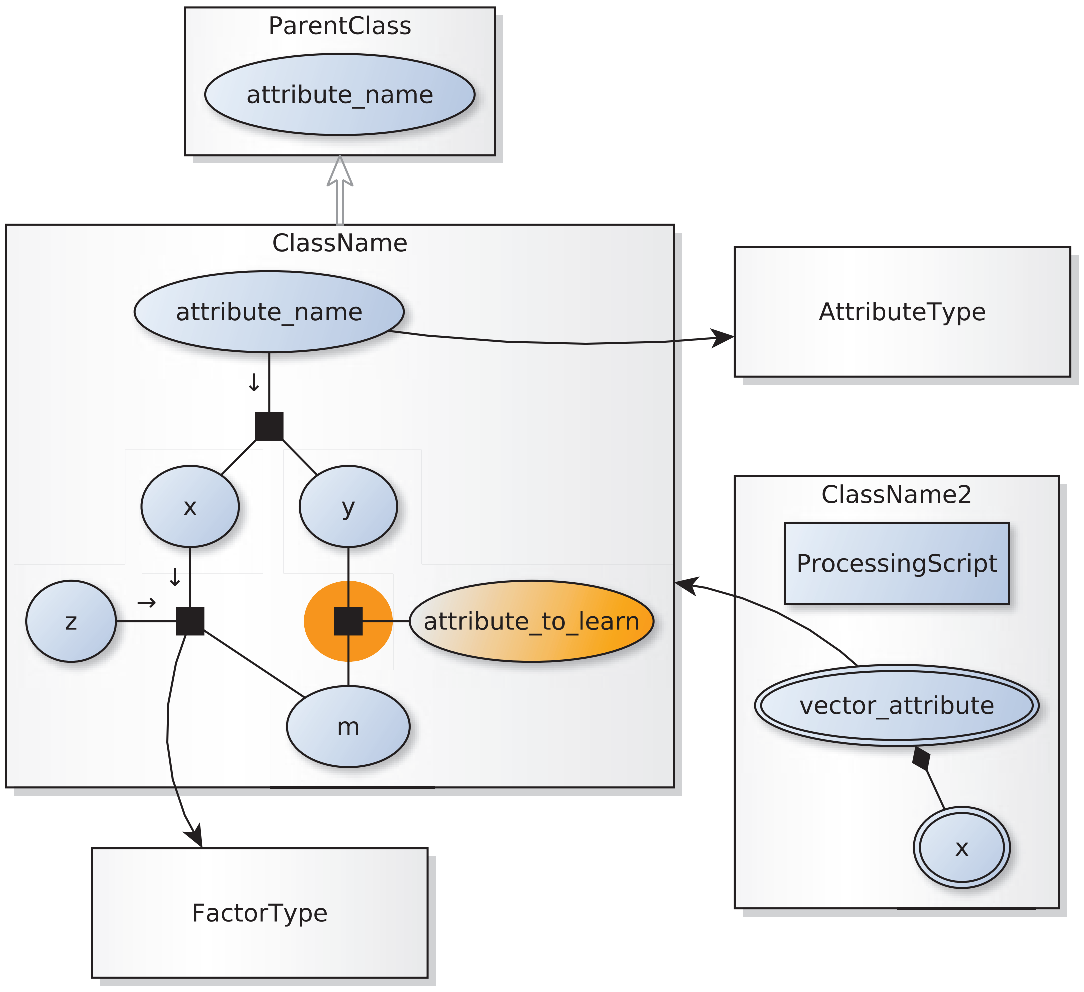


Figure 3.2: The chosen depiction of language elements. Class hierarchy is depicted by unfilled gray arrows. Attributes are round nodes, factors are black filled rectangular nodes. Type definitions are depicted by black arrows. Members of foreign attributes can be depicted by lines with a square-start. Multi-value array attributes are highlighted by double borders. In this example also a variable is highlighted to be learned by an orange color.

neighbors of f as in factor graphs. Similarly, all factors f_x that include attribute x are called neighbors of x.

Attributes and factors can reference other classes. They can be meant as single reference λ_s or multi-reference λ_m, meaning their instantiation can reference single instances or multiple instances of a target class:

$$I.\lambda_s = I_1 \tag{3.4}$$

$$I.\lambda_m = (I_1, \ldots, I_N) \tag{3.5}$$

The multi-reference can be seen as a vector attribute: $I.\lambda_m$ is a vector with N elements. Factors connected to such vector attributes have to be defined for arbitrary dimension of the vector since their actual dependency size is only defined after instantiation. Factors connecting a vector attribute to a single reference attribute can be used to convert (or compress) multiple values from multiple referenced instances into one attribute (similarly to aggregate slots in OPRML [64]). If a factor connects multiple vector attributes of same dimension and each dimension dependency is independent of the others the factor can also be defined as multi-reference factor. This references a non-vector dependency definition and applies it to all dimensions of the vector attributes.

For every reference attribute λ an inverse reference λ_{inverse} can be defined that references the original instance I_1 from the referenced instance I_2:

$$I_1.\lambda = I_2 \tag{3.6}$$

$$I_2.\lambda_{\text{inverse}} = I_1 \tag{3.7}$$

Using inverse references every connection introduced by a reference attribute can also be used in the inverse direction to connect to attributes of the instance. This opens various possibilities for modeling dependencies. The inverse reference of a vector attribute is a single reference or a vector reference itself according to whether it represents a one-to-many or many-to-many relationship.

Throughout this document OOFGML classes and instantiations are described as class definitions in special pseudo code or graphically as special diagrams. An example with exemplary elements is depicted in Fig. 3.2 and the corresponding pseudo code is given here:

```
□ ParentClass
    ○ attribute_name
□ ClassName : ParentClass
    ○ AttributeType attribute_name
    ○ x
    ○ y
    ○ z
    ○ attribute_to_learn
    ○ m
```

■ `FactorType f_1(x,z,m)`

■ `f_2(y,m,attribute_to_learn)`

■ `f_3(attribute_name,x,y)`

→ u_m(attribute_name, z, attribute_to_learn) =
f_3.$u_{x,y}$(attribute_name), f_2.$u_m(x,z)$, f_1.u_m(y, attribute_to_learn)

□ `ClassName2`

○ `ClassName vector_attribute`

s ProcessingScript():
```
new_object = newInstance(ClassName)
vector_attribute.append(new_object)
```
vector_attribute.u_m(attribute_name, z, attribute_to_learn)

Classes and members of classes are distinguished by special symbols:

□ `class`

○ `attribute`

■ `factor`

→ `update rule`

s `script`

Inherited member properties from a parent class are depicted in gray. This allows the visualization of additional definitions to the same member. Names of three factors are declared. One is referencing another class called `FactorType`.

One exemplary update rule is given that references three factor-related update rules. It implements a straight-forward strategy that would not have to be defined explicitly but could also be derived from graph structure. Instead of messages each update rule just uses the involved attributes as identifiers. This way the explicit definition of messages is circumvented in the depiction. If this identifier is ambiguous an additional `.from` suffix can ensure clarity, e.g. `attribute .from(FactorName)`.

One exemplary processing script is given. It uses a basic function to create instances (`newInstance()`), appends the new instance to a vector attribute and refers to an update rule. Special processing scripts can be used that are triggered after specific events, such as `onInstantiation()`.

In the example a member of an attribute is depicted that references another class (x in `ClassName2`). There (and also for factors) a shortcut can be taken by using the dot-member style, e.g. `vector_attribute.x`.

Classes are here depicted as rectangles. They can also be other geometrical shapes such as, e.g. diamonds to emphasize their semantic meaning as relationship in entity-relationship models (Fig. 4.9).

If attributes shall get a new meaningful name in a sub class, this can be achieved by introducing new attributes with same properties and equality factors between the old and the new attributes. For clarity this is abbreviated in the attribute definition:

```
○ new_attribute (=old_attribute)
```

The most detailed example of a whole class structure using all these pseudo code class definitions is given in Chapter 6.

3.6 Conclusion

The key points of this chapter are:

- The unified modeling language is called Object-Oriented Factor Graph Modeling Language (OOFGML).
- The language includes attributes, dependency and processing definitions in an object-oriented structure.
- The main usage idea includes a hierarchy of language derivations over a domain specific language (DSL) to a specific application.
- Instantiations can be created that represent a specific world configuration. Factors describe a factorization of the joint function of all attributes.
- The language is mainly defined for problem and algorithm formalization. There are various opportunities for implementations.
- The possibility to use multi-reference attributes further extends the language applicability. Several special terms and notations have been introduced.

The capabilities of the language will be discussed in the next chapter.

4 Language Properties

This chapter explains and evaluates the defined language from Chapter 3 using defined requirements and exemplary use cases. Several language properties will be motivated and formalized. The examples are motivated from the traffic domain but are valid for other domains.

The focus in this chapter lies on the single properties and their realization while an exemplary class structure and the integration in real applications is focus of subsequent chapters.

An overview over all 12 properties is given in Fig. 4.1. They can be grouped into three groups:

- Handling Complexity (Sec. 4.1)
- Handling Uncertain Information (Sec. 4.2)
- Handling Real World Relationships (Sec. 4.3)

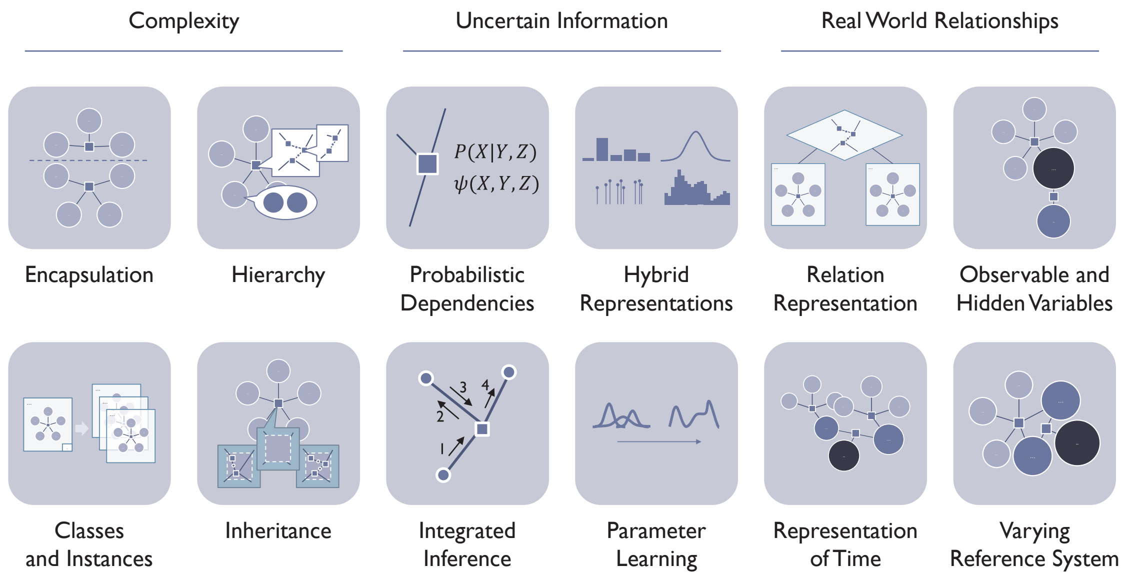


Figure 4.1: Twelve properties are required for a generic modeling language. They can be grouped into handling complexity, uncertain information and real world relationships.

4.1 Handling Complexity

Estimation problems can be complex due to a multitude of attributes that contribute to the understanding of the situation. This complexity has to be handled by several principles that are also used in object-oriented programming languages [119] or object-oriented development in general [20]. The taxonomy of object-orientation is often argued and differently defined [111] [63] [20]. D. J. Armstrong [20] proposes a taxonomy of 8 concepts as a result of an extensive survey over 239 sources. This section uses a similar taxonomy[1], but also incorporates additional requirements for the unified modeling language. The principles are grouped into *Encapsulation*, *Hierarchy*, *Classes and Instances* and *Inheritance*.

4.1.1 Encapsulation

Consider a system modeled as one big monolithic black box. If this system needs to be implemented it is hard to test functionalities at an early stage. Or if it needs to be debugged after detecting a failure it is very difficult to localize the error. Therefore bigger systems are usually modeled as single modules that focus on subtasks and interact with each other using defined interfaces. These interfaces have to be well-defined and understandable for human experts to be able to interpret the data and apply metrics on the values to determine the validity of the modules.

If a modular system is developed without a holistic system model, it is likely that interfaces restrict the estimation capabilities to an unwanted extent. While the desired language shall be able to model a probabilistic estimation system as a whole, a straight forward way to introduce and split the overall system into modules is needed.

The following requirements are given for the encapsulation property:

- **Grouping**
  It must be possible to group logical parts of the overall system into smaller components. These modules shall include the attributes and dependency descriptions necessary to describe the given part as an independent component.

- **Interfaces**
  Every module needs an interface to separate internals from the outside and to communicate with other modules.

[1]Armstrong's concepts *abstraction*, *class* and *object* are combined in *Classes and Instances*, the concept *polymorphism* is contained in *Inheritance* and the concepts *message passing* and *method* are handled in Sec. 4.2. In this work additional attention is given to the general *Hierarchy* of attributes and factors.

- **Module Connection**
  Interfaces of one module have to be providable by other modules. There has to be a direct way to describe a connection between two or more modules.

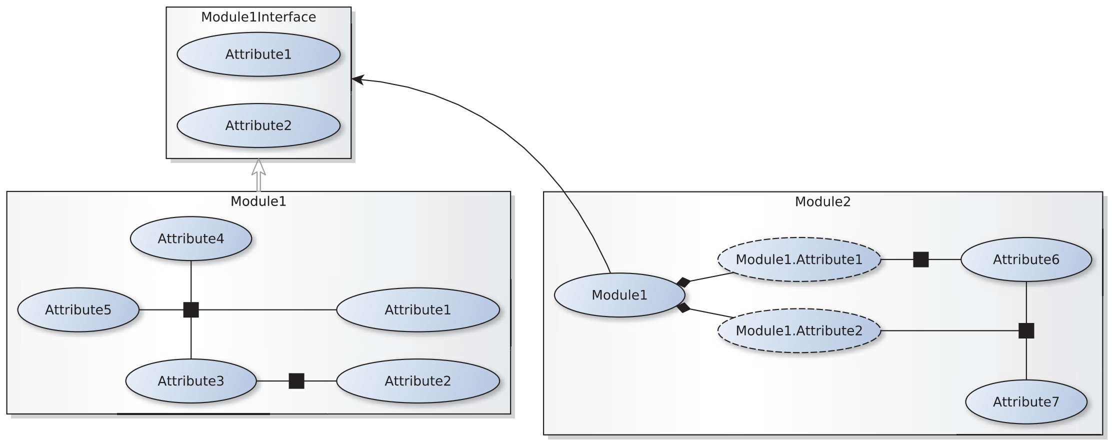


Figure 4.2: Example of the encapsulation property: The module interface of `Module1` is described by `Module1Interface` and used in `Module2`.

The OOFGML has the capability to fulfill the encapsulation property. Classes and thus also their instances can be used to group logical parts. Attributes describe module interfaces and references to other classes and instances can be used to implement module connections. An example is given in Fig. 4.2. It matches the encapsulation capabilities of the OPRML (Sec. 2.7).

Detailed usage of this property in applications can be found in Chapter 8 and Chapter 9.

4.1.2 Hierarchy

When developing an estimation system several aspects have to be considered in varying detail. Different experts may contribute to different aspects.

High level relationships should be modelable without knowing the details of each single component. On the other hand detailed algorithms have to be defined without considering their usage in a larger architecture. They are defined once and can be used as sub components in different larger components.

A hierarchical structure has to combine these components. Attributes have to be detailed with sub attributes, dependencies have to be detailed with sub dependency structures.

The following requirements are given for the hierarchy property:

- **Hierarchy for attributes**
  High level attributes have to be representable by more detailed low level attributes.

- **Hierarchy for factors**
  High level factors have to be representable by more detailed low level factors to describe detailed dependencies and independencies between subattributes.

- **Representation as relations**
  The relationship between classes describing that one class is part of another class has to be describable in an ontology. The ontology gives an overview of how classes are stacked into each other.

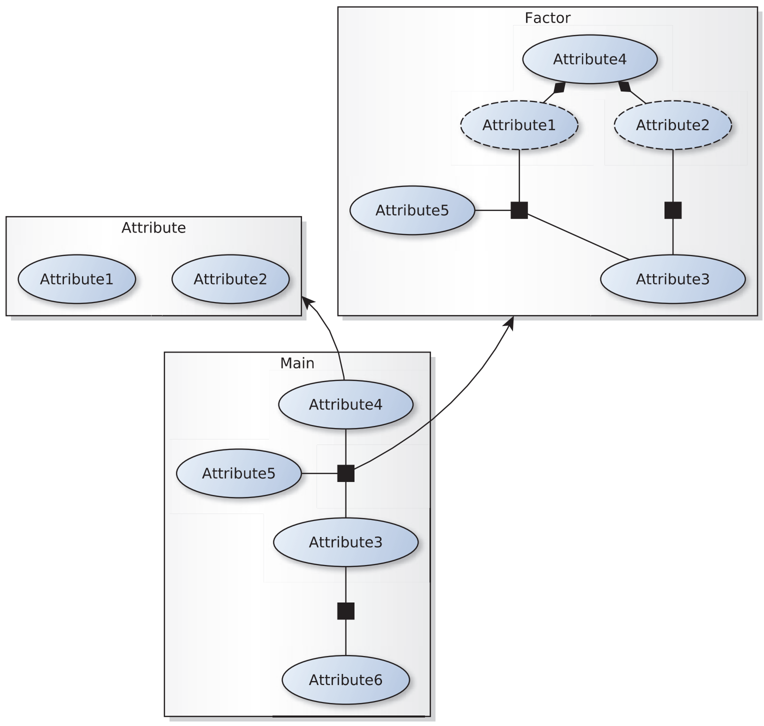


Figure 4.3: Example of the hierarchy property: Attributes and factors can be defined in separate classes.

The OOFGML has the capability to fulfill the hierarchy property. The definitions of attributes and factors directly allow referencing other classes. The resulting hierarchy on factors matches the hierarchical methods using factor graphs, especially the FFG representation in Sec. 2.4.4 and Sec. 2.6.3.

To emphasize the relation between two classes additional *has*-relations can be introduced in the entity-relationship view. An example is given in Fig. 4.3. The basic hierarchy implementation of the OOFGML matches the one of the elementary OOBN definitions (Sec. 2.7). The representation as relations is possible due to the extension that were also made when defining the OPRML.

Detailed usage of this property can be found in all applications from Chapter 6 to Chapter 9.

4.1.3 Classes and Instances

Real world traffic situations have a varying number of traffic participants and constellations. An automated vehicle has to solve them in a generic, coherent way. Principles have to be defined once and applied multiple times.

A classes and instances structure can help to define single generic classes and then instantiate them according to the given situation. These classes have to include the attributes and dependencies of the objects as well as processing scripts and update rules. They should also describe under which conditions instances will be created to represent a specific situation correctly.

The following requirements are given for the classes and instances property:

- **Classes**
  Real world principles have to be organized in classes. Classes have to define properties of the real world principles in a template that can be used to instantiate multiple instances of real world entities.

- **Instances**
  Instances of real world entities originating from the same class have to be handled as single, separate objects but have to share the properties described in their common class. The actual configurations of their attributes differ. Even the existence of such instances can be differently certain.

- **Instantiation rules**
  Rules have to be defined when and how instances are instantiated according to available sensor data or preceding estimation results.

The classes and instances property is fulfilled by the OOFGML. Classes and instances are directly available in the language definition. In contrast to existing FOPL (Sec. 2.7) the instantiation rules can be described in the processing scripts. A basic example is given in Fig. 4.4.

The property is used in all applications but most utilized in the object interaction application in Chapter 9.

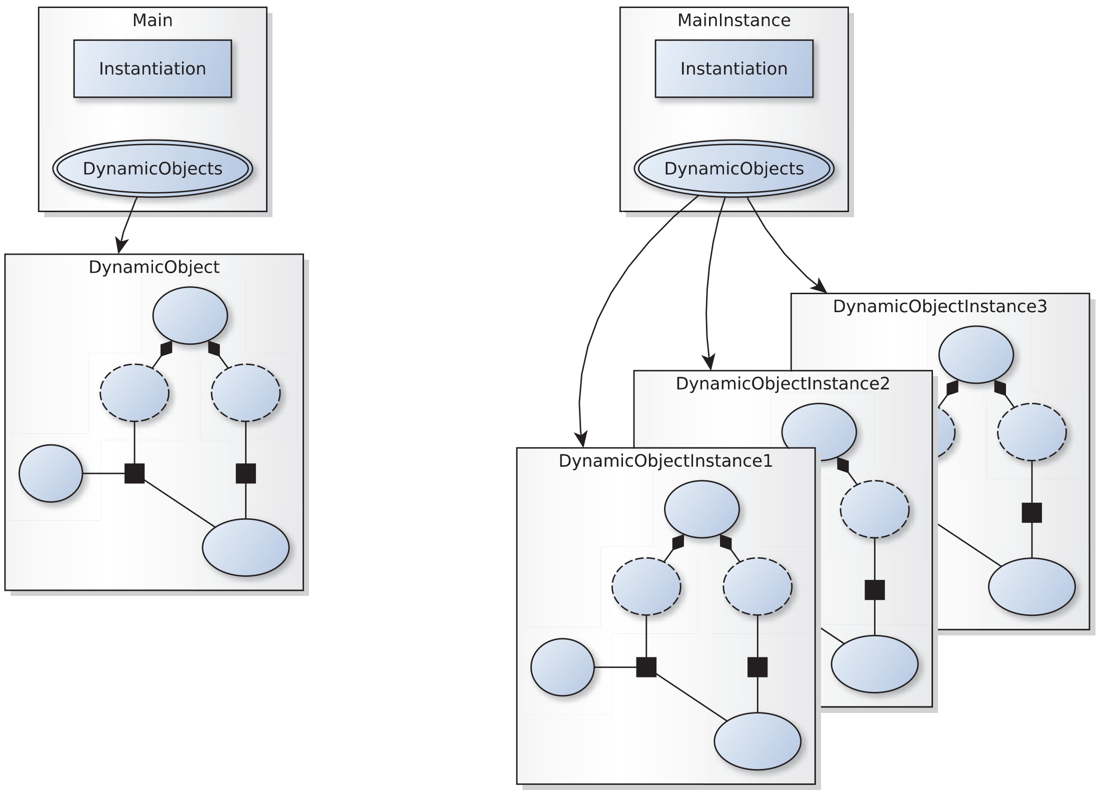


Figure 4.4: Example of the classes and instances property: A little amount of classes are defined on class level and are then instantiated multiple times. Instantiation rules can be defined on class level.

4.1.4 Inheritance

From the point of view of an automated vehicle all traffic participants are dynamic objects in general. All dynamic objects can move and their movement is important to consider.

But if detailed prediction is wanted also the characteristic of the actual dynamic object is relevant. Besides the ability to move a motor cycle has different acceleration capabilities than a truck. They both belong to the class of dynamic objects but are different specializations that extend the properties of the base class.

This separation into abstract and detailed class definitions is applicable to many real world entities. Therefore the unified modeling language needs a profound understanding of inheritance.

The following requirements are given for the inheritance property:

- **Inheritance hierarchy**
  An inheritance hierarchy on classes is needed that allows subclasses to share

common properties in their common parent class. These properties shall include attributes, dependencies between attributes and processing scripts.

- **Interfaces**
  Parent classes have to be usable as interfaces for subclasses. They have to be able to define only little information about the class structure e.g. just naming the attributes without defining types or even acting as an identifier only.

- **Multiple inheritence**
  Classes have to be able to inherit from multiple parents. On the one hand this allows defining multiple interfaces and thus preparing classes for usage in different environments. On the other hand it allows combining different features defined in parent classes, e.g. combine a state representation with a dependency definition.

- **Polymorphism**
  Different sub classes have to be able to be used via a common interface but

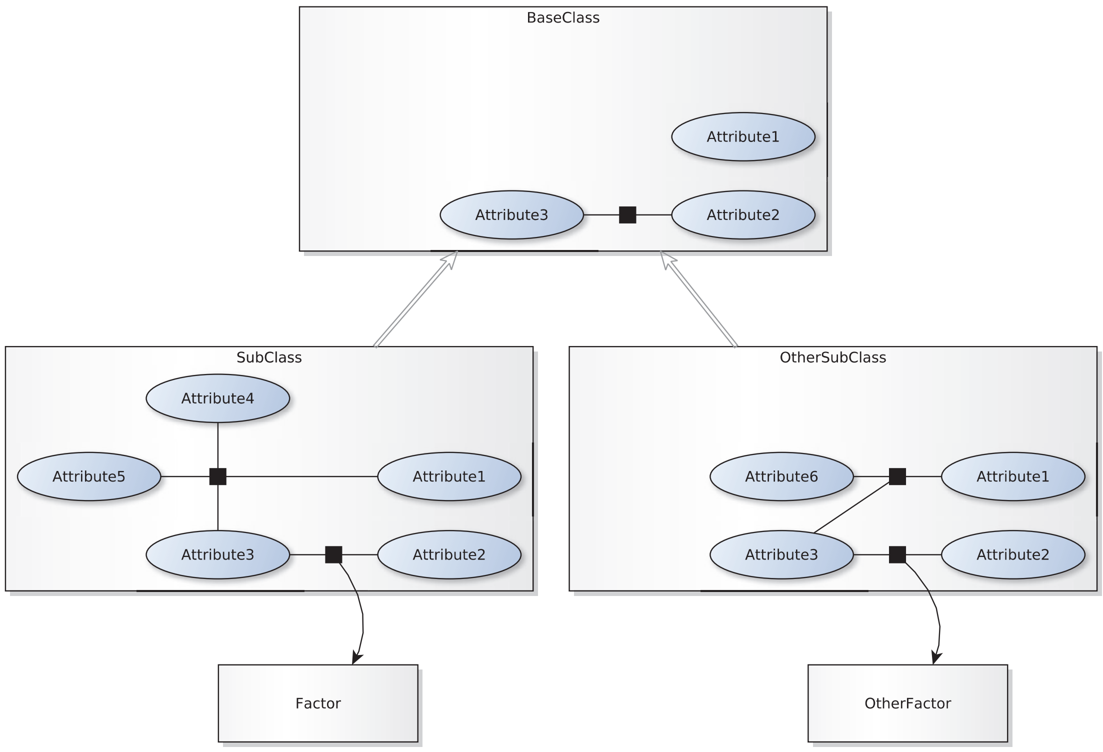


Figure 4.5: Example of the inheritance property: Every single property of the class definitions can be refined in sub classes, e.g. additional attributes and factors can be added and attributes and factors that had only names can be defined. The base class is a common interface for the sub classes.

respond with different specialized behavior given by their sub class definition.

The OOFGML is capable of the inheritance property. The class hierarchy directly induces the inheritance hierarchy. The inheritance is defined in a way that allows interfaces and multiple inheritance known from generic object-oriented languages. A basic example is given in Fig. 4.5.

The property is used in all applications especially for the state space model specialization in Chapter 6 and behavior formulation in Chapter 9.

4.2 Handling Uncertain Information

A fundamental challenge is how to handle uncertainties. Uncertainties lie in noisy sensor information and unknown dependencies. They have to be described in manageable representations and reduced by combining several information using dependencies. The principles are *Probabilistic Dependencies*, *Hybrid Probability Distribution Representations*, *Integrated Inference* and *Parameter Learning*.

4.2.1 Probabilistic Dependencies

The action of a traffic participant depends on its constellation to other objects. For example, dependent on the relative distance to a leading vehicle it can keep the velocity or has to brake to not collide with it. Thus the position attribute has to be brought in correlation with the action attribute. Dependencies have to be formulated between the attributes.

Usually the position of objects can only be observed with noise. There is no certain estimation of the object's position but a probability distribution over possible positions. The dependency formulation has to respect these uncertainties. It is obvious that established graphical models shall be supported such as Markov Random Fields, Bayesian Networks and Dempster Shafer Evidence Theory (Sec. 2.4).

The following requirements are given for the probabilistic dependencies property:

- **Dependencies**
  Dependencies between variables have to be modelable.
- **Uncertainties**
  Uncertainties have to be representable in dependencies. This includes respecting uncertainties on input variables when deriving output variables as well as modeling uncertainties of the dependency model itself.

- **Existing graphical models**
  Existing graphical models for correlations, causalities and credibility have to be representable (such as Markov Random Fields, Bayesian Networks and Dempster Shafer Evidence Theory).

- **Combination of existing graphical models in one language**
  Different existing graphical modeling methods shall be combinable in a single model. Having a large model, different parts of the model shall be described using different graphical models. Their connection shall be transparent.

The OOFGML has the capability to fulfill the probabilistic dependencies property. Dependencies can be modeled using the factors and their potential functions. The factors form a factor graph representation (Sec. 2.4.4) of all dependencies including the possibility to model deterministic as well as uncertain dependencies. Factor graphs can represent graphical models for correlations, causalities and credibility (Sec. 2.4.1 - Sec. 2.4.3). A combination of them is also modelable.

Probabilistic dependencies are the necessary core principle for all applications (Chapter 6 - Chapter 9).

4.2.2 Hybrid Probability Distribution Representations

Real world attributes include discrete and continuous values: There is a discrete set of lanes or a discrete set of behaviors a traffic participant can have while for example the position of objects is a continuous value. Since it is desired to model probabilistic dependencies respecting the uncertainties these attributes have to be described using probability distributions. There are several possibilities to approximate continuous probability distributions (Sec. 2.5.4).

Different approximations should be usable in the same model. Conversions between these approximations should be integratable in a transparent way.

The following requirements are given for the hybrid probability distribution representations property:

- **Independence from dependency model**
  The representation of state spaces needs to be independent from the dependency model.[2]

- **Hybrid graphs**
  Different state space representations need to be possible in the same model. They can be locally differing.

[2]This does not require that inference is independent from the state representations. Actually, inference methods are in general dependent on the state representations (Sec. 2.5.4).

- **Transparent conversions**
  Transformations between representations have to be possible and need to be integrated in class structure. On a semantic level, it shall be irrelevant how a variable is exactly approximated and if there needs to be a conversion to use it in a dependency with another variable.
- **Combination of representations in single variable**
  Different dimensions of a multi-dimensional variable shall be modelable using different representations.

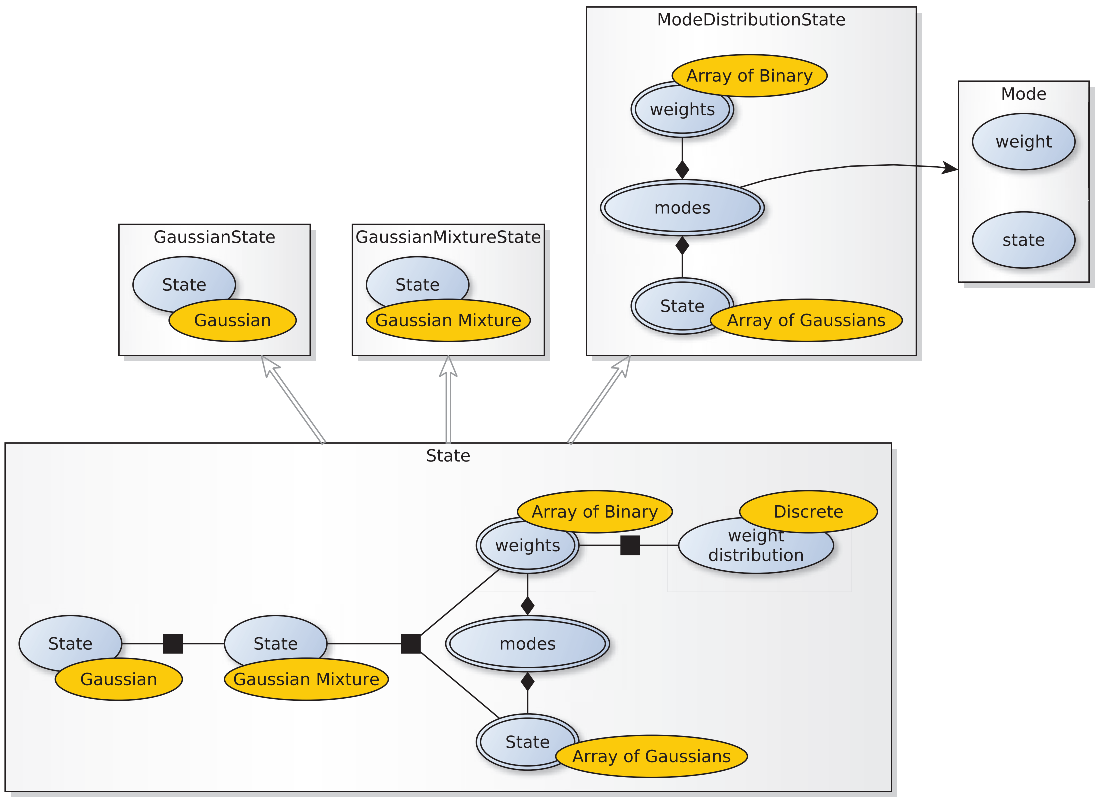


Figure 4.6: Example of the hybrid probability distribution representations property: Factors can convert between different probability distribution representations. Using different interface base classes a single `State` class can be used in other classes by referring to the corresponding approximation base class there.

The OOFGML has the capability to fulfill the hybrid probability distribution representations property. The definition of the representation of variables (Sec. 2.5.1) is separated from their name and their usage in factors. Thus, dependencies can be modeled on a base class of the variables and be specialized to a specific representation without considering the dependency model they are used in. State space representations can be defined locally. A large graph can consist of parts with different representations. The conversion between representations can be hidden in variable interfaces using multiple inheritance (Fig. 4.6): Different superclasses can provide varying state space approximations. Conversions can be hidden as internal dependency models that are needed if a corresponding interface is requested. Since variables can be aggregated to larger variables, also different representations inside one variable are possible. These can be helpful when modeling conversions (Table 2.5) or complex dependencies (Table 2.6).

Transparent approximation conversions are also considered in Chapter 8 and Chapter 9.

4.2.3 Integrated Inference

The main goal of estimation applications is to infer the most probable configuration under uncertain conditions for example by only having noisy observations. Depending on the inference task (Sec. 2.5) there are different requirements on runtime and precision of the inference algorithm. Additionally depending on the variables of interest only specific messages have to be passed. Message passing schedules (Sec. 2.5.3) can be defined that focus on a specific application and use specific approximation related update rules (Sec. 2.5.4).

These different inference algorithms and properties should be definable independently of the dependency model in a hierarchical structure.

The following requirements are given for the integrated inference property:

- **Independence from dependency model**
  The definition of the inference method needs to be independent of the dependency model. Different inference methods shall be applicable to the same dependency model depending on the requirements of the application, e.g. if fast prediction or precise smoothing is needed.
- **Hierarchical definition**
  Inference methods have to be described in a hierarchical structure to differ from basic principles to detailed message update rules. As much as possible has to be defined on domain-independent level.
- **Local definition**
  Different inference methods need to be applicable on parts of a single model.
- **Update Rules**
  Special update rules depending on the message representation have to be definable. They have to be defined in generic classes and applicable straightforward to specific dependency formulations.
- **Message passing schedules**
  The scheduling of inference steps (message passing) must be definable close to attribute and dependency formulation. Local definitions have to be combinable to overall scheduling rules.

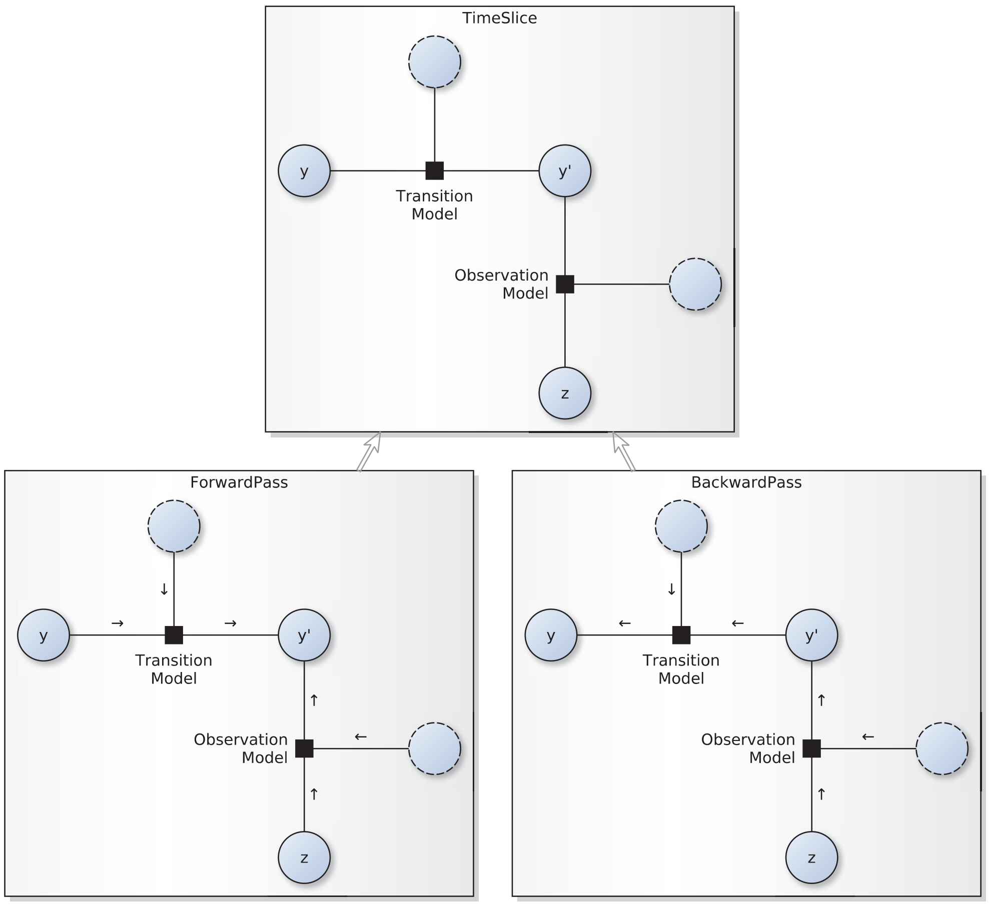


Figure 4.7: Example of the integrated inference property: The message passing schedules and update rule definitions can be left open in parent classes and defined differently in sub classes.

The integrated inference property is fulfilled by the OOFGML. Inference methods (Sec. 2.5) are defined in the processing scripts using the update rules which correspond to the exemplary rules in Sec. 2.5.4. Both are independent of the factor definitions and can be defined in sub classes of inference method independent base classes Fig. 4.7. The separation into classes can also be used to apply different inference methods on different parts of the overall model locally. Thus, message passing schedules (Sec. 2.5.3) and update rules are directly defined using the modeling language.

Generic inference methods are described in Chapter 6. A particle filter principle is applied in Chapter 7. A special modular inference method is depicted in Chapter 8.

4.2.4 Parameter Learning

In many estimation applications there are dependencies that are hardly modelable by human expert knowledge, such as for example the behavior of human traffic participants. A solution is to learn these dependencies from observations meaning the parameters of the dependency models are optimized according to a set of observed cases.

The proposed language should allow learning of dependencies from end to end but also mixing learned dependency models with other expert defined models. It should be possible to parametrize a large model and only learn a few local models inside that model. The learning methods should also be described using the language.

The following requirements are given for the parameter learning property:

- **Representation of parameters**
  Parameters have to be modeled as an estimatable variable. Learning algorithms must be able to infer the parameters from observations.

- **Variable synchronization throughout instantiations**
  Learning generalized parameters on class level from instantiated scenes has to be supported. Therefore, the variables have to be synchronized throughout the instances of one class.

- **Local and global learning**
  Machine learning has to be applicable on different levels of detail. Parameters of single dependency models have to be learnable while other dependency models use fixed parametrization from expert knowledge. Nevertheless, also parameters of large dependency models shall be learnable from end to end.

- **Inference methods for learning**
  Inference methods for parameter learning have to be definable.

The OOFGML has the capability to fulfill the parameter learning property. Parameters can be modeled as attributes of classes like any other variable as already described in Sec. 2.3. Instances have the knowledge from which class they were generated. Thus, their connection can be utilized during learning (Fig. 4.8). There is no restriction how many (parameter) variables are estimated at the same time. It only depends on the learning method. These can be defined using the same principles as for inference methods as described in Sec. 2.5 and Sec. 4.2.3. The learning algorithm itself is not focus of this work.

The learning property is used in Chapter 9 to define the behavior models for interaction estimation.

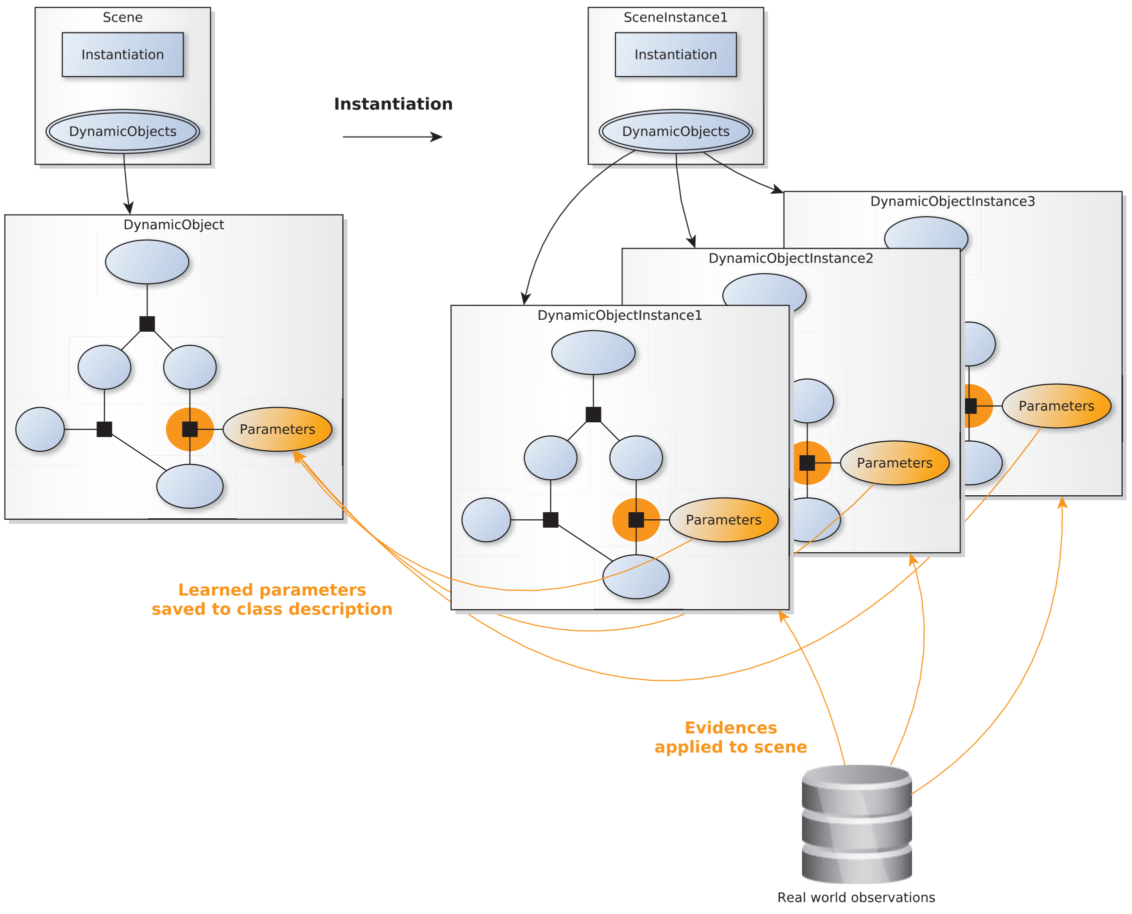


Figure 4.8: Example of the parameter learning property: Classes are instantiated to represent the real world constellation given in observations from the learning database. Model parameters are inferred and stored on class level.

4.3 Handling Real World Relationships

The language must be able to represent real world relationships. These comprise the relationship between observable and non-observable attributes, temporal relationships and relationships between different point of views. The principles are *Relation Representation*, *Observable and Hidden Variables*, *Representation of Time* and *Varying Reference Systems*.

4.3.1 Relation Representation

For understanding traffic scenes it is very important to consider the constellation of objects. The relation between dynamic objects, other dynamic objects and infrastructure elements gives much information about what traffic participants are doing and will do next.

A key objective is to estimate the existence of relationships. Therefore they should be represented in the modeling language and should be equipped with attributes that help to infer their existence probability from observations like distances between the objects.

The following requirements are given for the relations property:

- **Entities and relations**
  There needs to be a possibility to represent an entity-relationship model. Therefore classes have to be able to be used as entities and relations.
- **N-ary Relations**
  Relations must allow describing relationships between multiple entities.
- **Relations with attributes and factors**
  Relations shall be able to hold attributes and factors themselves. Properties shall not only be annotated on entities but also on relations.
- **Relations of relations**
  Relations should not only connect entities but should also be possible between existing relations.

The relation representation property is fulfilled by the OOFGML. Classes can be used to describe entities and relations. Parent classes for entites and relations can be created accordingly. Considering relations as full classes similar to entities, it is possible to allow n-ary relations, attributes in relations and relations of relations. These capabilities are comparable to the ones of the existing OPRML (Sec. 2.7).

A basic example is given in Fig. 4.9. The base class for all binary relations can be defined in a generic way:

```
☐ Entity

☐ Relation :  Entity

     ○ Entity subject

     ○ Entity object
```

Entities and relations are used as basis for the domain specific language (DSL) in Chapter 5 and thus serve as basis for all applications in Chapter 6 to Chapter 9.

4.3.2 Observable and Hidden Variables

Not all properties of a traffic situation can be observed. Many of the properties exist but are not observable by sensors. They can only be derived from observable attributes.

Generative observation models can describe how an uncertain measurement origins from a hidden (unobservable) state. These models should be definable and

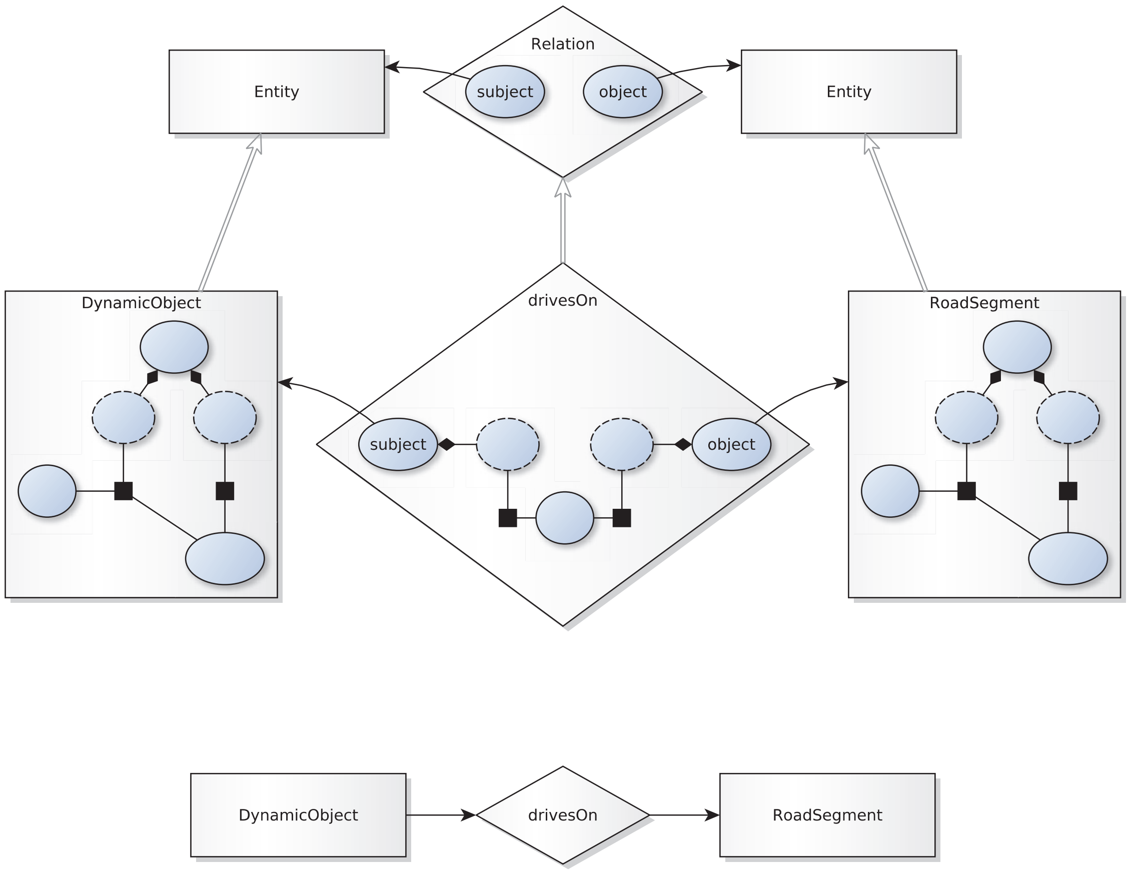


Figure 4.9: Example of the relation representation property: Entities and relations of an entity-relationship model (bottom) can be modeled using the OOFGML and generic base classes that represent entities and relations (top).

integratable in the overall dependency model such that whole existence probabilities of entities and relations can be derived from observations.

The following requirements are given for the observable and hidden variables property:

- **Observation models**
  The relation between measured observations, hidden state variable, the sensor itself and an observation model must be representable.
- **Generative modeling**
  Generative models have to be modelable to infer the inverse direction afterwards.
- **Inference of existence**
  The existence of objects, especially of relations between objects has to be able to be derived from observations.

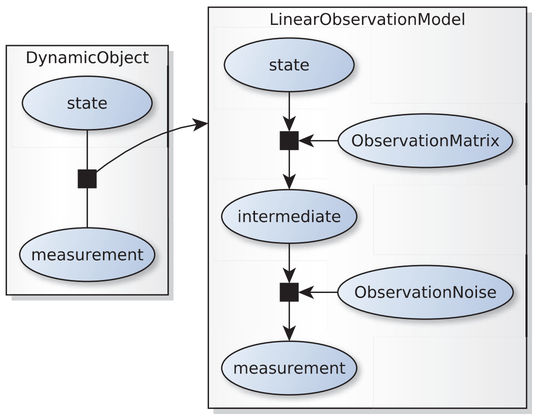


Figure 4.10: Example of the observable and hidden variables property: Factors can be used to describe observation models that define how measurements result from a given state. The arrows at the factor graph edges symbolize the Bayesian dependency.

The OOFGML has the capability to fulfill the observable and hidden variables property. Since the language is based on factor graphs (Sec. 2.4.4) all filter principles (Sec. 2.6) can be applied: Observation models can be implemented as dependencies between observable and unobservable variables. Object-orientation can be used to model the sensor and the entity to estimate as separate objects in the model with relations between them. Using causalities (Sec. 2.4.2) a generative model can be implemented describing how the measurements emerge from the hidden state. Bayesian inference allows then to estimate the hidden state from noisy observations. By adding existence attributes to entities and relations their existence probability can be integrated into the dependency model and thus observations can be used to infer the existence of entities and relations (details in Chapter 9).

A basic example is given in Fig. 4.10. Basic observation models are handled in detail in Chapter 6. Advanced models, combining several attributes are handled in Chapter 7, Chapter 8 and Chapter 9.

4.3.3 Representation of Time

Time extends the geometric world to a fourth dimensions that increases complexity enormously. All attributes in the model should also be related to a time where they are valid. Dynamic objects are only in a given state at a given time. Before and after that point in time their state can be different.

It should be possible to let attributes depend on time. This can be a dependency model to a time variable but also discrete time models using time slices should be possible. Together with the observable and hidden variables property (Sec. 4.3.2) the modeling of state space models (Sec. 2.6) should be possible.

The following requirements are given for the representation of time property:

- **Representation of time**
  Time and uncertainties in time have to be representable.
- **Temporal dependencies**
  Dependencies of variables to temporal aspects have to be modelable. This also includes transition models describing dynamics and logical sequences and processes.
- **Continuous and discrete state space models**
  Differing time models have to be easily derivable including continuous and discrete time frames.

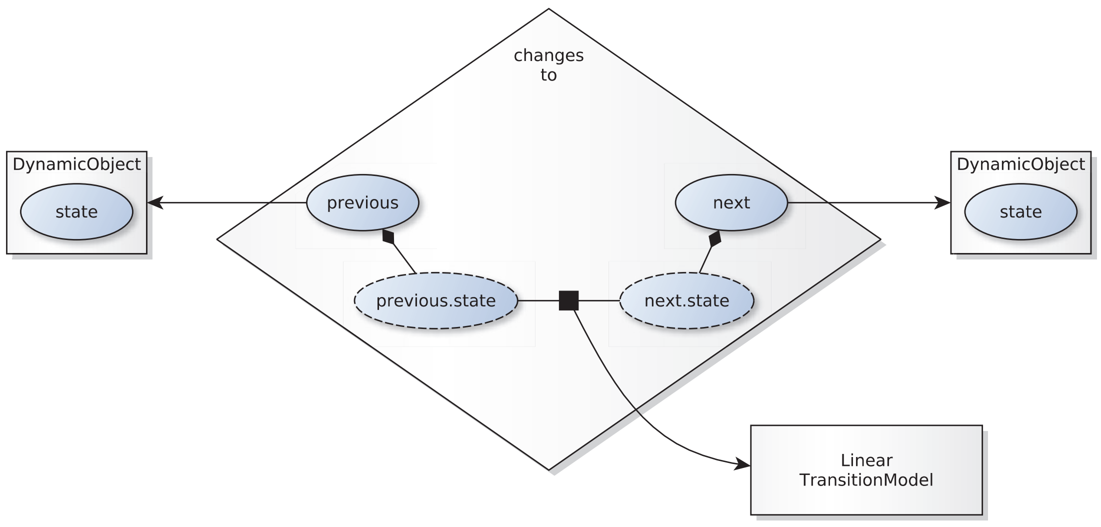


Figure 4.11: Example of the representation of time property: Temporal relations between object instances can describe transitions from time slice to time slice.

The OOFGML fulfills the representation of time property. Time can be modeled as a separate variable including the possibility to use discrete and continuous representations and arbitrary uncertainty model as described in Sec. 2.6. Other variables can depend on the time variable using factors and potential functions. Transition models can be implemented as factors deriving a future state from a source state and a change in time. Continuous and discrete state space models (Sec. 2.6.1) can be derived, e.g. discrete models by using a state representation per discrete time step and a (fixed) transition model for the transition between states (Fig. 4.11). This is the most established approach but also more advanced and generic methods (Sec. 2.6) are implementable using the language.

A class hierarchy for state space models is defined in Chapter 6. Temporal dependencies are also included in all applications (Chapter 7 - Chapter 9).

4.3.4 Varying Reference Systems

Measurements from different sensors are not always comparable. They are usually related to different coordinate systems. For example a sensor measures distances to objects relative to the sensor itself. For comparing them in a global view they have to be converted in a global reference coordinate system.

Conversions between these coordinate systems should be representable transparently in the proposed language.

The following requirements are given for the varying reference systems property:

- **Coordinate systems**
  The language has to support views from different perspectives. The same physical entities have to be describable in different coordinate systems.
- **Coordinate transformations**
  Transformations between different coordinate systems have to be integratable.

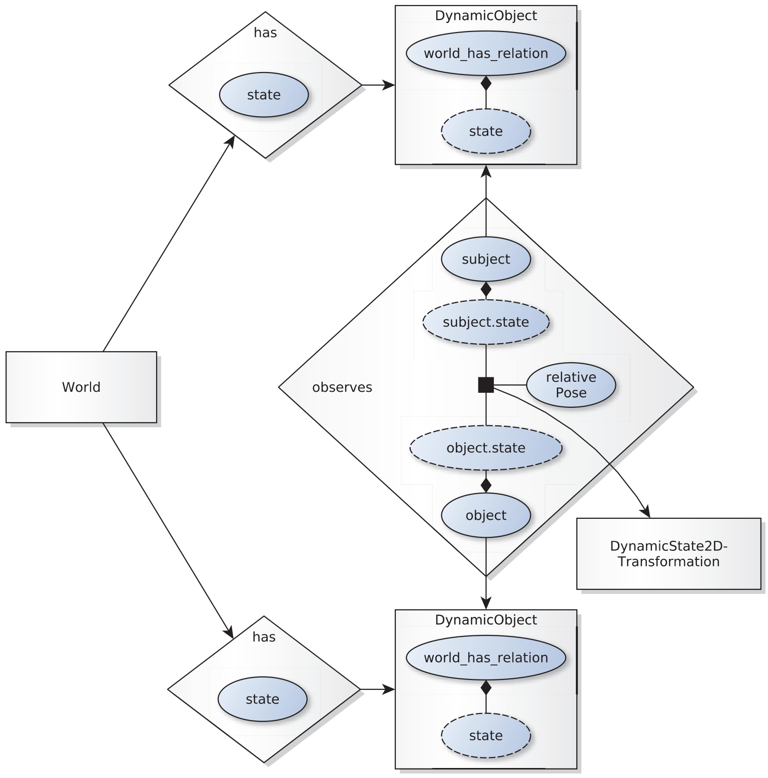


Figure 4.12: Example of the varying reference systems property: Multiple `DynamicObjects` have a global pose in the `World` but they observe each other by a relative pose. This relationship can be described by a dependency model implementing a coordinate transformation.

The OOFGML has the capability to fulfill the varying reference systems property. Different coordinate systems can be described by different relations to reference points. Transformations between coordinate systems can be described using factors to formulate local dependencies. An example with two dynamic objects is given in Fig. 4.12.

Coordinate transformations are also described in detail in Chapter 6. In the other applications they are used implicitly (Chapter 7 - Chapter 9).

4.4 Conclusion

In this Chapter the language definition was evaluated considering capability requirements originating from estimation problems. Motivated from the automated driving domain a total of 12 language properties were chosen and detailed with carefully defined requirements. These were analyzed using the definition and properties of the proposed language.

As the main theoretical evaluation it was discussed and shown that all the identified requirements can be represented in the language. Although the specification was carefully chosen it is possible that further requirements arise or the given ones have to be evaluated in more detail. Nevertheless, the OOFGML has the potential to solve the large variety of estimation tasks, especially in the automated driving domain.

In the following chapters it is shown how the language is applicable to real estimation problems in the traffic domain and how different applications can be described in a coherent way.

5 Traffic Domain Language

This chapter introduces the Traffic Domain Language (TDL) as example of specializing the generic OOFGML (Chapter 3) to a domain specific language (DSL). Besides defining the class hierarchy including entities and relations to build up an ontology (Sec. 5.1), an outlook how this Traffic Domain Language (TDL) is the basis for the actual applications (Sec. 5.2) is given which will be described in detail in the subsequent chapters. A summary is given in Sec. 5.3.

5.1 Traffic Domain

An exemplary definition of a DSL for the traffic domain is proposed. It is a basic language with focus on the applications handled in subsequent chapters. The applications are chosen to cover different aspects needed for automated driving applications and thus also the given TDL gives a good overview. Since the DSL formulates an ontology (Sec. 3.2) it is prepared for integrating other details like more sensors or different traffic participants straightforwardly.

For the ontology entities and relationships have to be defined on class level as described in Sec. 4.3.1. The given TDL focuses on binary relations and utilizes the option to build relations of relations.

5.1.1 Traffic Scenes as a Graphical Estimation Problem

Reconcile that this thesis focuses on the challenge of solving automated driving, including perceiving the environment, understanding it and making a decision how to act on the environment. Several components usually solve specific tasks as described in Sec. 2.2. They are ordered in a processing chain with a strict estimation order and only task-related information is extracted at their output interfaces. The goal is to use the OOFGML to describe these components in a more coherent way and to allow various inference directions. All tasks shall be seen as part of a holistic estimation problem as described in Sec. 2.3. This means a joint probabilistic distribution over all possible road layouts, object constellations and ego vehicle properties has to be estimated to find the most probable configuration (Fig. 5.1). The OOFGML will help to factorize and handle this large joint function.

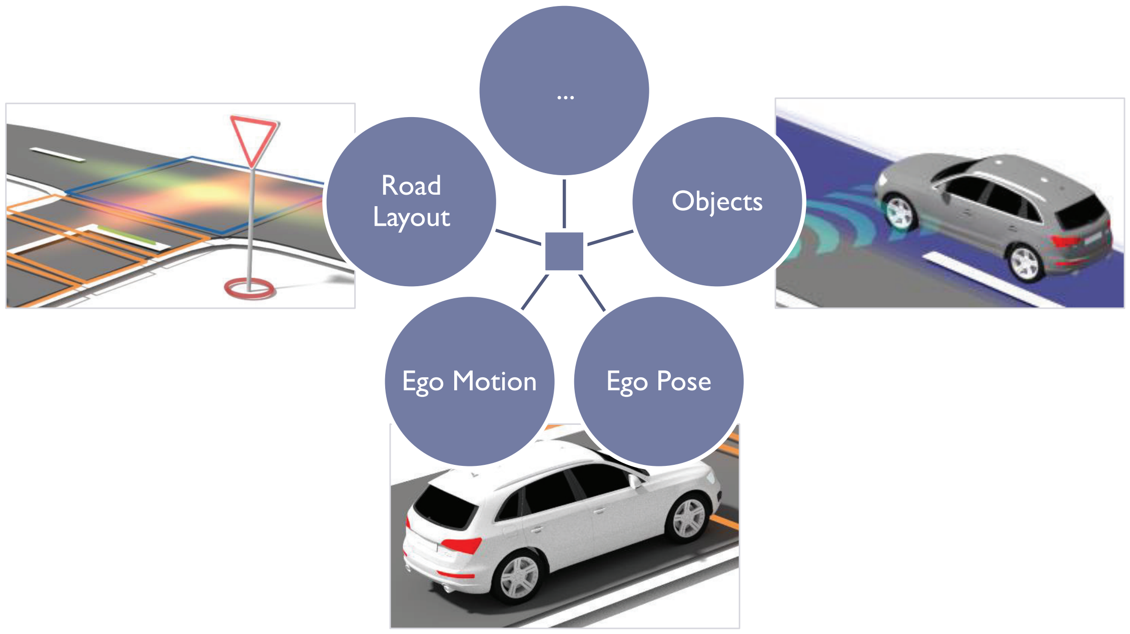


Figure 5.1: Automated driving has to be seen as one holistic estimation problem.

The DSL for the traffic domain is not only applicable to single automated vehicles. It mainly describes all entities and relationships occurring in traffic scenes. Thus, it is also valid for a variety of other applications such as multiple automated vehicles at once (communicating with each other) or traffic observing infrastructure without any automated vehicle.

5.1.2 Entities Class Hierarchy

The goal of the TDL is to represent all elements and properties of traffic scenes (that are introduced in Sec. 2.1). Thus, the entities shall include

- Real world scene objects
- Dynamics of different scene object kinds
- Hierarchical (sub) parts of the road layout
- Technical components and their specializations
- Intelligence concepts like trajectories, behaviors and routes

Fig. 5.2 shows an extract of the entity class hierarchy including exemplary entities for all these groups.

Of course, also many additional intermediate classes are possible. The chosen granularity is sufficient to show many of the possible applications. For example

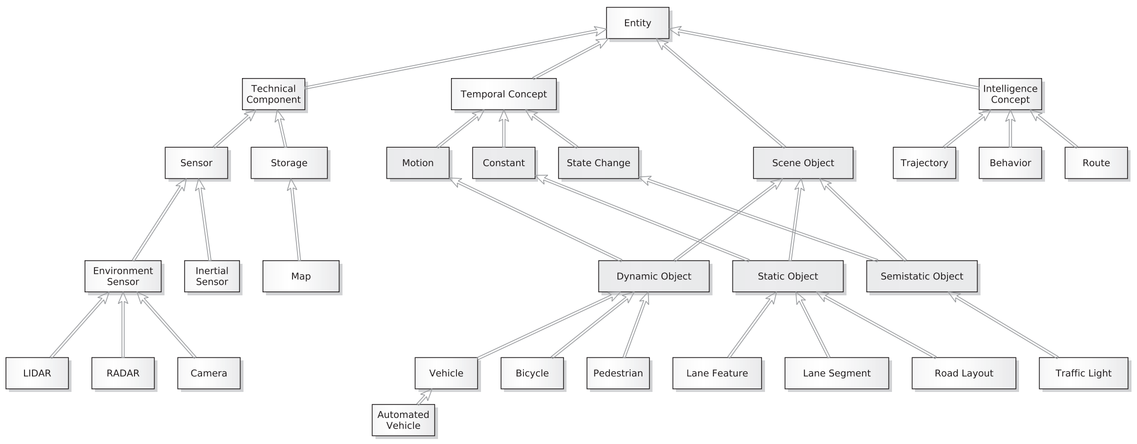


Figure 5.2: Entity class hierarchy of the TDL.

scene objects are divided into three different generic classes that additionally inherit from different temporal concepts. These temporal concepts represent different dynamic properties like moving, static or state-switching (semi-static) traffic elements. Of course also other classes could inherit from the temporal concepts. It is shown how the temporal property is brought to other classes by relations in the next sections.

5.1.3 Relations

On the class hierarchy binary relationships are defined. These are defined at parent classes as "high" as possible. Entity sub classes inherit these relations directly or use a specialized sub class of the relation.

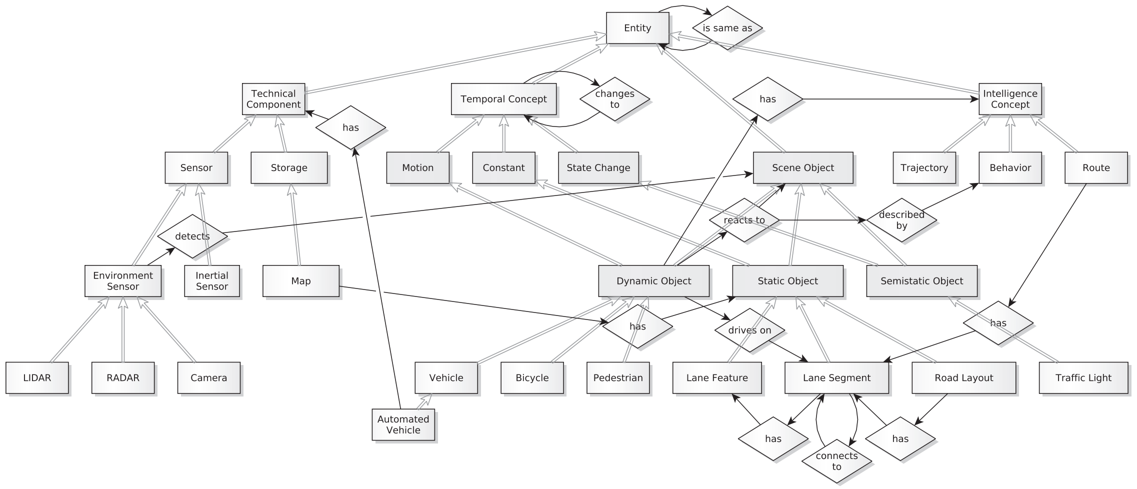


Figure 5.3: Entities and relations build up a traffic domain specific ontology.

All relations necessary for the chosen applications are shown in Fig. 5.3. Also the possibility to create relations of relations is used: Dynamic objects can `reactTo` scene objects and their reaction can again be `describedBy` behaviors.

The entities with the relationships build up a domain specific ontology. It is a class structure that can be used to describe traffic scenes including automated vehicles. For example an automated vehicle has a technical component that can be a camera environment sensor that can detect other scene objects, e.g. another dynamic object like a bicycle or a static object like a lane marking. Lane markings are part of lane segments which are again part of the road layout.

With minor modifications the ontology is compatible with the (non-probabilistic) ontologies / class hierarchies in [73] and [51]. In addition to the applications described in the subsequent chapters the ontology has been applied to the task of deriving intensional semantic relations from extensional observable relations in [10] and to reinforcement learning of driving behaviors in [15].

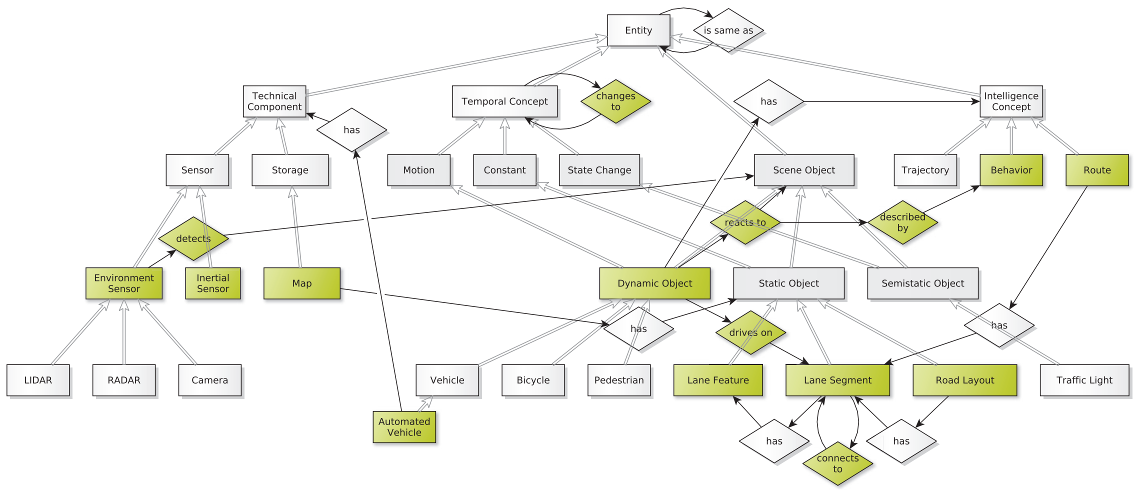
5.1.4 Compact View

Using the ontology within the OOFGML means that every class (entity and relation) can hold multiple attributes and dependencies (factors) between them. Thus the ontology is a rough separation of the overall graphical estimation problem (Sec. 5.1.1) into smaller classes. To make this connection visually more intuitive the ontology can be depicted more compactly (Fig. 5.5a).

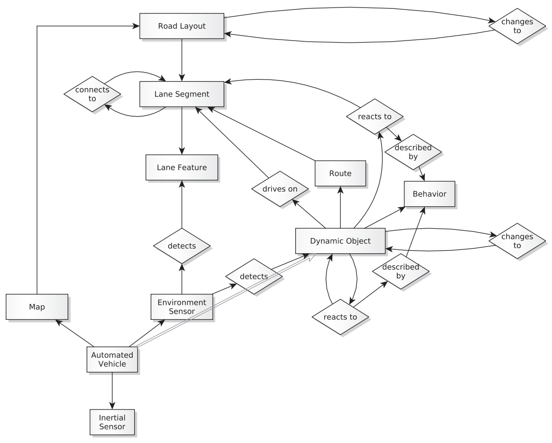
The steps taken to achieve this representation are (Fig. 5.4):

- Choose all relations.
- Select entities to be able to describe relations (e.g. `EnvironmentSensor` has to be selected to describe the `detects` relation).
- Draw `has` relations as simple arrows or as stacked boxes where possible (e.g. Road Layout has Lane Segment has Lane Feature).
- Draw relations that are defined on not represented entity parent classes onto all available sub classes (e.g. `changesto` relation on `RoadLayout` and `DynamicObject`)
- Draw parent classes as dashed boxes inside sub classes. Draw their relations only once if they are visible multiple times. (e.g. `DynamicObject` inside `AutomatedVehicle`)

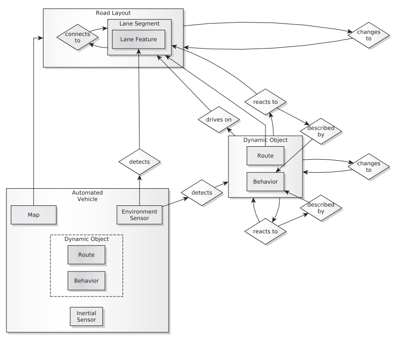
The resulting compact view shows nicely how the overall estimation problem is split into sub problems distributed over several entities and relationships (Fig. 5.5). This is a rough factorization of the overall dependency and each class can hold additional factorizations of their internal dependency model.



(a) Choose all relations and all entities that are necessary to describe them.

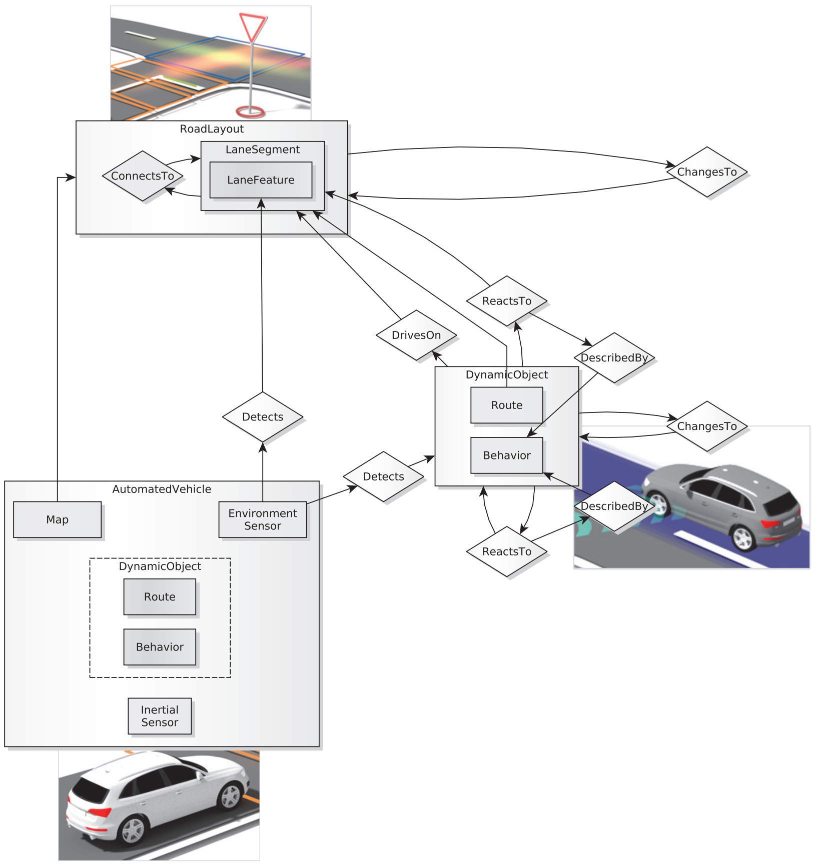


(b) Draw has relations as simple arrows and draw relations that are defined on not represented entity parent classes onto all available sub classes.

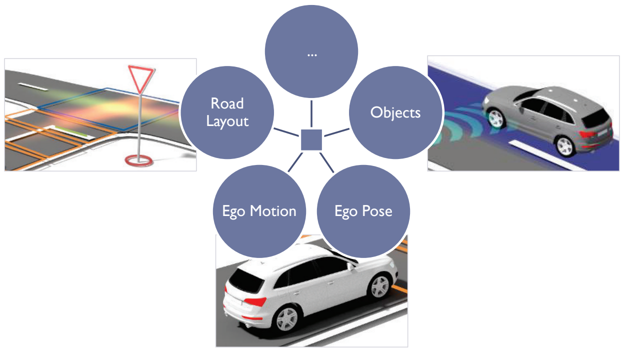


(c) Draw parent classes as dashed boxes inside sub classes and stack some has relations. Comp. Fig. 5.5a

Figure 5.4: Steps to achieve the compact view.



(a) Compact view of the ontology



(b) General estimation problem

Figure 5.5: The compact view (a) visualizes the factorization of the estimation problem (b).

The factorization induced by the ontology is only dependent on the traffic domain itself and represents the expert's understanding of the domain. Additional factorizations and adaptations will be introduced by application-specific requirements like the attributes of interest and algorithm performance. Sec. 5.2 shortly introduces specializations of exemplary applications that are described in more detail in the subsequent chapters.

5.2 Outlook to Applications

Four different exemplary applications are described in detail in the subsequent chapters. Here an outlook on how the introduced TDL will be used in these applications is given.

5.2.1 GNSS Tracking

The GNSS tracking is a simple example to show the basic usage of the OOFGML. The idea is that the noise of absolute pose measurements (e.g. from a GNSS sensor) is reduced by a temporal fusion considering the motion of the (ego) vehicle where the sensor is attached.

The ontology of the DSL can be utilized to describe the necessary components and their relationships as depicted in Fig. 5.6: The GNSS sensor is represented as a subclass of the `InertialSensor` class attached to the `AutomatedVehicle` via a `has` relation. The temporal relationship is represented by the `changes To` relation of the `DynamicObject` which is a parent class of the `Automated Vehicle`.

In Chapter 6 this ontology will be used to derive a Kalman filter for the ego vehicle pose estimation.

5.2.2 Ego Vehicle Localization

(Ego vehicle) localization is one of the basic tasks for automated vehicles. Using the ontology the ego vehicle pose can be estimated by various relationships to other elements of the traffic scene. The approach focuses on localization using a static environment map and interpreted observed dynamic objects. Key idea is that the movement of dynamic objects and their relative position to the ego vehicle give hints on where the ego vehicle is located on a geometric street map.

The ontology of the DSL is used to describe the necessary entities and their relationships as depicted in Fig. 5.7: The `AutomatedVehicle` is equipped with an `OdometrySensor` (modeled as Inertial Sensor), a `Map` and an `Environment Sensor`. The `EnvironmentSensor` can `detect` other `DynamicObjects`. All

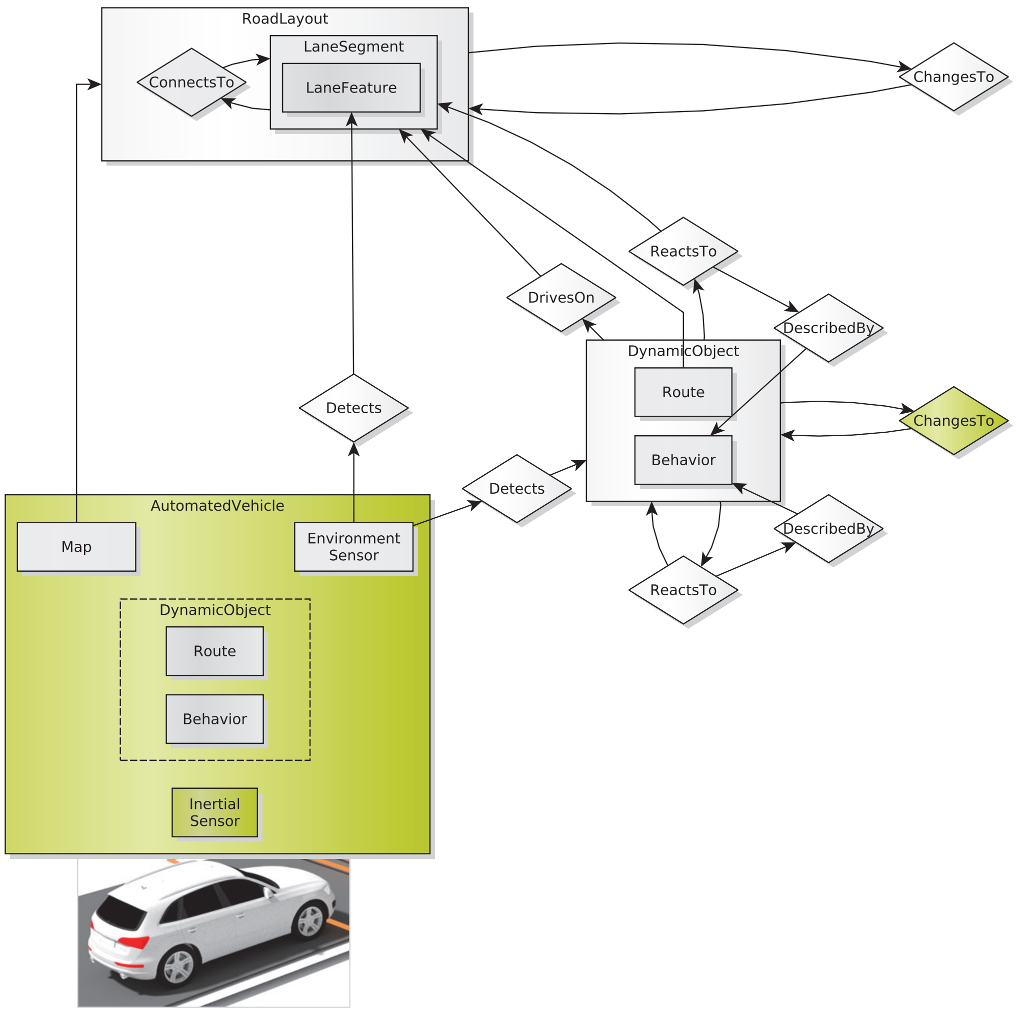


Figure 5.6: Only a few entities and relations (green) are needed to model the GNSS pose estimation application.

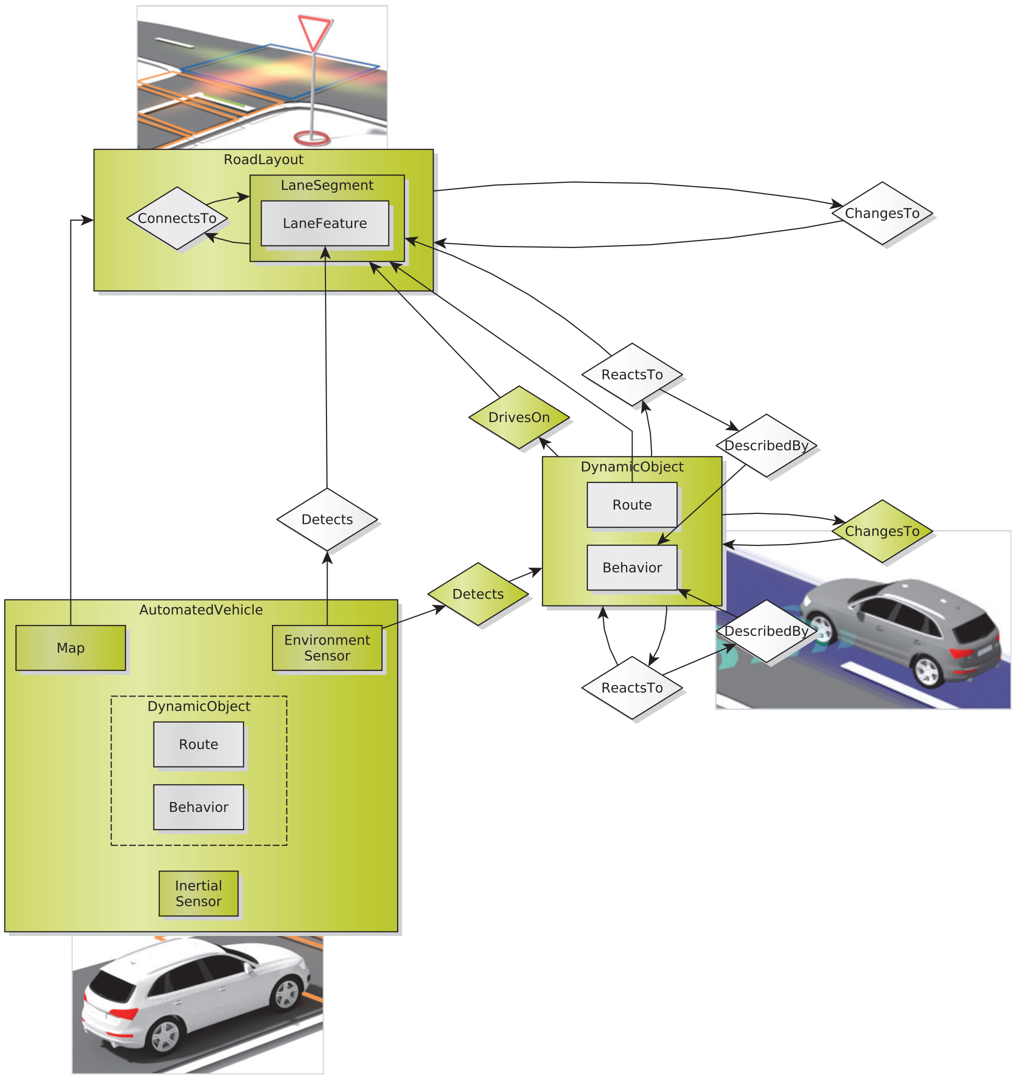


Figure 5.7: Partial traffic domain ontology (green) for ego vehicle localization using the surrounding object constellation.

DynamicObjects (including the ego vehicle) driveOn LaneSegments which are part of the RoadLayout which is stored in the Map. A temporal fusion can be described by the changesTo relation like in the GNSS pose estimation example (Sec. 5.2.1).

In Chapter 7 this ontology will be used to derive a particle-based filter for the ego vehicle pose estimation.

5.2.3 Road Layout Estimation

The estimation of static environment elements is easier than estimating dynamic objects (because of the simpler temporal relation) but is often used as additional input to dynamic object estimation. Thus, a precise estimation of static objects is beneficial. One approach is to utilize the relation between observable features to derive an overall consistent road layout.

The ontology of the DSL is used to describe the necessary entities and their relationships as depicted in Fig. 5.8: The AutomatedVehicle is equipped with an OdometrySensor (modeled as Inertial Sensor) and an EnvironmentSensor which detects observable LaneFeatures. LaneFeatures, LaneSegments and RoadLayout are organized in a hierarchical structure built of has and connectsTo relations. Two temporal relations are integrated: The changesTo relation on the AutomatedVehicle represents its movement and the same relation on the RoadLayout represents its constant position over time.

In Chapter 8 this ontology will be used to implement a grid-based filter on small scale vehicles in the Audi Autonomous Driving Cup.

5.2.4 Route, Behavior and Trajectory Estimation

Estimating the route, behavior and (future) trajectory of dynamic objects is one of the most challenging problems to solve for successful automated driving. While basic trajectory prediction can already be accomplished by existing object tracking algorithms, the estimation of routes and different behaviors requires much more knowledge on the relationships between the dynamic and static objects in the scene. The idea is to use the observations at the current time to estimate interactions between traffic participants, derive behaviors and routes and extrapolate a more precise trajectory prediction.

The ontology of the DSL is used to describe the necessary entities and their relationships as depicted in Fig. 5.9: The AutomatedVehicle uses an EnvironmentSensor to detect DynamicObjects while the static RoadLayout is read from a Map storage. The focus lies on the intelligent concepts of DynamicObjects: They can reactTo the properties of LaneSegments (e.g. curves) and other

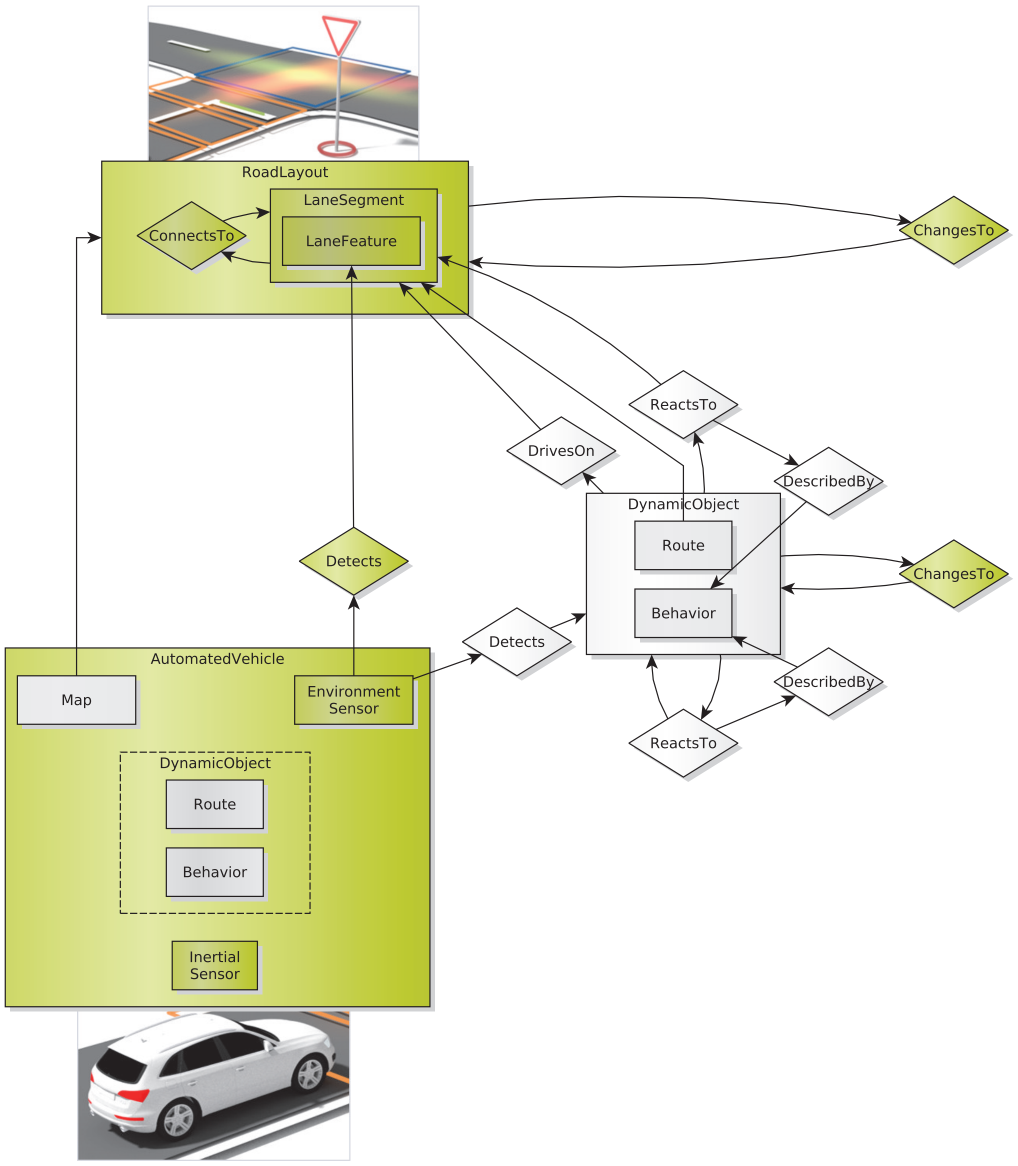


Figure 5.8: Partial traffic domain ontology (green) for road layout estimation.

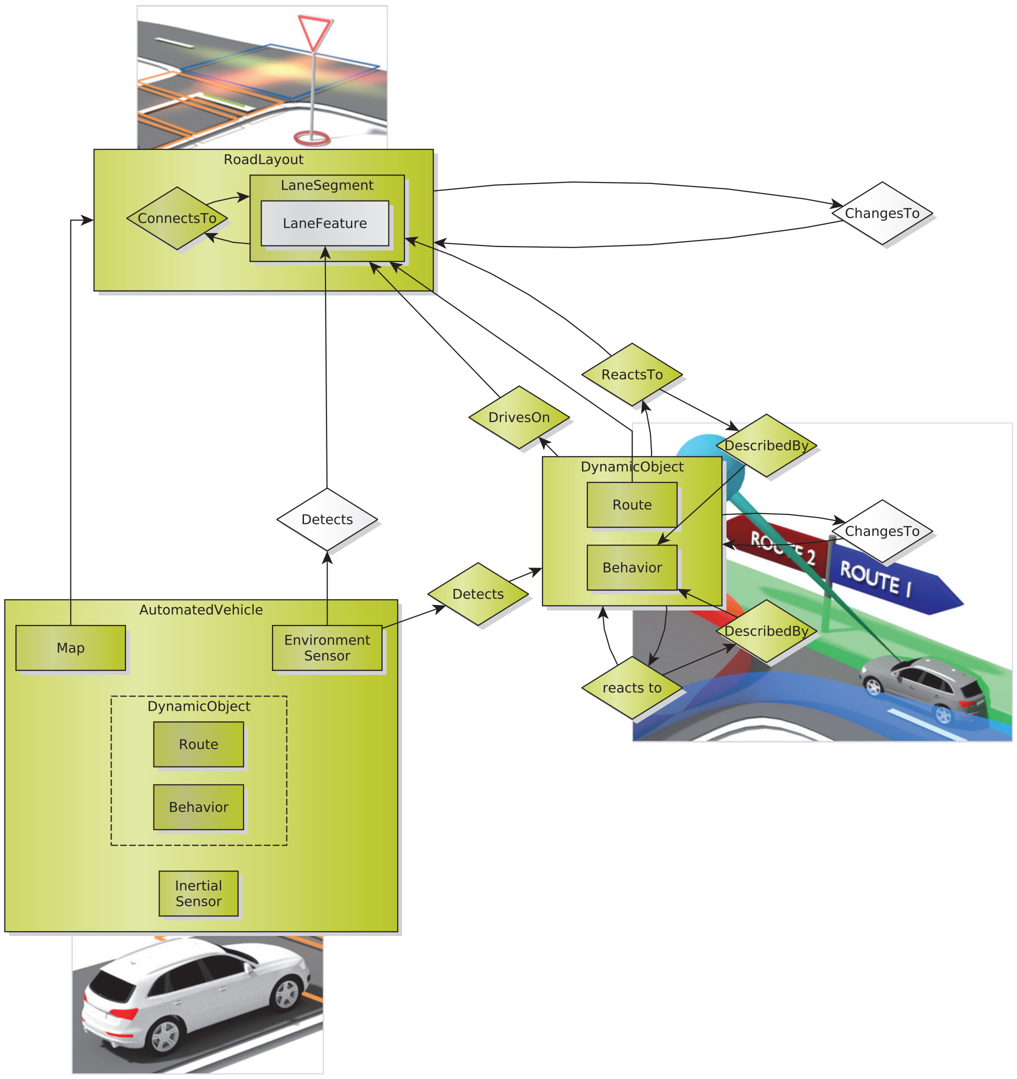


Figure 5.9: Partial traffic domain ontology (green) for route, behavior and trajectory estimation.

DynamicObjects. Both interactions can be described by Behaviors. The behaviors influence the object state which is hidden in the has relation.

In Chapter 9 this ontology is the basis for two different approaches of estimating routes, behaviors and trajectories. Additional entities and relations will be introduced there.

5.3 Summary

The key points of this chapter are:

- A domain specific language (DSL) can be created by introducing domain-specific entities and relations to the generic OOFGML.
- The entities and relations build up a domain-specific probabilistic ontology.
- The DSL for the traffic domain can be depicted as a compact view highlighting the components automated vehicle, dynamic objects and road layout.
- The DSL for the traffic domain is a basic factorization of the estimation problem.
- Application-specific ontologies can be derived from the DSL for the traffic domain.

Details on the application-specific adaptations of the DSL are given in the subsequent chapters.

6 Basic Example: GNSS Tracking

This chapter gives an easily understandable basic example how the OOFGML can be used to model an application. After clarifying the application goal (Sec. 6.1), general domain-independent classes and attributes are introduced (Sec. 6.2) and then applied to the given application (Sec. 6.3). These steps are similar to the ones in the subsequent applications but handled in more detail. Many detailed class definitions use the notation introduced in Sec. 3.5. Subsequently, the different possible implementation opportunities are discussed (Sec. 6.4) before Sec. 6.5 gives a conclusion.

6.1 Application Goal

For the basic example a very elementary application is chosen. The ego vehicle pose has to be estimated using absolute position measurements from a GNSS sensor. Better algorithms consider GNSS raw data from single satellites to also estimate clutter and scattered satellite signals but a very basic setup is intended here. The goal is not to compete against the state of the art but to show how a rather simple application is modeled using the OOFGML. This understanding is the basis for all subsequent applications. It is assumed that the GNSS measurement is already converted in a metric coordinate system, e.g. Universal Transverse Mercator (UTM) [60].

In this application the goal is to reduce the uncertainty of the noisy measurements by running a temporal filter respecting the possible vehicle movement. This is a simple ego vehicle localization task that can be seen as single target tracking (in contrast to multi target tracking in Sec. 2.6.6). The association of measurements is fixed since it is clear that every measurement corresponds to the ego vehicle. Thus, no association problem has to be solved.

The application can be solved with a simple state-space model and a discrete time filter algorithm such as a Kalman or particle filter (Sec. 2.6). The application is used to illustrate the steps that have to be taken to get from the DSL to the actual Kalman filter implementation.

6.2 General Domain-Independent Classes and Attributes

Based on the findings from Chapter 2, a class hierarchy can be constructed that represents all knowledge about message representations and inference methods including state space models like established filters as described in Sec. 2.6.

The here defined class hierarchy is not meant to be complete. It shows the usage of the OOFGML at the example of the Kalman filter (Sec. 2.6.3). Other filters can be integrated analogously. Hints to other message representations and models are given at relevant places.

6.2.1 General Attribute Representation Classes

At first the different message representations Sec. 2.5.4 have to be represented in the class structure. Fig. 6.1 shows the basic classification hierarchy. These classes have (almost) no member definition. They are solely meant for classifying later message definitions.

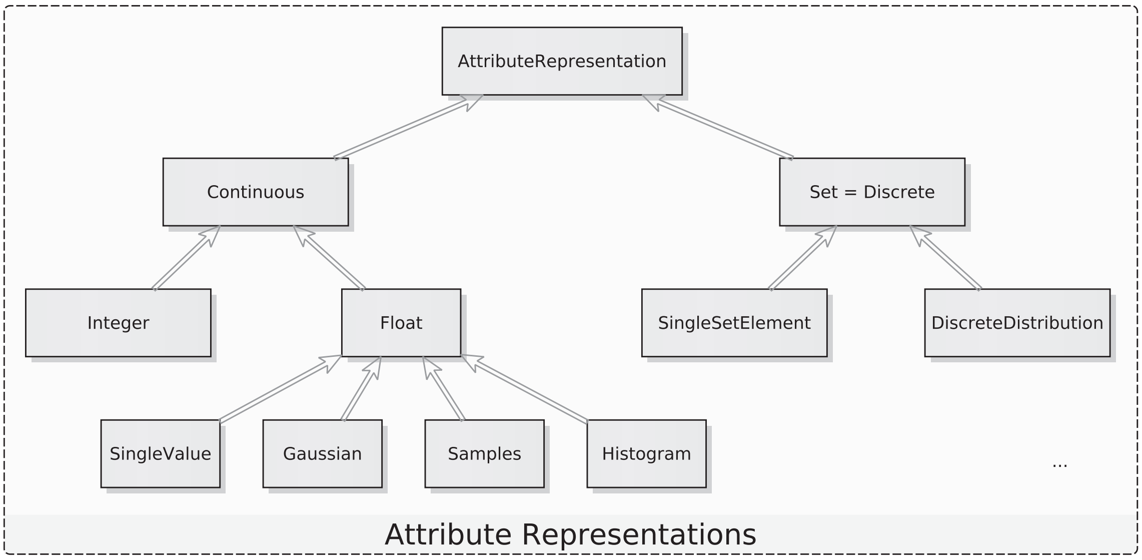


Figure 6.1: Class hierarchy representing the different message types and approximations. These classes are semantic identifiers and do not specify dimensions. Hierarchies for scalar and vector based sub classes can be depicted analogously.

Exemplary class definitions are given:

- ☐ `Float`
- ☐ `SingleValue : Float`
    - ◯ `value`

- ☐ `Gaussian : Float`
    - ○ `mean`
    - ○ `variance`

The message representations `SingleValue` and `Gaussian` for continuous messages include already a name definition for their usual members: Single value float approximations approximate a float by a single value stored in the *value* attribute while Gaussians use a *mean* and *variance*. An attribute λ implemented as Gaussian can be seen as a vector consisting of these (sub) attributes:

$$\lambda = \begin{pmatrix} \text{mean} \\ \text{variance} \end{pmatrix} \tag{6.1}$$

Note that the dimension of mean and variance are not defined. Only the names of these attributes are set.

From these basic classifications and semantics, specific implementations can be derived for scalar and vector based attributes. Based on a `FloatScalar` the exemplary sub classes `SingleValueFloatScalar` and `GaussianApproximated FloatScalar` can be derived:

- ☐ `FloatScalar : Float, Scalar`
- ☐ `SingleValueFloatScalar : FloatScalar, SingleValue`
    - ○ `FloatBasetype value`
- ☐ `GaussianApproximatedFloatScalar : FloatScalar, Gaussian`
    - ○ `FloatBasetype` μ `(=mean)`
    - ○ `FloatBasetype` σ^2 `(=variance)`

The `GaussianApproximatedFloatScalar` inherits the declaration of mean and variance and defines them as using a single `FloatBasetype`. Additionally aliases (Sec. 3.5) μ and σ^2 are introduced that are typical for one-dimensional Gaussians.

Analogously the representations for `FloatVector` can be defined:

- ☐ `FloatVector : Float, Vector, FloatScalar[]`
- ☐ `SingleValueFloatVector : FloatVector, SingleValue`
    - ○ `FloatBasetype[] value`
- ☐ `GaussianApproximatedFloatVector : FloatVector, Gaussian`
    - ○ `FloatBasetype[]` μ `(=mean)`
    - ○ `FloatBasetype[][]` Σ `(=variance)`

Here, the brackets `[]` denote a vector (array) of the given type.[1]

[1]Dimensions of these vectors have to match. For convenience these conditions are not explicitly depicted in the notation here.

Based on the `FloatVector` definition application-specific named vectors can be derived, e.g. for 2D poses:

- ☐ `Translation2D : FloatVector`
  - ◯ `FloatScalar x (=FloatVector[0])`
  - ◯ `FloatScalar y (=FloatVector[1])`
- ☐ `Velocity2D : FloatVector`
  - ◯ `FloatScalar x (=FloatVector[0])`
  - ◯ `FloatScalar y (=FloatVector[1])`
- ☐ `DynamicState2D : FloatVector`
  - ◯ `Translation2D position (=FloatVector[0:1])`
  - ◯ `Velocity2D velocity (=FloatVector[2:3])`
- ☐ `Rotation2D : FloatVector`
  - ◯ `FloatScalar yaw (=FloatVector[0])`
- ☐ `RotationMatrix2D : FloatMatrix`
- ☐ `Pose2D`
  - ◯ `RotationMatrix2D rotation`
  - ◯ `Translation2D translation`

The notation `FloatVector[i]` corresponds to the i-th element in the vector and `FloatVector[i:j]` to a sub vector with elements from i to j.

Following this definition, the `DynamicState2D` class can be used as generic `FloatVector` but also as a specific class with named members *position* and *velocity*. The named members are useful for `DynamicState2D`-specific dependency formulations that are easily readable. The generic `FloatVector` interface is beneficial if the `DynamicState2D` type is used in a generic module such as a filter. There, a transition or observation matrix can operate on the vector. Exemplary matrix definitions are[2]:

- ☐ `ConstantVelocityTransitionMatrix2D : TransitionMatrix`
  - ◯ `FloatScalar` Δt
  - ◯ `TransitionMatrix` $= \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- ☐ `PositionObservationMatrix2D : ObservationMatrix`
  - ◯ `ObservationMatrix` $= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

[2]Assuming corresponding parent classes `TransitionMatrix` and `ObservationMatrix` are given and defined analogously to the `FloatVector` class.

`ConstantVelocityTransitionMatrix2D` predicts a future `DynamicState2D` from a given `DynamicState2D` assuming constant velocity over a time horizon Δt. `PositionObservationMatrix2D` implements a mapping to an observation space where only the 2D position can be observed. These matrices will later be used in the `ConstantVelocityTransitionModel` and the `PositionObservationModel`.

6.2.2 Update Rules

After having defined basic message representation classes representation-specific update rules can follow. Every elementary factor from the tables in Sec. 2.5.4 is defined as a generic class interface including a factor with dependency model formulation. Representation-specific sub classes can then define special update rules. This is exemplarily illustrated using the Gaussian update rules from Table 2.2 and Table 2.3.

The names for the rules are:

`Equals2Node` A factor mapping two attributes to be the same.

`Equals3Node` A factor mapping three attributes to be the same. (Comp. rule 1 in Table 2.2)

`PlusNode` A factor building the sum of two attributes. (Comp. rule 2 in Table 2.2)

`MatrixMultiplicationNode` A factor multiplying a `Matrix`-Attribute to a `FloatVector`-Attribute. (Comp. rule 5 in Table 2.2)

`EqualsMatrixCompositeNode` A composite factor combining an `Equals3Node` with a `MatrixMultiplicationNode`. (Comp. rule 1 in Table 2.3)

`MatrixPlusCompositeNode` A composite factor combining a `MatrixMultiplicationNode` with a `PlusNode`. (Comp. rule 2 in Table 2.3)

Note that all update rules of Table 2.2 and Table 2.3 are defined here in classes.[3] Analog base classes can be defined for the update rules in Table 2.4 - 2.5 if they do not already match these base classes.

From the base classes representation specific sub classes are derived implementing the update rules. Their class name has a prefix according to their message approximation, e.g. `Gaussian*` or `Samples*`, for instance `GaussianEquals3Node` or `GaussianEqualsMatrixCompositeNode`.

[3]The prior node (rule 3 in Table 2.3) can be implemented using evidences on a `Gaussian` attribute.

The `Equals2Node` update rules ($\rightarrow$) can already be defined on the base class; Approximation specific sub classes are not necessary:

- □ `Equals2Node`
  - ○ `x`
  - ○ `y`
  - ■ `equals(x,y,z)` $\delta(x = y)$
  - $\rightarrow$ $u_y(x) = x$
  - $\rightarrow$ $u_x(y) = y$

The base class and Gaussian sub class for `Equals3Node`, `PlusNode` and `Matrix MultiplicationNode` are given as follows:

- □ `Equals3Node`
  - ○ `x`
  - ○ `y`
  - ○ `z`
  - ■ `equals(x,y,z)` $\delta(x = y = z)$
- □ `GaussianEquals3Node : Equals3Node`
  - ○ `GaussianApproximatedFloatVector x`
  - ○ `GaussianApproximatedFloatVector y`
  - ○ `GaussianApproximatedFloatVector z`
  - ■ `equals(x,y,z)` $\delta(x = y = z)$
  - $\rightarrow$ $u_z(x,y) = \begin{pmatrix} z.\mu \\ z.\Sigma \end{pmatrix} = \begin{pmatrix} (x.\Sigma^{-1} + y.\Sigma^{-1})^{\mathsf{T}}(x.\Sigma^{-1}x.\mu + y.\Sigma^{-1}y.\mu) \\ x.\Sigma(x.\Sigma + y.\Sigma)^{\mathsf{T}}y.\Sigma \end{pmatrix}$
- □ `PlusNode`
  - ○ x
  - ○ y
  - ○ z
  - ■ `plus(x,y,z)` $\delta(x + y = z)$
- □ `GaussianPlusNode : PlusNode`
  - ○ `GaussianApproximatedFloatVector` x
  - ○ `GaussianApproximatedFloatVector` y
  - ○ `GaussianApproximatedFloatVector` z
  - ■ `plus(x,y,z)` $\delta(x + y = z)$
  - $\rightarrow$ $u_z(x,y) = \begin{pmatrix} z.\mu \\ z.\Sigma \end{pmatrix} = \begin{pmatrix} x.\mu + y.\mu \\ x.\Sigma + y.\Sigma \end{pmatrix}$
- □ `MatrixMultiplication`
  - ○ `FloatVector` x
  - ○ `Matrix` a
  - ○ `FloatVector` y
  - ■ `multiply(`x, a, y`)` $\delta(a * x = y)$

☐ `GaussianMatrixMultiplication : MatrixMultiplication`

- ○ `GaussianApproximatedFloatVector` x
- ○ `Matrix` a
- ○ `GaussianApproximatedFloatVector` y
- ■ `plus(`x`,`a`,`y`)` $\delta(a * x = y)$
- $\rightarrow\ u_y(x, a) = \begin{pmatrix} y.\mu \\ y.\Sigma \end{pmatrix} = \begin{pmatrix} ax.\mu \\ ax.\Sigma a^H \end{pmatrix}$
- $\rightarrow\ u_x(y, a) = \begin{pmatrix} x.\mu \\ x.\Sigma \end{pmatrix} = \begin{pmatrix} a^{-1}y.\mu \\ a^{-1}y.\Sigma a^{-\mathsf{T}} \end{pmatrix}$

It is recognizable that the update rules for different inference directions can be implemented in the same class (which defines the dependency model).

As an example for composite update rules a deeper look into the `EqualsMatrixCompositeNode` is taken which is the basis for the elementary Kalman filter method that can be modeled as a separate factor graph (Fig. 2.12).

The class definition consists of four classes:

☐ `EqualsMatrixCompositeNode`

- ○ `GaussianApproximatedFloatVector` x
- ○ `GaussianApproximatedFloatVector` y
- ○ `GaussianApproximatedFloatVector` z
- ○ `GaussianApproximatedFloatVector intermediate`
- ○ `Matrix` a
- ■ `Equals3Node equals(`x`,intermediate,`z`)`
- ■ `MatrixMultiplication multiply(intermediate,`a`,`y`)`

☐ `GaussianEqualsMatrixCompositeNode : EqualsMatrixCompositeNode`

- ○ `GaussianApproximatedFloatVector` x
- ○ `GaussianApproximatedFloatVector` y
- ○ `GaussianApproximatedFloatVector` z
- ○ `GaussianApproximatedFloatVector intermediate`
- ○ `Matrix` a
- ■ `GaussianEquals3Node equals(`x`,intermediate,`z`)`
- ■ `GaussianMatrixMultiplication multiply(intermediate,`a`,`y`)`
- $\rightarrow\ u_y(x, z, a) = \texttt{equals}.u_y(x, z), \texttt{multiply}.u_y(x, a)$
- $\rightarrow\ u_z(x, y, a) = \texttt{KalmanUpdateCompositeNode}(x, y, z, a) :: u_z(x, y, a)$

☐ `KalmanUpdateCompositeNode`

- ○ `GaussianApproximatedFloatVector` x
- ○ `GaussianApproximatedFloatVector` y
- ○ `GaussianApproximatedFloatVector` z
- ○ `Matrix` a
- ○ `GaussianApproximatedFloatVector` x_1
- ○ `GaussianApproximatedFloatVector` x_2

  - ○ `GaussianApproximatedFloatVector` y_1
  - ○ `GaussianApproximatedFloatVector` Δy
  - ■ `GaussianEquals3Node equals_x`(x,x_1,x_2)
  - ■ `GaussianMatrixMultiplication multiply`(x_1,a,y_1)
  - ■ `GaussianDifference diff_y`(y,y_1,Δy)
  - ■ `KalmanUpdateNode kalman`(x_2,a,Δy,z)
  - → $u_{\Delta y}(x,a,y) = \texttt{equals\_x}.u_y(x), \texttt{multiply}.u_y(x,a), \texttt{diff\_y}.u_z(x,y)$
  - → $u_z(x,a,y) = u_{\Delta y}(x,a,y), \texttt{equals\_x}.u_z(x), \texttt{kalman}.u_z(x,a,y)$

- □ `KalmanUpdateNode`
  - ○ `GaussianApproximatedFloatVector` x
  - ○ `GaussianApproximatedFloatVector` y
  - ○ `GaussianApproximatedFloatVector` z
  - ○ `Matrix` a
  - ■ `kalman(x,a,y,z)`
  - → $u_z(x,a,y) = \begin{pmatrix} z.\mu \\ z.\Sigma \end{pmatrix} = \begin{pmatrix} x.\mu + Ky.\mu \\ x.\Sigma - Kax.\Sigma \end{pmatrix} = \begin{pmatrix} x.\mu + Ky.\mu \\ (I - Ka)x.\Sigma \end{pmatrix}$
    `with` $K := x.\Sigma a^{\mathsf{T}} y.\Sigma^{-1}$

While `EqualsMatrixCompositeNode` is the Gaussian-independent base class, `GaussianEqualsMatrixCompositeNode` corresponds to the specialization to Gaussian message representations. Inference of y from x, z and a is straightforwardly implemented in $u_y(x,z,a)$. The update rule for the opposite direction $u_z(x,y,a)$ refers to a separate class `KalmanUpdateCompositeNode` that models the specialized inference graph. It uses other update rule implementation classes and a special class `KalmanUpdateNode` that corresponds to the update rule given in Equ. 2.106. The update rule is split into two parts allowing also the explicit inference of Δy needed for IMM filter implementation (Sec. 2.6.5).

6.2.3 Basic Dependency Models

On the way to the definition of state space models basic dependency models are needed. Fig. 6.2 shows a class hierarchy based on the base class `MappingWithParameters`.

`MappingWithParameters` can be seen as a very basic class interface describing all estimation problems (Sec. 2.3):

- □ `MappingWithParameters`
  - ○ `MultiValueAttribute in`
  - ○ `MultiValueAttribute out`
  - ○ `MultiValueAttribute parameters`

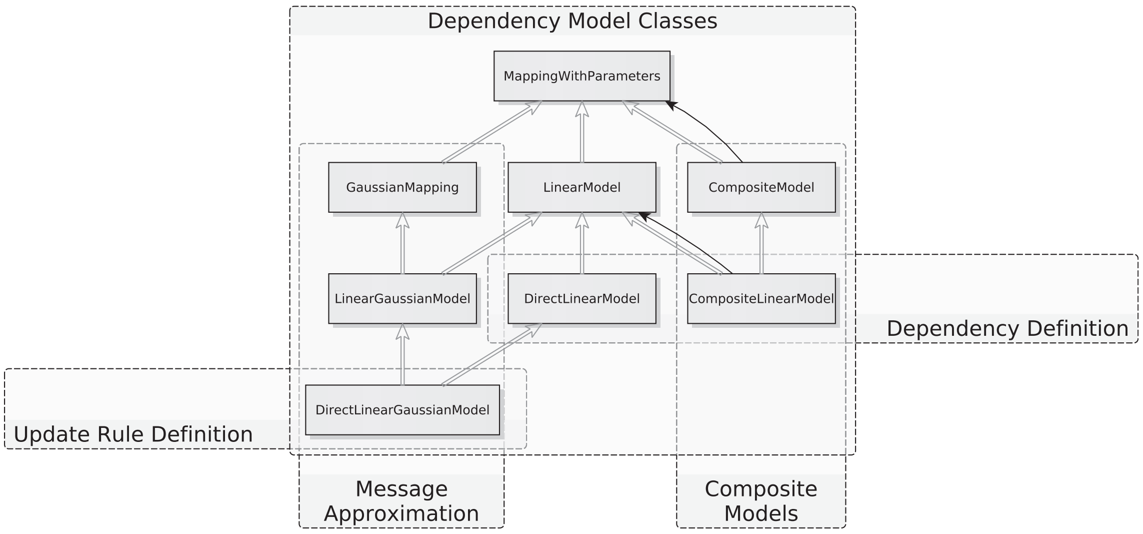


Figure 6.2: Generic dependency models, approximations and update rule definitions.

Observed variables (*in*) are used to derive unobservable variables (*out*) with the help of given model parameters (*parameters*). The same mapping can also be used to learn parameters from annotated observations (*in* and *out* given).

As already described in Sec. 2.3, complex estimation problems can also be solved by factorizing into smaller sub problems. This approach can be represented by a generic `CompositeModel` derived from the `MappingWithParameters` class and combining two `MappingWithParameters` internally:

```
□ CompositeModel :  MappingWithParameters

    ○ MultiValueAttribute in

    ○ MultiValueAttribute out

    ○ MultiValueAttribute parameters

    ○ MultiValueAttribute intermediate

    ○ MultiValueAttribute parameters1

    ○ MultiValueAttribute parameters2

    ■ MappingWithParameters model1(in, parameters1, intermediate)

    ■ MappingWithParameters model2(intermediate, parameters2, out)

    ■ param_combination(parameters1, parameters2, parameters)
```

The generic `MappingWithParameters` can be specialized in two different ways, either by

- restricting the mapping description including the *parameters*, or
- restricting the *in* and *out* attributes.

The first restriction is illustrated by the example of linear models and the second restriction by Gaussian attribute representations.

Linear models are a special sub class of mappings that use a linear function as mapping description. This linear function can be described using the basic factor classes from Sec. 6.2.2. The definition of linear models uses 3 classes:

- □ `LinearModelParameters`
  - ○ `Matrix` a
  - ○ `MultiValueAttribute` b
- □ `LinearModel : MappingWithParameters`
  - ○ `MultiValueAttribute in`
  - ○ `MultiValueAttribute out`
  - ○ `LinearModelParameters parameters`
- □ `DirectLinearModel : LinearModel`
  - ○ `MultiValueAttribute in`
  - ○ `MultiValueAttribute out`
  - ○ `LinearModelParameters parameters`
  - ○ `MultiValueAttribute intermediate`
  - ■ `MatrixMultiplication multiply(in, parameters.`a`, intermediate)`
  - ■ `PlusNode plus(intermediate, parameters.`b`, out)`

The actual dependency description is given in `DirectLinearModel`. There the `MatrixMultiplicationNode` and `PlusNode` are used to implement the linear model. The interface is already given in the parent class `LinearModel` which does not imply any factor to be able to implement the linear model also in a different way. It restricts the parameters already to `LinearModelParameters` which consist of the `Matrix` a and `MultiValueAttribute` b.

An alternative to the `DirectLinearModel` is the `CompositeLinearModel` which refers to two `LinearModels` internally. It can easily be implemented by inheriting from the generic `CompositeModel`:

- □ `CompositeLinearModel : CompositeModel, LinearModel`
  - ○ `MultiValueAttribute in`
  - ○ `MultiValueAttribute out`
  - ○ `LinearModelParameters parameters`
  - ○ `MultiValueAttribute intermediate`
  - ○ `LinearModelParameters parameters1`
  - ○ `LinearModelParameters parameters2`
  - ■ `LinearModel model1(in, parameters1, intermediate)`
  - ■ `LinearModel model2(intermediate, parameters2, out)`
  - ■ `LinearModelParameterCombination param_combination(parameters1, parameters2, parameters)`

Orthogonally to linear models, Gaussian mappings use a Gaussian attribute approximation for the *in* and *out* variables and do not restrict the dependency model (and *parameters*):

```
□ GaussianMapping :  MappingWithParameters

    ○ GaussianApproximatedFloatVector in

    ○ GaussianApproximatedFloatVector out

    ○ MultiValueAttribute parameters
```

The specialization to `GaussianMappings` and `LinearModels` can be combined in a `LinearGaussianModel` resulting in 3 subclasses:

```
□ LinearGaussianModelParameters :  LinearModelParameters

    ○ Matrix a

    ○ GaussianApproximatedFloatVector b

□ LinearGaussianModel :  LinearModel, GaussianMapping

    ○ GaussianApproximatedFloatVector in

    ○ GaussianApproximatedFloatVector out

    ○ LinearGaussianModelParameters parameters

□ DirectLinearGaussianModel :  DirectLinearModel, LinearGaussianModel

    ○ GaussianApproximatedFloatVector in

    ○ GaussianApproximatedFloatVector out

    ○ GaussianApproximatedFloatVector intermediate

    ○ LinearGaussianModelParameters parameters

    ■ GaussianMatrixMultiplication multiply(in, parameters.a, intermediate)

    ■ GaussianPlusNode plus(intermediate, parameters.b, out)
```

The `DirectLinearGaussianModel` uses the `GaussianMatrixMultiplication` and `GaussianPlusNode` sub classes that include the update rule definitions for Gaussian messages (Sec. 6.2.2).

These models are the basis for the later Kalman filter classes that will additionally include the special update rule defined by `KalmanUpdateCompositeNode`.

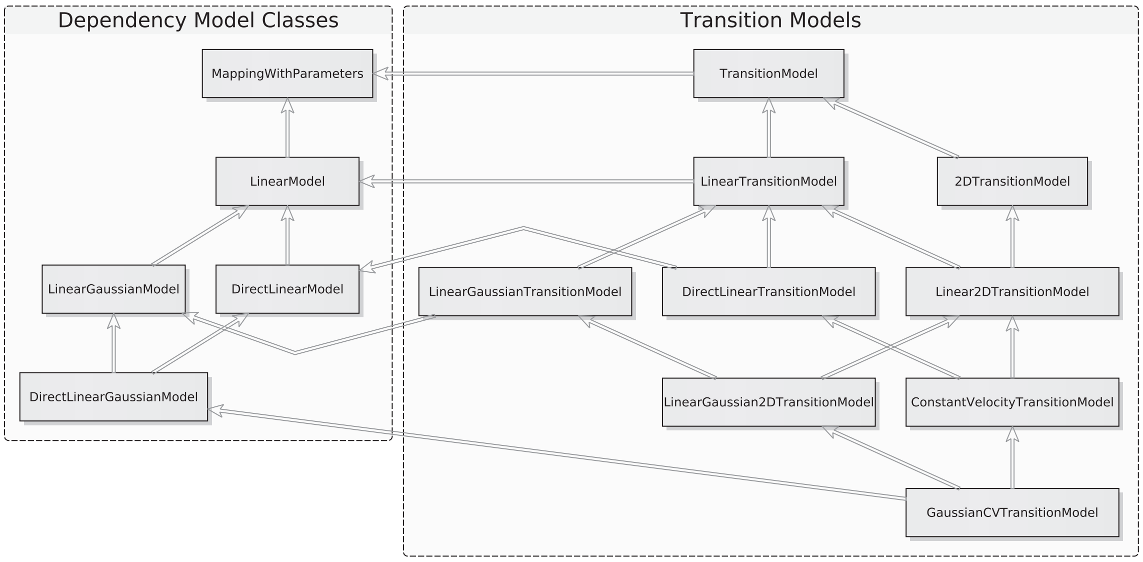


Figure 6.3: Different transition models and their relation to generic mapping classes.

6.2.4 Transition and Observation Models

The basis for state space models is the explicit definition of transition and observation models. These can be derived from the `MappingWithParameters` class and its sub classes.

Fig. 6.3 shows the class hierarchy at the example of transition models. Observation models can be constructed analogously.

The base classes `TransitionModel` and `ObservationModel` can directly be derived from `MappingWithParameters`. Aliases (*previous*/*next*, *state*/*measurement*) are introduced for context based better readability:

- ☐ `TransitionModel : MappingWithParameters`
  - ○ `MultiValueAttribute previous (=in)`
  - ○ `MultiValueAttribute next (=out)`
  - ○ `parameters`
- ☐ `ObservationModel : MappingWithParameters`
  - ○ `MultiValueAttribute state (=in)`
  - ○ `MultiValueAttribute measurement (=out)`
  - ○ `parameters`

Sub classes can be derived using the already defined sub classes of `MappingWithParameters`, such as `LinearModel`, `DirectLinearModel` and `LinearGaussianModel`:

- ☐ `LinearTransitionModel : TransitionModel, LinearModel`
  - ○ `MultiValueAttribute previous`
  - ○ `MultiValueAttribute next`
  - ○ `TransitionMatrix` f `(=parameters.`a`)`
  - ○ `TransitionNoise` w `(=parameters.`b`)`
- ☐ `DirectLinearTransitionModel : LinearTransitionModel, DirectLinearModel`
  - ○ `MultiValueAttribute previous`
  - ○ `MultiValueAttribute next`
  - ○ `TransitionMatrix` f `(=parameters.`a`)`
  - ○ `TransitionNoise` w `(=parameters.`b`)`
  - ○ `MultiValueAttribute intermediate`
  - ■ `MatrixMultiplication multiply(in, parameters.`a`, intermediate)`
  - ■ `PlusNode plus(intermediate, parameters.`b`, out)`
- ☐ `LinearGaussianTransitionModel : LinearTransitionModel, LinearGaussianModel`
  - ○ `GaussianApproximatedFloatVector previous`
  - ○ `GaussianApproximatedFloatVector next`
  - ○ `TransitionMatrix` f `(=parameters.`a`)`
  - ○ `GaussianTransitionNoise` w `(=parameters.`b`)`

Additionally, sub classes that use the `DynamicState2D` as state type are introduced[4]:

- ☐ `2DTransitionModel : TransitionModel`
  - ○ `DynamicState2D previous`
  - ○ `DynamicState2D next`
  - ○ `parameters`
- ☐ `Linear2DTransitionModel : LinearTransitionModel, 2DTransitionModel`
  - ○ `DynamicState2D previous`
  - ○ `DynamicState2D next`
  - ○ `TransitionMatrix` f `(=parameters.`a`)`
  - ○ `TransitionNoise` w `(=parameters.`b`)`
- ☐ `LinearGaussian2DTransitionModel : LinearGaussianTransitionModel, Linear2DTransitionModel`
  - ○ `GaussianDynamicState2D previous`
  - ○ `GaussianDynamicState2D next`
  - ○ `TransitionMatrix` f `(=parameters.`a`)`
  - ○ `TransitionNoise` w `(=parameters.`b`)`

[4]These classes can also be backed up by corresponding generic parent classes.

Based on the Linear2DTransitionModel specific transition models can be implemented, such as the ConstantVelocityTransitionModel using the ConstantVelocityMatrix2D from Sec. 6.2.1:

- □ ConstantVelocityTransitionModel : DirectLinearTransitionModel, Linear2DTransitionModel
  - ○ DynamicState2D previous
  - ○ DynamicState2D next
  - ○ ConstantVelocityMatrix2D f (=parameters.a)
  - ○ TransitionNoise w (=parameters.b)
  - ○ DynamicState2D intermediate
  - ■ MatrixMultiplication multiply(in, parameters.a, intermediate)
  - ■ PlusNode plus(intermediate, parameters.b, out)

The derivation of a Gaussian approximated specialization can directly be defined by inheriting from the corresponding classes without further definitions:

- □ GaussianCVTransitionModel : LinearGaussian2DTransitionModel, ConstantVelocityTransitionModel
  - ○ GaussianDynamicState2D previous
  - ○ GaussianDynamicState2D next
  - ○ ConstantVelocityMatrix2D f (=parameters.a)
  - ○ GaussianTransitionNoise w (=parameters.b)
  - ○ GaussianDynamicState2D intermediate
  - ■ GaussianMatrixMultiplication multiply(in, parameters.a, intermediate)
  - ■ GaussianPlusNode plus(intermediate, parameters.b, out)

If analog classes are defined for the ObservationModel specialized models can also be defined there, such as the PositionObservationModel using the PositionObservationMatrix2D from Sec. 6.2.1:

- □ PositionObservationModel : DirectLinearObservationModel
  - ○ DynamicState2D state
  - ○ Translation2D measurement
  - ○ PositionObservationMatrix2D h (=parameters.a)
  - ○ ObservationNoise v (=parameters.b)
  - ○ Translation2D intermediate
  - ■ MatrixMultiplication multiply(in, parameters.a, intermediate)
  - ■ PlusNode plus(intermediate, parameters.b, out)

Independently of the transition and observation models, also DynamicState 2DTransformation is introduced, that transforms a DynamicState2D into a different coordinate system using a relative Pose2D for transformation.

- □ DynamicState2DTransformation : DirectLinearModel
  - ○ DynamicState2D from (=in)
  - ○ DynamicState2D to (=out)

○ `Pose2D relative (=parameters)`

○ `DynamicState2D intermediate`

■ `MatrixMultiplication multiply(in, parameters.`a`, intermediate)`

■ `PlusNode plus(intermediate, parameters.`b`, out)`

6.2.5 General State Space Model Classes

State space models (Sec. 2.6.1) are the basis for all well-known filter methods (Sec. 2.6.2 - Sec. 2.6.6). Fig. 6.4 shows how this can be represented in a class hierarchy.

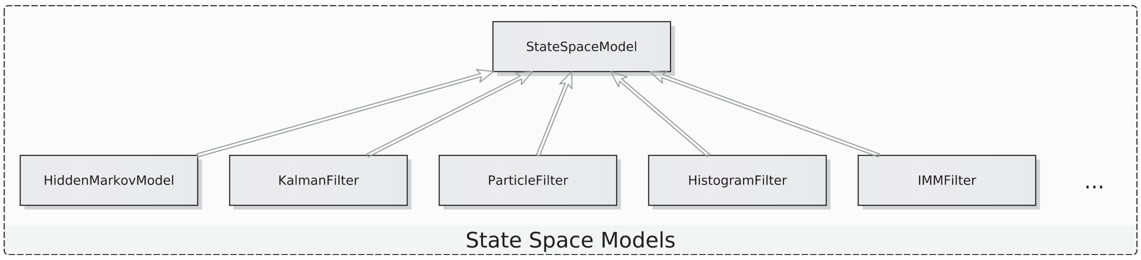


Figure 6.4: State space models are the basis for all well-known filter methods. The specific derivation will be shown at the example of the Kalman filter.

This section shows the specific derivation of filters from the generic state space model at the example of the Kalman filter (Fig. 6.5).

Note that the generic `Transition-` and `ObservationModel`, the linear and gaussian specializations and the `LinearGaussianTransitionModel` and `LinearGaussianObservationModel` are already defined in Sec. 6.2.4. The switch from generic mappings to linear Gaussian models corresponds exactly to the difference between the generic state space model and the Kalman filter.

The basis for the generic state space model is the definition of time slices consisting of 3 classes (Fig. 6.6):

□ `TimeSliceBase`

○ `state`

○ `observation`

■ `ObservationModel observation_model(state, observation)`

s `forwardpass()`

s `backwardpass()`

s `forwardpredict()`

s `backwardpredict()`

□ `FirstTimeSlice : TimeSliceBase`

○ `state`

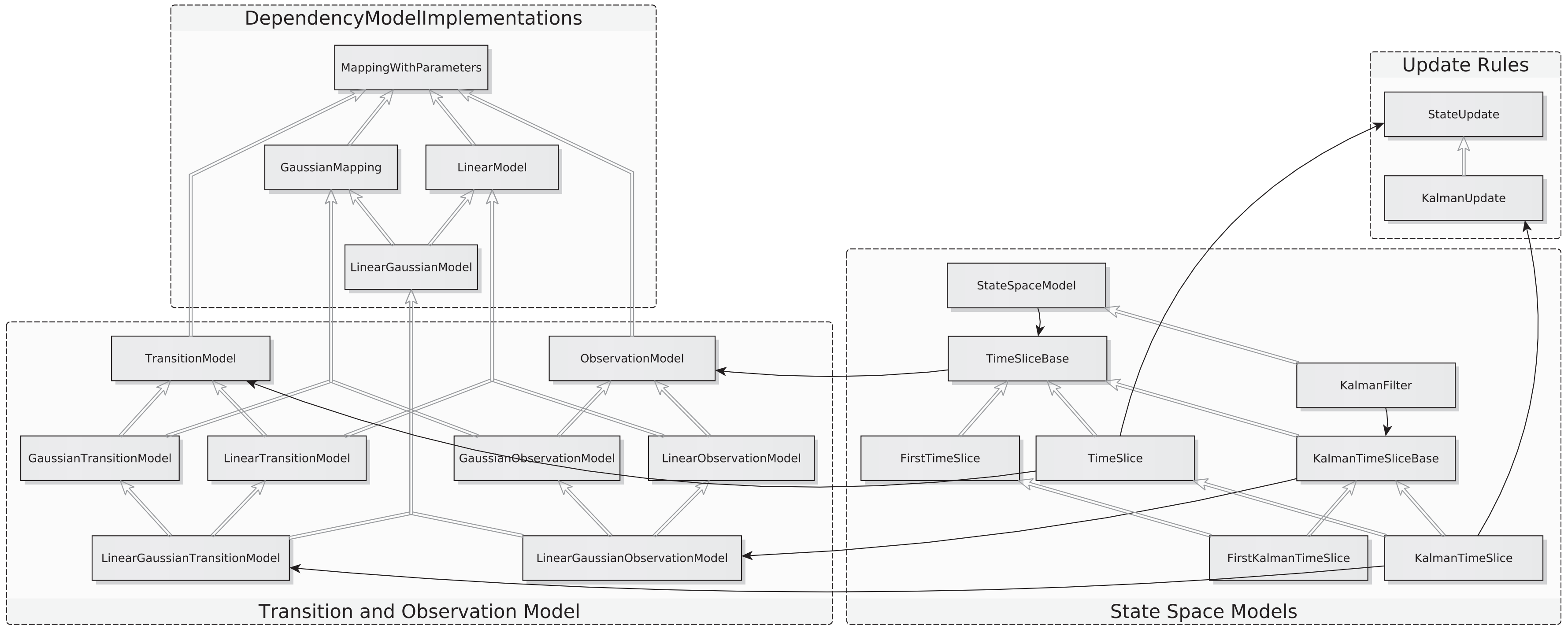


Figure 6.5: The Kalman filter is a state space model that uses special sub classes of the used transition and observation models.

○ observation

■ ObservationModel observation_model(state, observation)

s forwardpass():
u_{state}(observation)

s backwardpass():
none

s forwardpredict():
none

s backwardpredict():
none

□ TimeSlice : TimeSliceBase

○ state

○ observation

■ ObservationModel observation_model(state, observation)

○ TimeSliceBase previous

■ TransitionModel transition_model(previous.state, state)

s forwardpass():
u_{state}(previous.state, observation)

s backwardpass():
$u_{\text{previous.state}}$(observation)

s forwardpredict():
u_{state}(previous.state)

s backwardpredict():
$u_{\text{previous.state}}$(state)

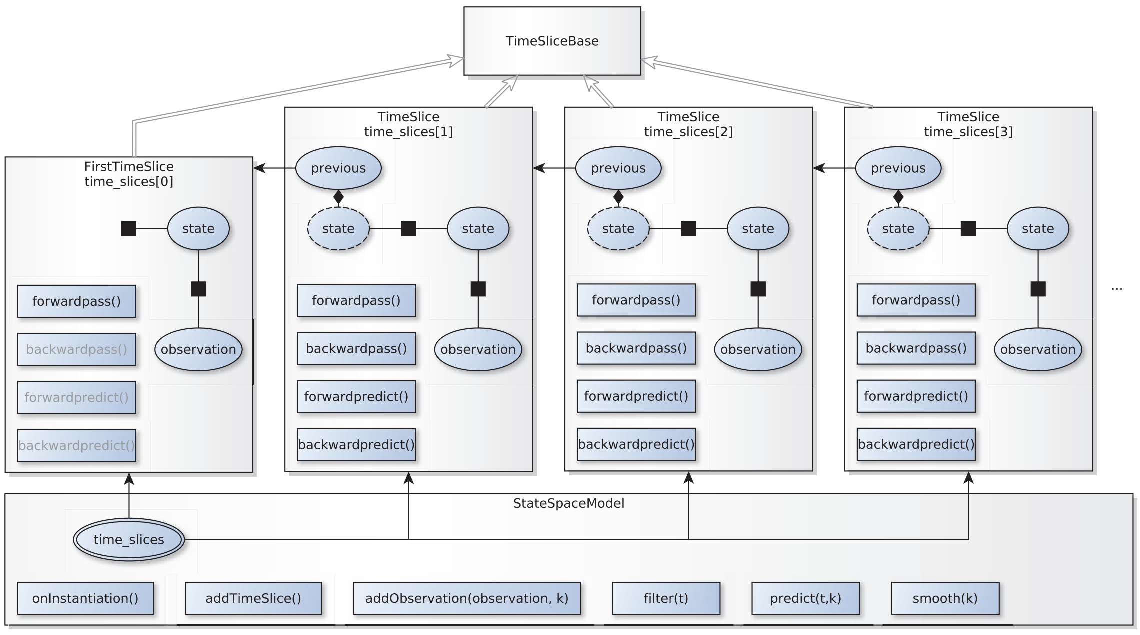


Figure 6.6: Class structure and processing scripts of the state space model. Specific filter principles (Sec. 2.6.2 - Sec. 2.6.6) are modeled as sub classes according to Fig. 6.5.

Having `TimeSlice` and `FirstTimeSlice` with a common base class `TimeSliceBase` allows building up a chain of time slices starting with a `FirstTimeSlice` that can include a state prior. All later time slices are of type `TimeSlice` and reference the preceding time slice.[5] Several inference directions are already defined on this level but have to be scheduled on higher level. Therefore a `StateSpaceModel` class is introduced:

```
□ StateSpaceModel

     ○ TimeSliceBase[] time_slices

     s onInstantiation():
       time_slices.append(newInstance(FirstTimeSlice))

     s addTimeSlice():
       new_time_slice = newInstance(TimeSlice)
       new_time_slice.previous = time_slices.last
       time_slices.append(new_time_slice)

     s addObservation(observation, k):
       setPrior(time_slices[k].observation, observation)

     s filter(t):
       time_slices[0:t].forwardpass()

     s predict(t,k):
       time_slices[0:t].forwardpass()
       time_slices[t:k].forwardpredict()

     s smooth(k):
       time_slices[0:k].forwardpass()
       time_slices[T:k+1].backwardpass()
```

The overall processing scripts for filtering, predicting and smoothing implement the generic state space model inference methods (Sec. 2.6). They are defined using the scripts of the `TimeSlice` classes which themselves refer to update rules that are not yet defined, such as $u_{\text{state}}(\text{previous.state},\ \text{observation})$.

The update rules can either be derived from the graph structure: Since it is tree-structured the generic message passing schedule (Sec. 2.5.3) can be applied. Or they can be defined explicitly in a sub class of `TimeSlice`:

```
□ GenericTimeSlice :  TimeSlice

     ○ state

     ○ observation

     ■ ObservationModel observation_model(state, observation)

     ○ TimeSliceBase previous

     ■ TransitionModel transition_model(previous.state, state)

       ⋮

     → $u_{\text{state}}$(previous.state,observation) =
       transition_model.$u_{\text{next}}$(previous,parameters),
       observation_model.$u_{\text{state}}$(measurement,parameters)
```

[5]The time slice idea is also used in [64]. State space models are there called *dynamic domain*.

→ $u_{\texttt{previous.state}}$(observation) =
observation_model.$u_{\texttt{state}}$(measurement,parameters),
transition_model.$u_{\texttt{previous}}$(next,parameters)

→ $u_{\texttt{state}}$(previous.state) =
transition_model.$u_{\texttt{next}}$(previous,parameters)

→ $u_{\texttt{previous.state}}$(state) =
transition_model.$u_{\texttt{previous}}$(next,parameters)

Of course, the straightforward update rule based on the tree-structure is not always desirable and can be replaced by a more efficient update rule, such as in the case of the Kalman filter: The Kalman filter consists of the following sub classes of the given state space model classes:

☐ KalmanStateSpaceModel : StateSpaceModel

○ KalmanTimeSliceBase[] time_slices

s addObservation(observation, k)

s addTimeSlice()

s filter(t)

s predict(t,k)

s smooth(k)

☐ KalmanTimeSliceBase : TimeSliceBase

○ state

○ observation

■ LinearGaussianObservationModel observation_model(state, observation)

s forwardpass()

s backwardpass()

s forwardpredict()

s backwardpredict()

☐ FirstKalmanTimeSlice : KalmanTimeSliceBase, FirstTimeSlice

○ state

○ observation

■ LinearGaussianObservationModel observation_model(state, observation)

⋮

☐ KalmanTimeSlice : KalmanTimeSliceBase, TimeSlice

○ state

○ observation

■ LinearGaussianObservationModel observation_model(state, observation)

○ KalmanTimeSliceBase previous

■ LinearGaussianTransitionModel transition_model(previous.state, state)

⋮

→ $u_{\texttt{state}}$(previous.state,observation) =
transition_model.$u_{\texttt{next}}$(previous,parameters),
$u_{\texttt{state}}$(state.from(transition_model),observation)

$\rightarrow$ $u_{\texttt{state}}$(`state.from(transition_model),observation`) $=$
`KalmanUpdateCompositeNode(state.from(transition_model),`
`observation,state,observation_model.parameters)` $:: u_z(x, a, y)$

Besides the specialization to the LinearGaussian sub classes of the models a special implementation of the $u_{\text{state}}(\text{previous.state},\ \text{observation})$ update rule is defined using the `KalmanUpdateCompositeNode` from Sec. 6.2.2 for state update.

This is a very generic definition of state space models and filter specializations, independent of the actual application and domain. It will be further specialized according to the GNSS tracking application in Sec. 6.3. Other applications (Chapter 7-9) build up on a similar basis using the same state space model classes but different filter sub classes.

6.3 Application Specific Definitions

After having built up the class hierarchy for the general attribute representations, dependency models, state space models and Kalman filter, they can be applied to the GNSS tracking application. The idea is that the traffic domain ontology (Chapter 5) is brought together with the general models (Sec. 6.2).

6.3.1 Application Specific Ontology

From the DSL an application specific ontology has to be derived to represent the conditions of the application. From the entity and relation class structure of the DSL for the traffic domain (Fig. 5.3) appropriate entities and relations have to be chosen to bring over to the application specific ontology (Fig. 6.7).

For this example it is enough to select the `AutomatedVehicle`, the `Inertial Sensor` and the `changesTo` relation to represent the temporal fusion. This basic ontology is extended by sub classes and helper classes to fully define the estimation problem, approach and inference method.

The first application specific extension is related to the attributes of interest: The position of the ego vehicle has to be estimated via position measurements from the GNSS sensor. Both positions are related to a reference coordinate system. This coordinate system is called `World` and a `Has` relation from `World` to `Scene Object` is added as well as a `Detects` relation from `GNSSSensor` to `World` (Fig. 6.8). These describe that all scene objects are positioned in the world and the GNSS sensor can observe its position in the world.

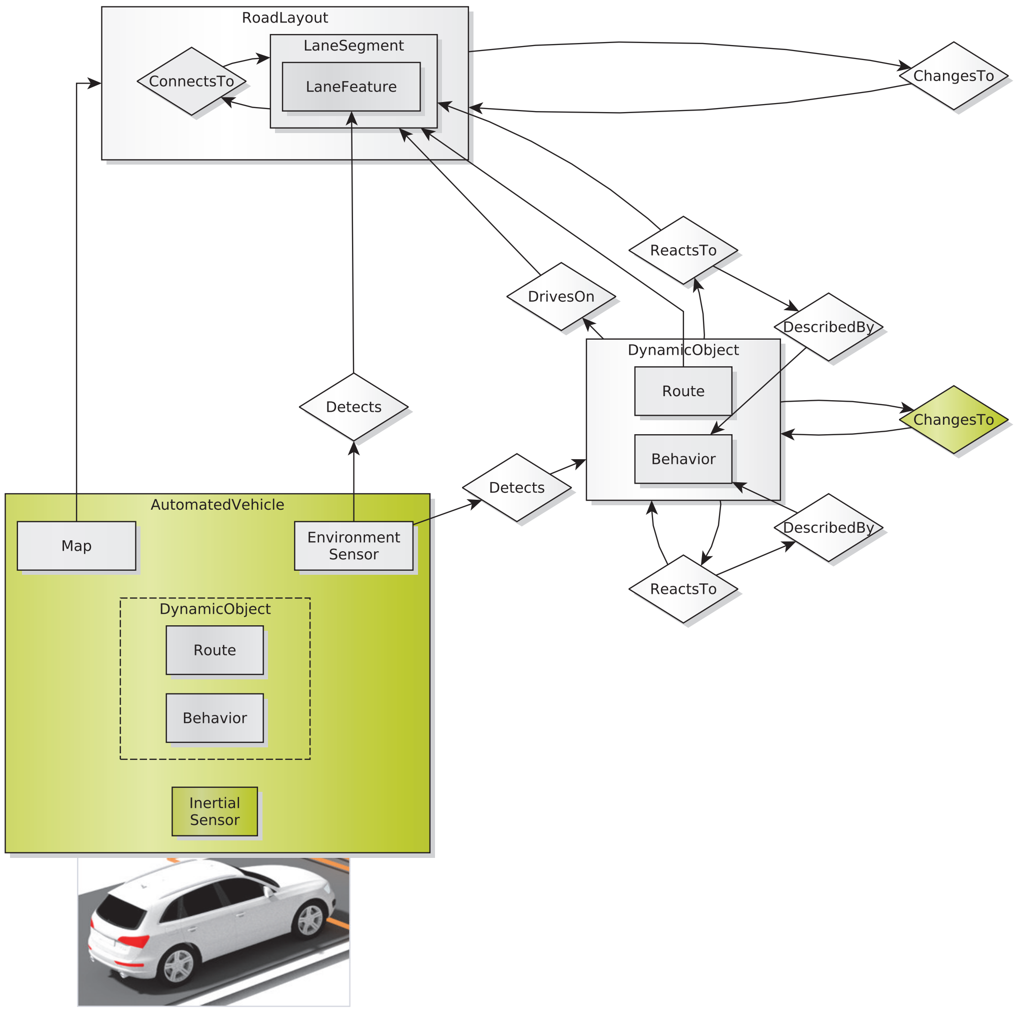


Figure 6.7: Only a few entities and relations (green) are needed to model the GNSS pose estimation application.

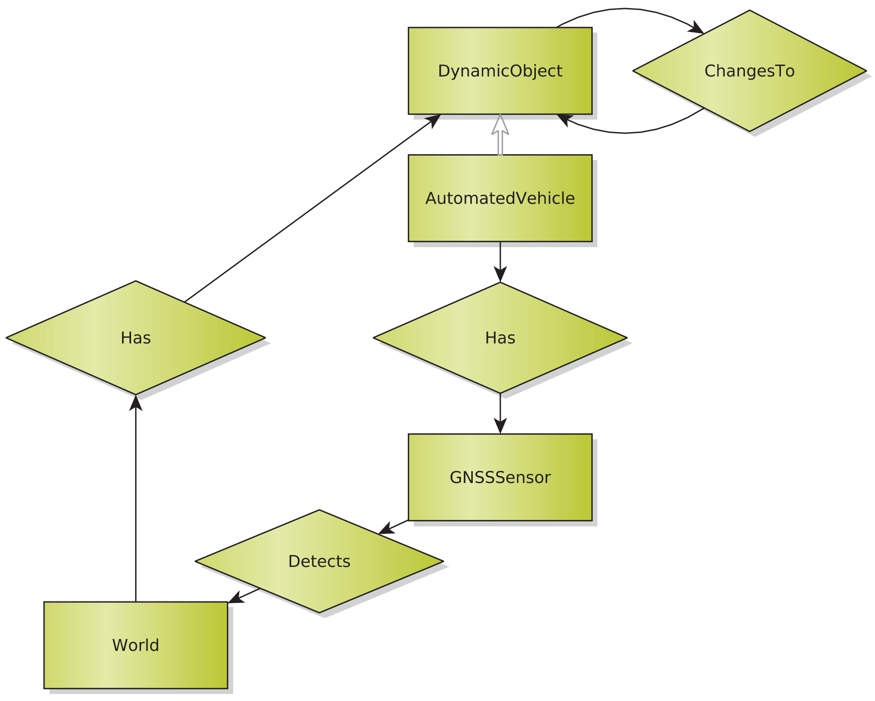


Figure 6.8: Traffic Domain Ontology for GNSS Pose Estimation with added `World` class and relations. Some parent classes are neglected to keep analogy to Fig. 6.7.

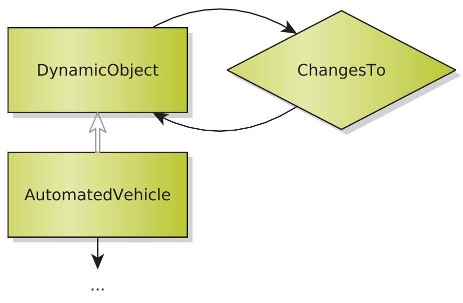
6.3.2 Classes, Attributes and Dependencies

After having defined the application ontology, which defines all entities and relations between them, the classes representing the entities and relations with their attributes and dependencies (factors) can be introduced. These will model the real world properties and build up the basis for applying an inference method.

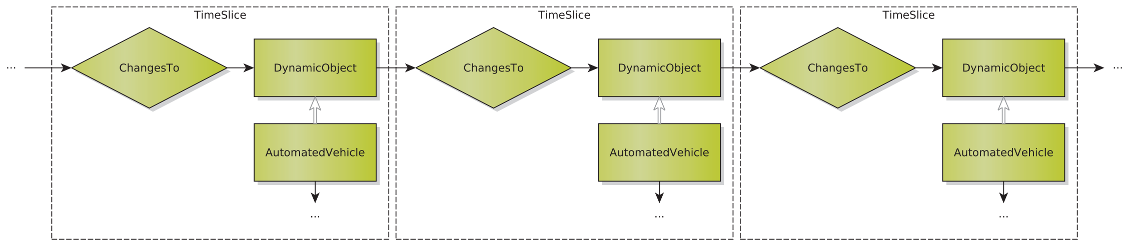
Note that it is exemplarily shown how the attributes and factors of the classes can look like. This is an exemplary solution and does not mean the maximum amount of object-orientation is used. Attribute and factor definitions can be spread over further sub classes making the utilization of object-orientation and re-usability possibilities much larger.

The `changesTo` relation can be interpreted as a temporal dependency connecting a current state of an automated vehicle to a future state. This is depicted in Fig. 6.9.

Before the principle of a state space model is applied the attributes and dependencies in the entities and relations are defined. Every entity and relation corresponds to one OOFGML class each and is part of a larger class hierarchy (Fig. 6.10). The focus lies on the attributes necessary for GNSS-based pose estimation. These are the pose (state) of the automated vehicle in the real world and the position



(a) Schematic view on the `changes To` relation.



(b) The `changesTo` relation can be used as a connection point for a sequence of time slices.

Figure 6.9: Intepreting the `changesTo` relation as a temporal dependency leads to a rolled out model.

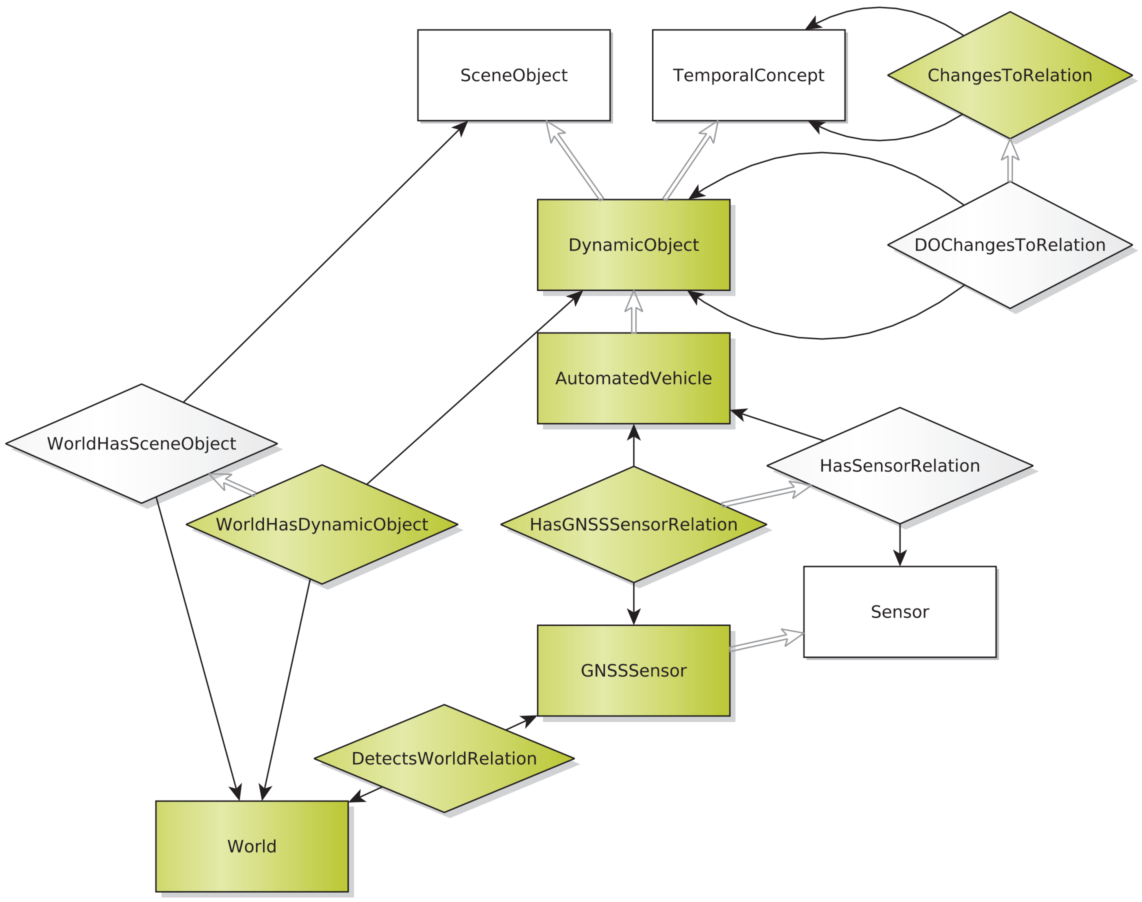


Figure 6.10: Class hierarchy and references of the defined OOFGML classes. Note that the green classes correspond to the ontology in Fig. 6.8. Arrows of relation classes are now depicting the references of their subject/object attributes.

measurements detected by the GNSS sensor. Dependencies have to be modeled between the position measurements and the vehicle state as well as temporally between subsequent vehicle states. Fig. 6.11 shows these attributes and factors. Additional attributes which implement the relations between classes are depicted.

The class definitions are as follows:

- ☐ `World`
- ☐ `TemporalConcept`
  - ○ `ChangesToRelation changes_from` (inverse)
  - ○ `ChangesToRelation changes_to` (inverse)
- ☐ `SceneObject`
- ☐ `DynamicObject : SceneObject, TemporalConcept`
  - ○ `WorldHasDynamicObject world_has_relation` (inverse)
- ☐ `WorldHasSceneObject : Relation`

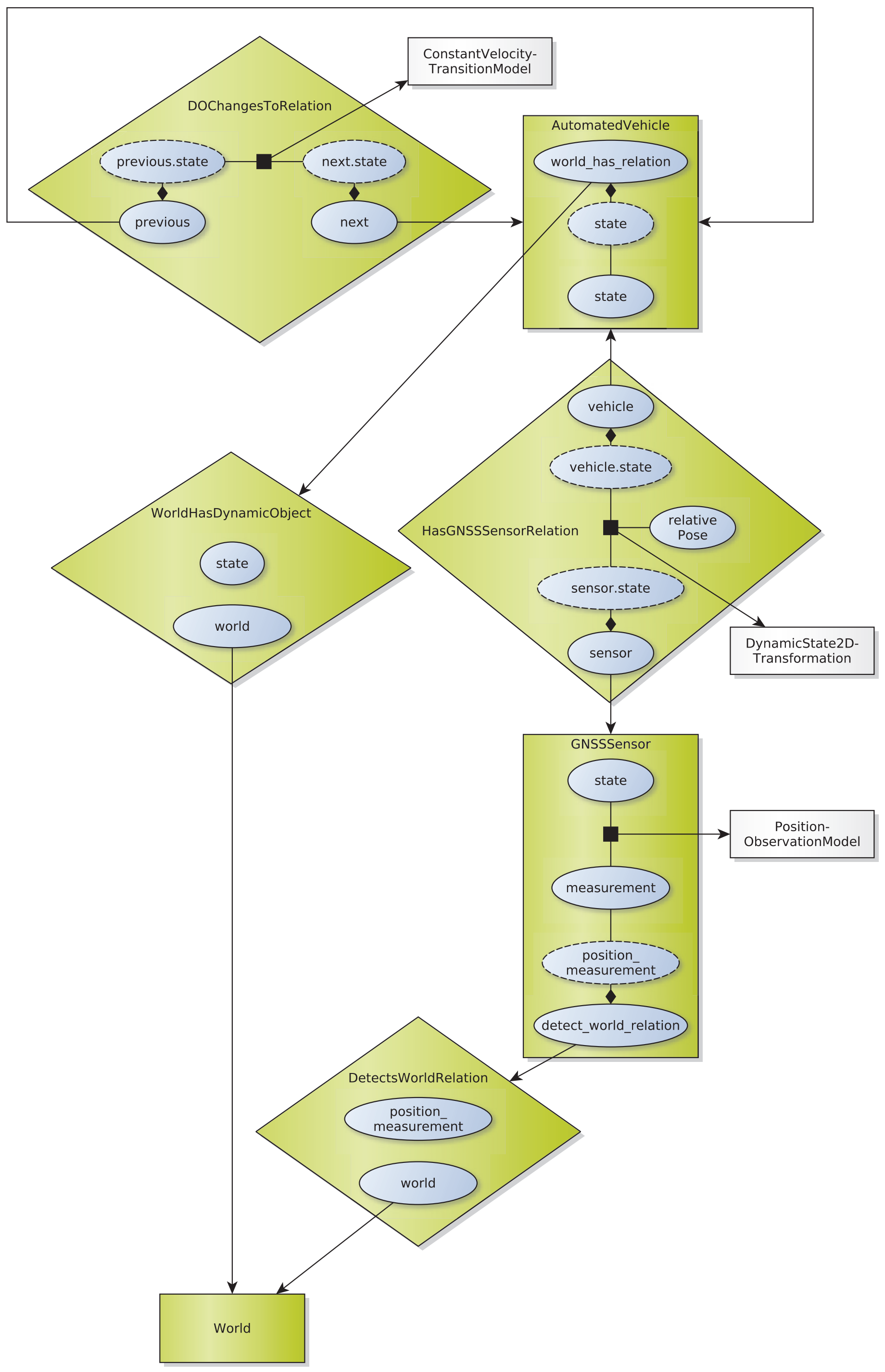


Figure 6.11: Attributes and dependencies motivated from application-specific real world properties.

- ○ World world (=subject)
- ○ SceneObject object
- ○ Pose2D pose

□ WorldHasDynamicObject : WorldHasSceneObject

- ○ World world (=subject)
- ○ DynamicObject object
- ○ Pose2D pose
- ○ DynamicState2D state

□ ChangesToRelation : Relation

- ○ TemporalConcept previous (=subject)
- ○ TemporalConcept next (=object)
- ■ TransitionModel transition(previous, next)

□ HasSensorRelation : Relation

- ○ AutomatedVehicle vehicle (=subject)
- ○ Sensor sensor (=object)
- ○ Pose relative_pose

□ Sensor

- ○ HasSensorRelation host (inverse)
- ○ state
- ○ measurement
- ■ ObservationModel observation_model(state, measurement)

□ AutomatedVehicle : DynamicObject

- ○ WorldHasDynamicObject world_has_relation (inverse)
- ○ HasGNSSSensorRelation gnss (inverse)
- ○ DOChangesToRelation changes_from (inverse)
- ○ DOChangesToRelation changes_to (inverse)

□ DOChangesToRelation : ChangesToRelation

- ○ AutomatedVehicle previous (=subject)
- ○ AutomatedVehicle next (=object)
- ■ ConstantVelocityTransitionModel transition(previous,next)

□ HasGNSSSensorRelation : HasSensorRelation

- ○ AutomatedVehicle vehicle (=subject)
- ○ GNSSSensor sensor (=object)
- ○ Pose relative_pose
- ■ DynamicState2DTransformation transformation(vehicle.state, sensor.state, relative_pose)

□ GNSSSensor : Sensor

- ○ GNSSSensorHasRelation host (inverse)
- ○ DetectsWorldRelation detect_world_relation (inverse)
- ○ DynamicState2D state

```
        ○ Translation2D measurement (=detect_world_relation.position_measurement)
        ■ PositionObservationModel observation_model(state, measurement)
  □ DetectsWorldRelation
        ○ GNSSSensor subject
        ○ World object
        ○ Translation2D position_measurement
```

These class definitions match the real world properties of the application scenario. The `Relation` base class is defined in Sec. 4.3.1. Inheritence from the `Entity` base class is neglected. The dependencies are formulated using the factor classes from Sec. 6.2.4: `ConstantVelocityTransitionModel`, `PositionObservationModel` and `DynamicState2DTransformation`.

Note that classes only describe local dependencies and can be used similarly in other applications. Relations that are defined on parent classes can be reused, even when other sub classes are used that map attributes differently.

The defined model describes already the application sufficiently with attributes and dependencies between them. A factor graph can directly be derived and simple message passing can be used to infer desired attributes from observations. Depending on the actual attribute representations the inference can be very expensive and applying well-known inference methods and approximations is beneficial. Additionally, several inference tasks and their message passing schedules have to be managed with the help of processing scripts and update rules. Therefore, the application is first mapped to a state space model (Sec. 6.3.3) and then specialized to a specific attribute representation (Sec. 6.3.4).

6.3.3 Application of State Space Model

To benefit from generic state space model principles the state space model classes from Sec. 6.2.5 are applied to the GNSS application classes from Sec. 6.3.2.

As the generic state space model consists of the 4 classes `StateSpaceModel`, `TimeSliceBase`, `FirstTimeSlice` and `TimeSlice`, sub classes of these are derived. The GNSS-specific sub classes derive all members and extend them by references to the already defined application-specific classes representing the entities and relations of the application:

```
  □ GNSSStateSpaceModel : StateSpaceModel
        ○ GNSSTimeSliceBase[] time_slices
          ⋮
  □ GNSSTimeSliceBase : TimeSliceBase
        ○ state
        ○ observation
        ■ GNSSSensorObservationModel observation_model(state, observation)
        ○ AutomatedVehicle automated_vehicle

        ○ DOChangesToRelation changes_to

        ○ GNSSSensorHasRelation has

        ○ GNSSSensor gnss

        ■ state_definition(state, self)
          # connects selected GNSS attributes to state

        ■ observation_definition(observation, self)
          # connects selected GNSS attributes to observation

        s onInstantiation():
          automated_vehicle = newInstance(AutomatedVehicle)


          ⋮

          observation_model = newInstance(GNSSSensorObservationModel)
          observation_model.model1 = has.transformation
          observation_model.model2 = gnss.observation_model


          ⋮
    □ FirstGNSSTimeSlice :  GNSSTimeSliceBase, FirstTimeSlice

        ○ AutomatedVehicle automated_vehicle

        ○ DOChangesToRelation changes_to

        ○ GNSSSensorHasRelation has

        ○ GNSSSensor gnss

          ⋮
    □ GNSSTimeSlice :  GNSSTimeSliceBase, TimeSlice

        ○ GNSSTimeSliceBase previous

        ○ state

        ○ observation

        ■ GNSSSensorObservationModel observation_model(state, observation)

        ■ ConstantVelocityTransitionModel transition_model(previous.state, state)
          (=changes_to.transition)

        s onInstantiation():
          automated_vehicle = newInstance(AutomatedVehicle)
          changes_to = newInstance(DOChangesToRelation)
          ⋮

          ⋮
```

The processing scripts from the generic state space model definition are inherited, such as the `filter(t)`,`predict(t,k)` and `smooth(k)` functions. Thus, state space model specific inference tasks are already defined. To make them work with the application-specific ontology, the transition and observation model factors map to the corresponding factors in the application specific classes. For the observation model a special `GNSSSensorObservationModel` is used that maps to the two involved factors:

□ `GNSSSensorObservationModel : CompositeLinearObservationModel`[6]

- ○ `DynamicState2D state (=in)`
- ○ `Translation2D measurement (=out)`

  ⋮

- ■ `DynamicState2DTransformation model1 (in, parameters1, intermediate)`
- ■ `PositionObservationModel model2 (intermediate, parameters2, out)`

Note that the mapping of the two models *model1* and *model2* to the models in the application-specific classes is realized in the `onInstantiation()` script of `GNSSTimeSliceBase`.

These classes build up a generic state space model on the GNSS application classes (Fig. 6.12).

6.3.4 Kalman-Based Inference Method

For applying an efficient inference method the attribute representations have to be chosen according to the application characteristics. Until now, the attribute representations have been left open. This allowed a generic description of the estimation problem and a generic mapping to state space models. Now a proper sub class of the StateSpaceModel class (Fig. 6.4) has to be chosen. Therefore, a deeper look into the properties of the float-based attributes and the transition and observation models has to be taken.

Assume that the probability distribution of the state and measurement attributes are unimodal and can be approximated sufficiently by a Gaussian. Sub classes of the ones defined before can be derived that specialize to Gaussian attribute representations. Additionally, all the relevant factors (`ConstantVelocityTransitionModel`, `PositionObservationModel` and `DynamicState2DTransformation`) are derived from LinearModel. Linear models together with Gaussian state and observation space fits to the `KalmanFilter` sub class of the `State SpaceModel` class (Fig. 6.5). This deduction can easily be derived from class hierarchy by expert knowledge. But even automatic tools are imaginable which are not focus of this thesis and part of possible future research.

Sub classes of the given ones can be created that additionally inherit from `Kalman Filter`. Using the `KalmanFilter` class automatically includes all Kalman-specific methods, such as the special update rule via `KalmanUpdateComposite Node`. This update rule is defined in the generic `KalmanTimeSlice` (Sec. 6.2.5) and applied via `GNSSSensorObservationModel` (Sec. 6.3.3) onto the application-specific dependency models (Sec. 6.3.2).

[6]The `CompositeLinearObservationModel` ist defined analogously to the classes in Sec. 6.2.4, as a sub class of `LinearObservationModel` and `CompositeLinearModel`.

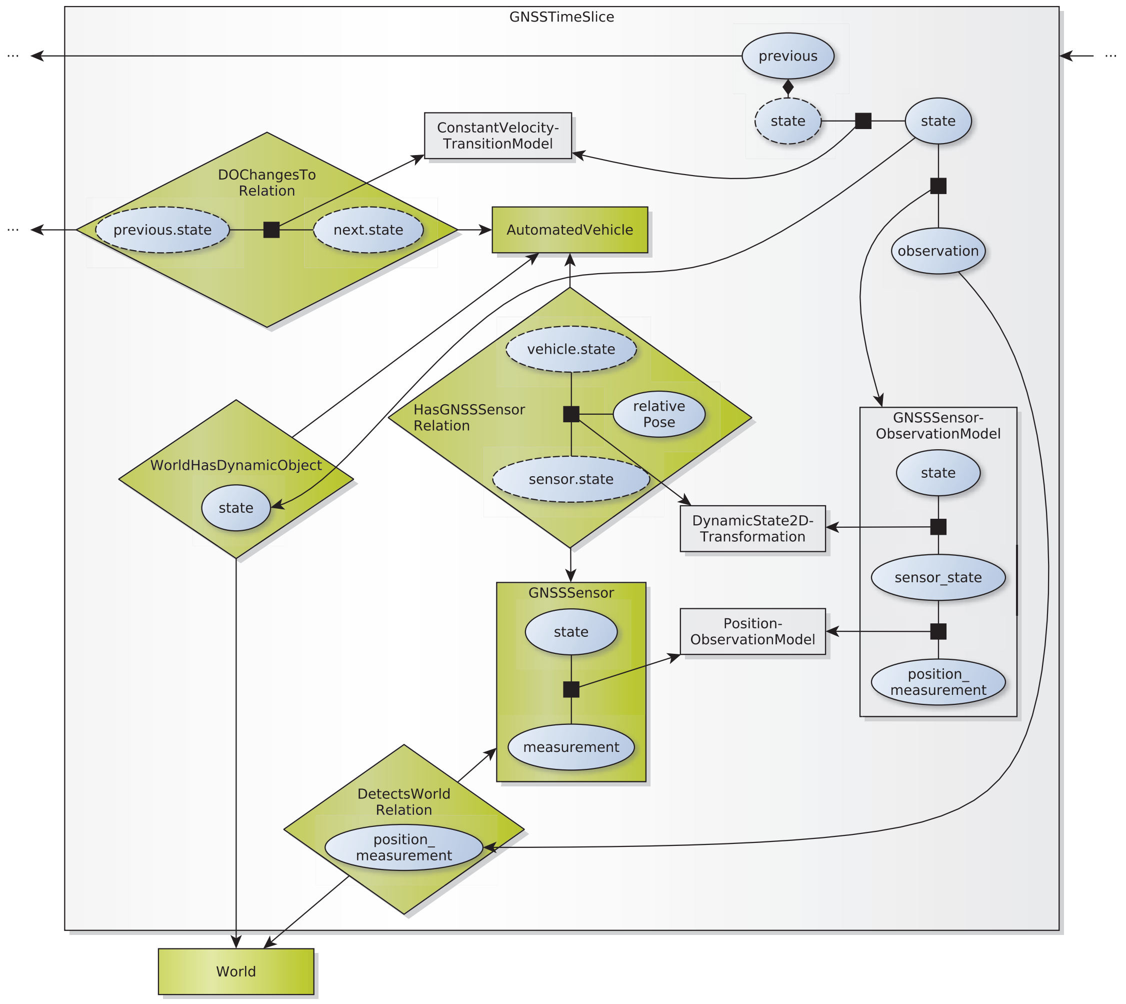


Figure 6.12: The `GNSSTimeSlice` class and its connection to the ontology induced GNSS application classes from Fig. 6.11. Only the relevant parts are depicted.

6.4 Implementation Opportunities

As described in Sec. 3.4 the generic language is mainly a formalization of estimation problems and does not restrict to a specific implementation. The model created in this GNSS example can be implemented by different opportunities offered by the OOFGML:

- **Full OOFGML implementation**
  All classes (entities and relations) are represented in the implementation. When new sensor measurements arrive the corresponding processing scripts are triggered, instances are instantiated, evidences applied and inference is triggered. This requires a full OOFGML implementation. The comfort of full flexibility comes with organizational overhead which will have some effect on the runtime performance. Especially in the case of this simple example other implementation opportunities are more beneficial.

- **FOPL implementation**
  Instead of a full OOFGML implementation an (existing) implementation for a first-order probabilistic language (FOPL) (Sec. 2.7) can be used. The OPRML (as a subgroup using discrete state spaces and Bayesian network methods) cannot be applied to the given example since, although Kalman filters can be described as Bayesian networks, they are not discrete. A FOPL with continuous state spaces would be needed. Also, the advantage of FOPL having a dynamic domain is not needed by the application. The FOPL approach is used in Chapter 9 for a more dynamic application (estimating interactions between traffic participants).

- **Specialized implementation**
  After fully modeling the application in the OOFGML in Sec. 6.3.4 it became clear that for actual inference very few dependency models are combined in one graph. The object-oriented model can be reduced to a small factor graph (Fig. 6.13). This can be implemented in a static way. Even dynamic parts like the arrival of new sensor measurements can be incorporated without implementing a full, generic class and instance system as in the previous options. This method is also chosen in Chapter 7 for a more advanced application localizing an ego vehicle using the observed surrounding object constellation.

- **Established filter implementation**
  In this example the actual inference method can be streamlined to an existing established filter principle (Sec. 2.6), the Kalman filter. It can directly be implemented in a traditional Kalman filter. The instructions for the `ConstantVelocityTransitionModel` can be represented in the Kalman prediction step whereas the two components of the `GNSSSensorObservationModel` can be combined in the update step. This is the most efficient implementation for the given application.

- **Modular implementation**
  Since the given example is very small a modular implementation is not worth of consideration. A modular implementation is used in the application in Chapter 8 for estimating the road layout in front of an ego vehicle.

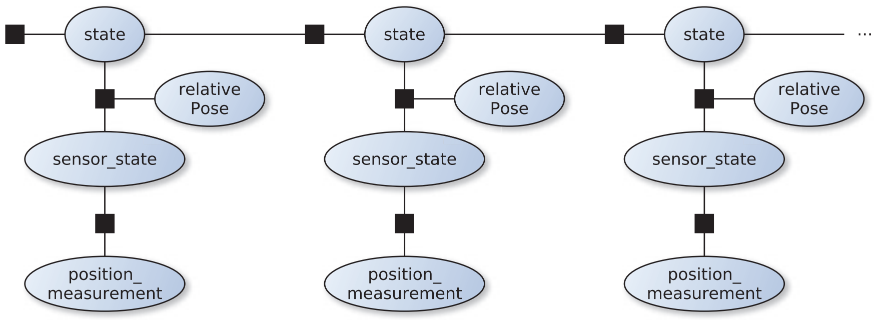


Figure 6.13: A small factor graph can directly be derived from the object-oriented model (Fig. 6.12).

6.5 Conclusion

The application of the OOFGML to a specific application has been shown in detail on a very simple, easily understandable example. All steps from building up a generic class hierarchy over understanding the application up to a specific implementation have been discussed. The key points are:

- Attribute representations, update rules and basic dependency models can be represented in a detailed class hierarchy.
- Transition and observation models are straightforwardly derived using the generic classes.
- Established filter methods are represented using a common state space model base class.
- The traffic domain ontology can be specialized to a specific application.
- Application-specific dependencies can be implemented in the application-specific classes using the generic classes.
- The application of a state space model to the application's model directly induces the specialization to a specific inference method sub class.
- Even for a small application, different implementation opportunities can be applied.

In the subsequent chapters more advanced application examples in the fields of ego vehicle localization, road layout estimation and traffic participant prediction are handled.

7 Ego Vehicle Localization

This chapter shows how an example application considering ego vehicle localization can benefit from the introduced OOFGML. A special localization method is created that infers the ego vehicle pose on a map from the observed dynamic objects in the vehicle surrounding. This means higher level information (dynamic object movements) is used to infer lower level information (ego pose). The OOFGML supports the whole development process from application description to filter implementation.

The goal of this chapter is to outline how the features of the OOFGML can be utilized to achieve the application: It is shown that all relevant aspects are describable in a consistent way. Nevertheless also the performance of the resulting application is evaluated by comparing it to a high precision inertial measurement unit as reference sensor and other common approaches such as map matching used in current GNSS navigation system solutions.

This chapter is largely based on the previous publication [7]. More details considering the application can be found there.

7.1 Related Work

Ego vehicle localization is an extensively handled field in research considering autonomous vehicles or robots.

If precision for a single vehicle is of highest priority, the most common approaches for ego vehicle localization are using *sensor-specific landmarks* [81], [83], [84], [116]. These features are calculated using as much sensor-specific information as possible and are then compared to a map, recorded with the same sensor. Several sensor types can be combined to increase robustness, e.g. monocular camera and lidar [117] or stereo camera, lidar and GNSS sensor [129].

Detected objects can be used to filter out features that belong to dynamic objects to avoid adding non-static features to the map during mapping and also to avoid using them for localization [61].

To reduce sensor dependency, interpreted static environment objects can be used as landmarks. These include lane markings on an intersection [99], lane centerlines (*LaneSLAM*) [62] or lanes, obstacles and parking spaces in a parking garage (*Semantic Localization*) [100]. Dynamic objects are only rarely used in these works.

A different approach is so-called *map matching*, known from navigation systems, where the ego vehicle is matched onto a map on road level using only GNSS and the relative movement (odometry). Recent approaches achieved increased precision using particle filters and zone maps [102], [5]. There, the movement of the vehicle (speed, direction) is utilized to find matching road types.

The approach described in this chapter is close to *Scene Understanding* by Geiger et al. [53], where tracklets are used as one of many features extracted from a stereo camera system. In contrast to the approach in this chapter, Geiger et al. do not use a map but estimate a parametric road layout model using machine learning techniques. The result is not a localization relative to a predefined map but relative to a local road model giving similar capabilities to the automated vehicle.

Many of the advanced current works [62], [53] use *factor graphs* for formalizing their approach. Obviously, all of these approaches could be formalized using the OOFGML. The chosen application can serve as an example.

7.2 Application Goal

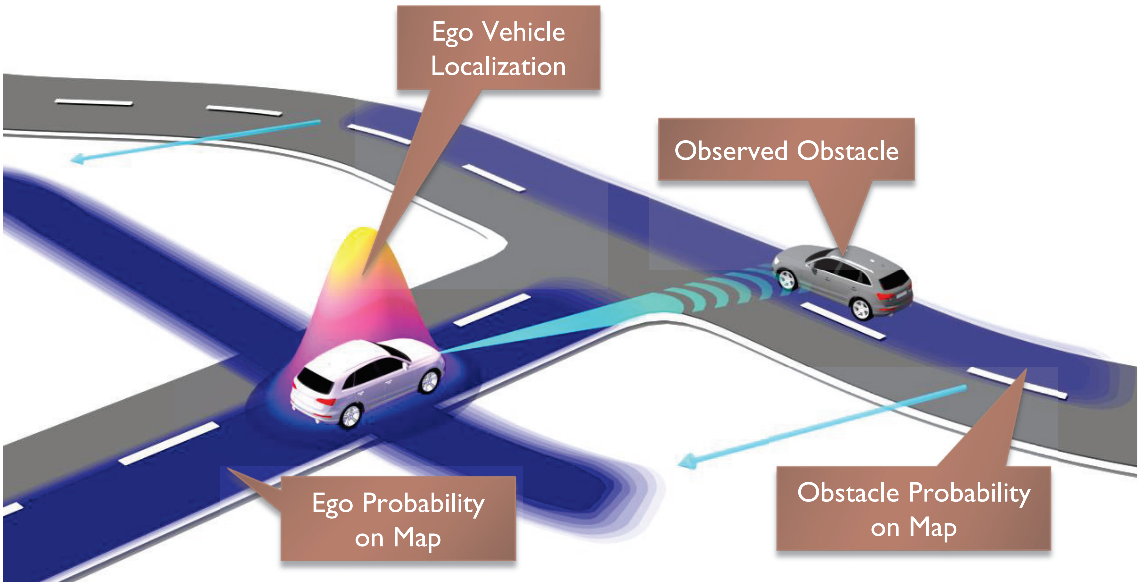


Figure 7.1: The basic idea of the object constellation localization application. Observed dynamic objects are assessed on a map to estimate the relative pose of the ego vehicle. [7]

The goal of the application is to estimate the ego vehicle's pose in a world coordinate system (Fig. 7.1). A parametric road network map is given. It describes where vehicles usually drive in which way (direction, speed) in the world coordinate system. Dynamic objects are observed relative to the ego vehicle using an object detection sensor (e.g. a Lidar (Sec. 2.2.1) with a subsequent object tracking (Sec. 2.2.2)). The observed objects can be evaluated on the map to estimate the ego vehicle's pose. The probability distribution for the ego vehicle's pose is multimodal and depends largely on the currently observed objects. Temporal fusion (tracking over time) will enable lane-precise longitudinal and lateral localization on multilane roads.

These application properties have to be formalized using the OOFGML.

7.3 Application Specific Ontology

From the DSL (Chapter 5) an application specific ontology has to be derived to represent the conditions of the application (Sec. 7.2). From the entity and relation class structure of the DSL for the traffic domain (Fig. 5.3) appropriate entities and relations have to be chosen to bring over to the application specific ontology (Fig. 7.2).

For this application several entities and relations have to be included: The `AutomatedVehicle`, the `DynamicObject` and the `RoadLayout` correspond to the real world entities. The `detects` and `drivesOn` relation realize the relative observation and the map association respectively. The `changesTo` relation represents the temporal fusion of the ego vehicle pose estimate. Analogously to the GNSS example (Chapter 6) this basic ontology is extended by sub classes and helper classes to fully define the estimation problem, approach and inference method.

7.4 Classes and Attributes

The classes describing the entities and relations are equipped with attributes and dependency models to describe the real world properties (Fig. 7.3 and Fig. 7.4). For example the dynamic object includes a pose and motion variable and the `Detects` relation incorporates the relative positioning of the object pose to the sensor pose. Note that this specialization is realized by a hierarchy of sub classes. The details are neglected here since they were already discussed analogously in Chapter 6.

7.5 Dependency Model

The dependency model (Fig. 7.3 and Fig. 7.4) is largely inspired by the work of [62]. While several factors refer to generic transformations similar to the de-

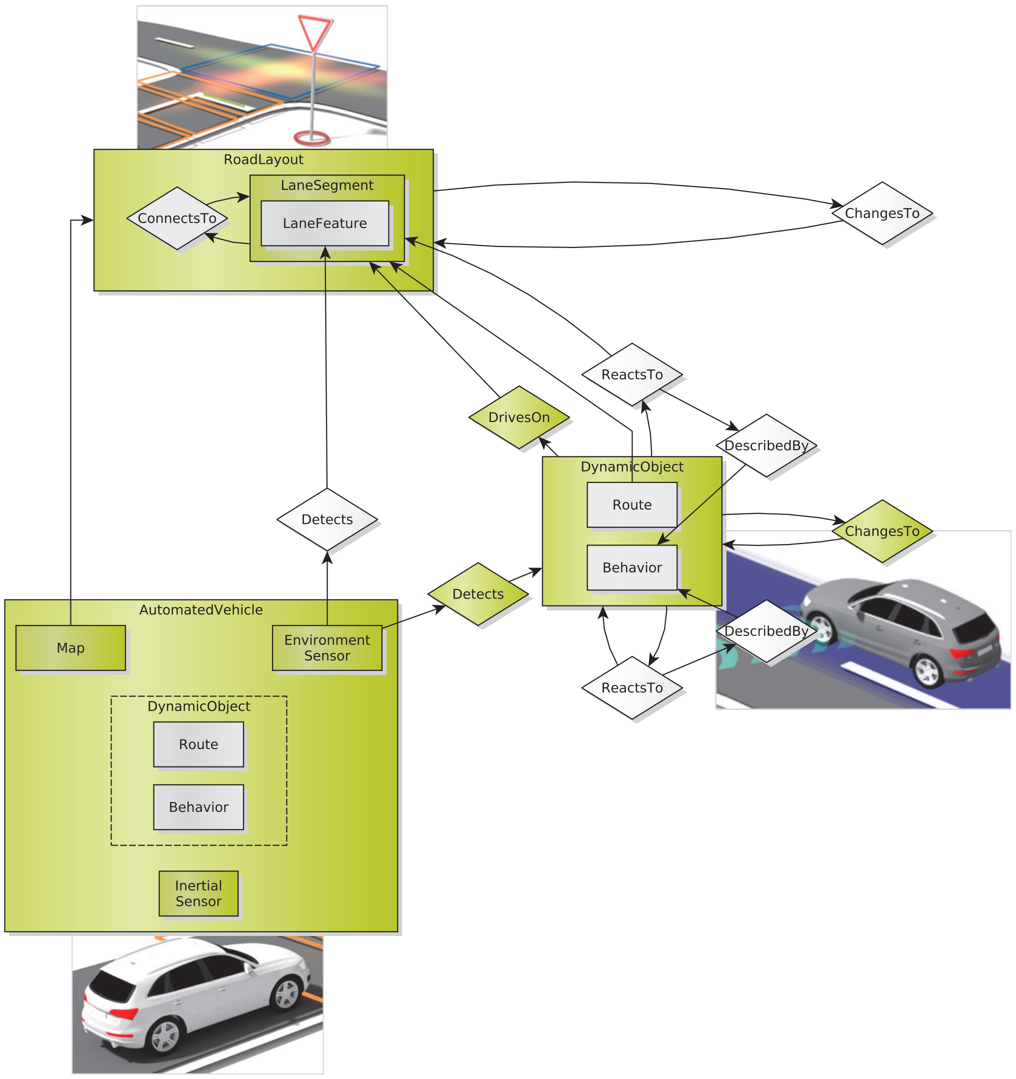


Figure 7.2: The necessary entities and relations comprise the ego vehicle, dynamic objects and the road layout.

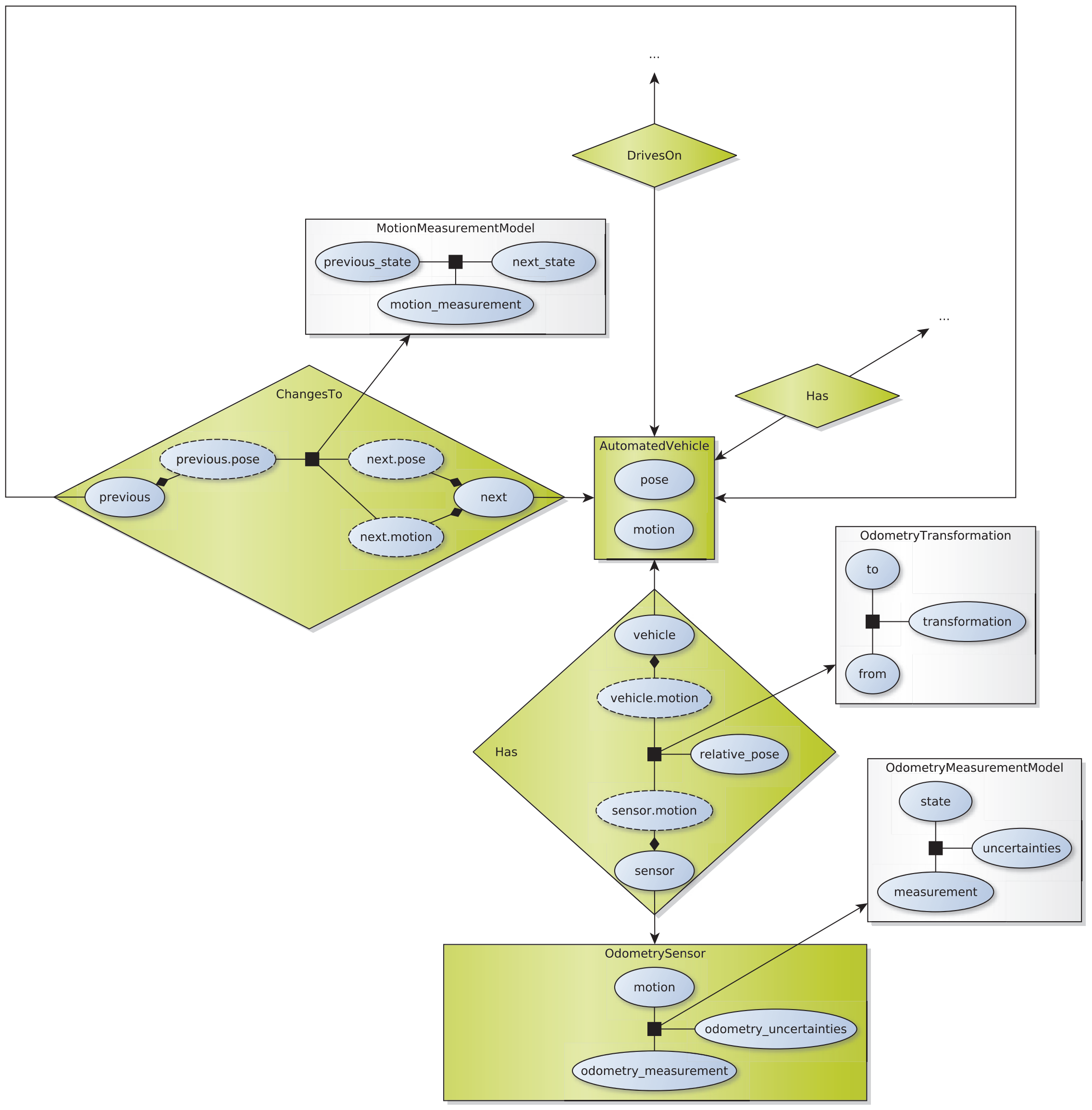


Figure 7.3: Attributes and dependency models in the lower part of the application specific ontology.

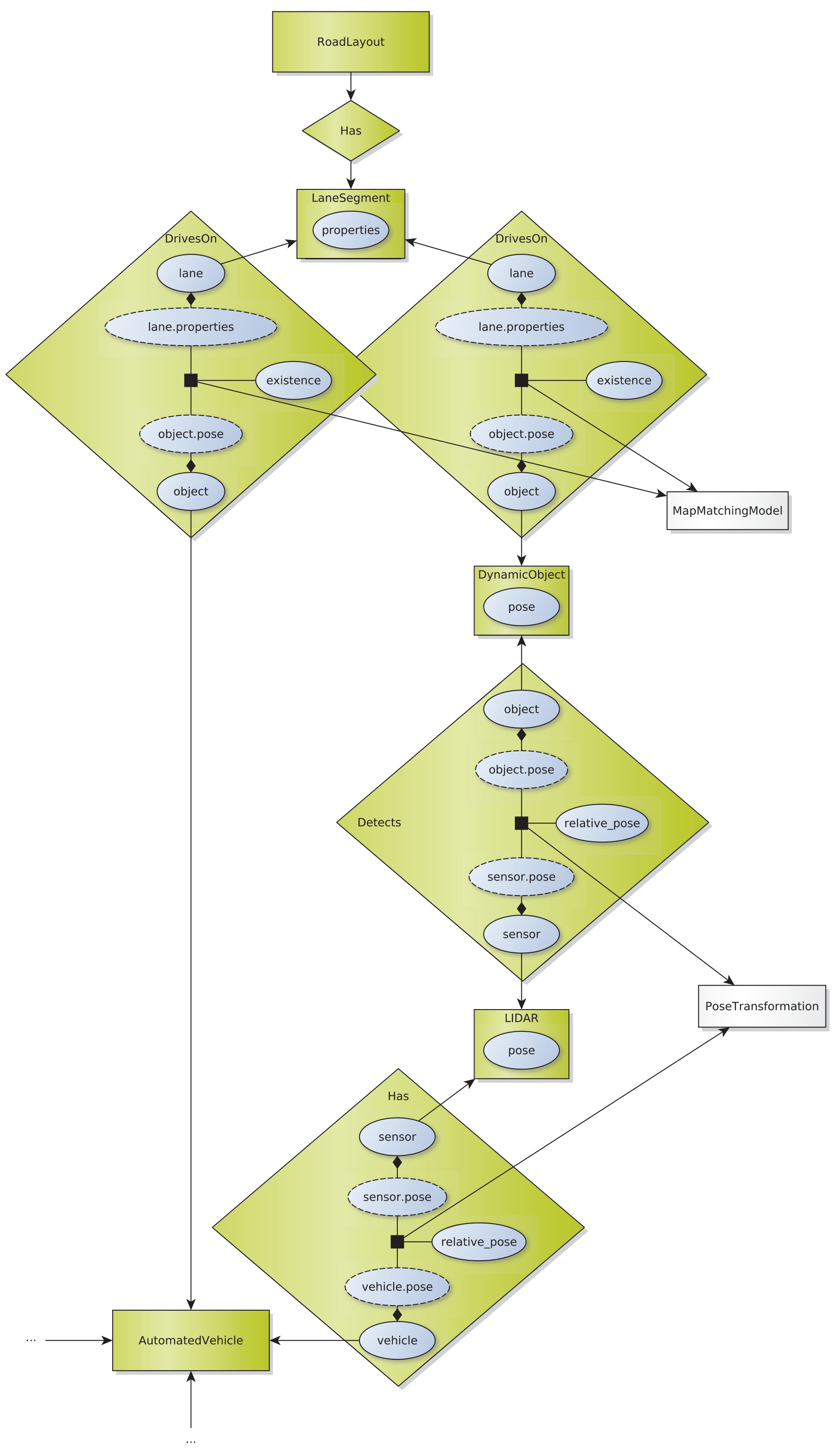


Figure 7.4: Attributes and dependency models in the upper part of the application specific ontology.

scribed `DynamicState2DTransformation` in Sec. 6.2.4 some special factors have to be mentioned in detail.

7.5.1 Odometry-based Motion Model

The odometry-based motion model in the `changesTo` relation is a `Transition Model` with additional motion measurement. The motion measurement is imported from the odometry sensor entity via some transformations.

The dependency in the factor can be described by a probability distribution over the difference between the current ego pose P_E^t and the previous pose P_E^{t-1} transformed by a motion model S_m into the space of odometry measurements δ^t:

$$\psi_1(P_E^{t-1}, P_E^t, \delta^t) = \mathcal{N}_{[\delta^t, \Sigma_{\delta^t}]}\left(S_m(P_E^t - P_E^{t-1})\right) \tag{7.1}$$

For S_m a half-turn-straight-half-turn motion model is used.

7.5.2 Relative Object Measurements Model

The relative object measurement model in the `detects` relation connects the object poses P_O of observed vehicles to the sensor pose P_S using the object measurements y_O.

$$\psi_2(P_S^t, P_O^t, y_O^t) = \mathcal{N}_{[y_O^t, \Sigma_{y_O^t}]}(P_O^t - P_S^t) \tag{7.2}$$

7.5.3 Map Evaluation Model

The map evaluation model `MapMatchingModel` in the `drivesOn` relation connects the dynamic object poses P_O (used for ego and observed objects) to the map data L. In general, this can be described as

$$\psi_3(P_{E/O}^t, L) = \mathcal{P}_{[L, p_L]}(P_{E/O}^t) \tag{7.3}$$

$\mathcal{P}_{[p_Y]}(x)$ can be any multi-dimensional continuous probability distribution with distribution specific parameters p_Y, evaluated on point x. The distribution on the map $\mathcal{P}_{[L,p_L]}$ is described by several uniform and gaussian distributions depending on the area in the map. For example lanes are modeled with a gaussian distribution around the lane's center while free area is modeled with a uniform distribution. Thus, the lane-detailed map data is used as a zone map similar to [102].

7.6 Inference Methods

Since evidences will be given from odometry sensor and environment perception and the ego pose has to be estimated over time it is straightforward to implement a state space model similar to the simple GNSS example in Sec. 6.3.3. Since the map evaluation is nonlinear a simple Kalman filter approach cannot be applied. The `ParticleFilter` sub class of the state space model implementation (Sec. 6.2.5) is chosen.[1] The resulting inference-dependent classes are depicted in Fig. 7.5.

Large computation effort lies in the map evaluation model (Sec. 7.5.3). If the association between object poses and lane segments is unclear, every pose hypothesis has to be evaluated against every map element. The gating approach (see Sec. 2.6.6, Fig. 2.17) is applied here using different sub classes of the dependency model: One class roughly estimates the existence of an object-lane relation by comparing their positions. Subsequently, only a small subset of lane segments of the map has to be evaluated in detail against the object pose using the gaussian distribution around the lane's center.

7.7 Implementation Opportunities

One of the five implementation opportunities described in Sec. 3.4 has to be chosen. Similar to the GNSS example (Chapter 6) the localization application is manageable and has no dynamic components (besides the varying number of objects). Thus the same considerations can be undertaken as in the GNSS example (Sec. 6.4).

For the experiments in this chapter an *established filter implementation* cannot be used directly: A simple particle filter implementation is not applicable since the dynamic change in number of objects and the map model evaluation have to be integrated. To avoid computational effort induced by an unnecessarily advanced implementation, a *specialized implementation* is proposed, integrating the object-related map evaluation in a particle based temporal estimation of the ego vehicle's pose.

During implementation also some dependency models can be combined. Fig. 7.6 shows a reduced factor graph with inference schedule.

[1]The necessary sub classes can be derived by applying the particle filter principles (Sec. 2.6.4) analogously to the application of the Kalman filter principles (Sec. 2.6.3) in the example in Sec. 6.2.5.

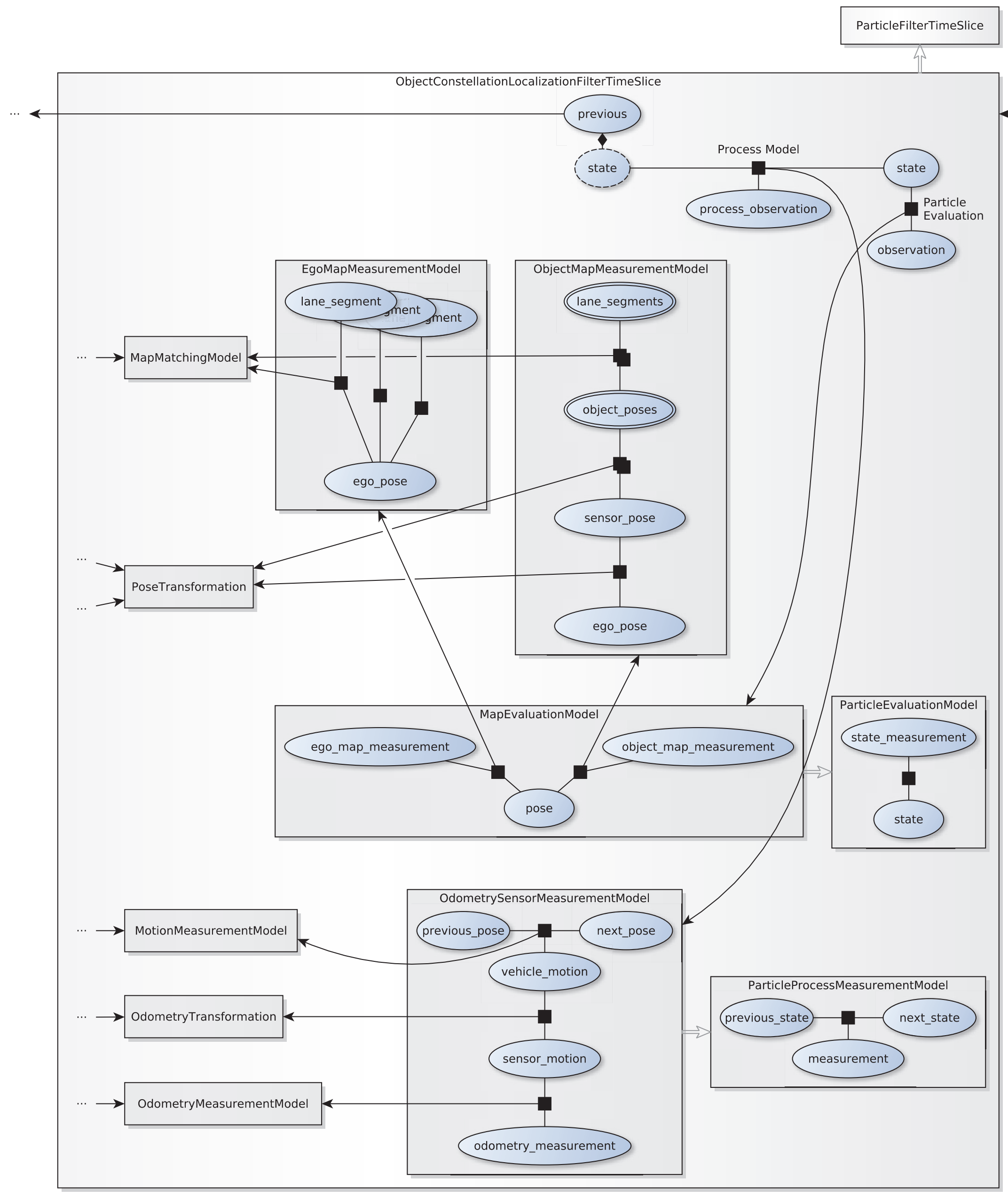


Figure 7.5: The particle filter sub class of the state space model is chosen for inference. The model classes on the left correspond to classes in Fig. 7.3 and Fig. 7.4.

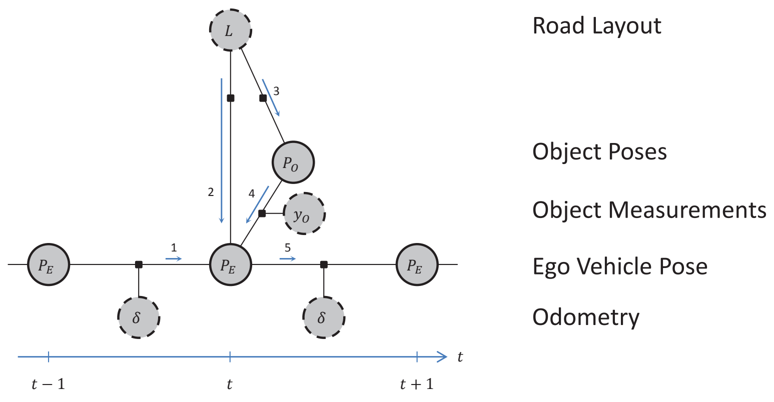


Figure 7.6: The defined model can be reduced to a small factor graph. The inference schedule is depicted by numbered arrows. Adapted from [7].

7.8 Evaluation

To evaluate the performance of the application, the described model is implemented as the proposed *specialized implementation* and integrated into a real automated vehicle. The localization results are compared to a baseline map matching algorithm.

7.8.1 Experiment Configuration

The test vehicle CoCar (cognitive car) [74] of the FZI Research Center for Information Technology is used to perform the experiments. Besides the car's odometry data, a set of three 4-layer Ibeo laser sensors including built-in object detection and classification is used as object sensor. The map data is described via *lanelets* [27].

The system is evaluated as a proof of concept on a 10 minute drive in the city of Karlsruhe. The route (about 5.56 km) includes different driving situations, such as intersections, straight roads, multiple parallel lanes, curves and also light and dense traffic. It gives a first impression of the performance but more extensive data is necessary for a profound evaluation.

7.8.2 Metrics

One key metric for the quality of the localization method is the position error of the calculated ego vehicle position compared to a ground truth reference value calculated as euclidean distance. As ground truth a position estimate of a high precision inertial measurement unit is used. Note that this ground truth estimate does not completely match the application goal: The localization method shall give a precise-as-possible position estimate on a given map in a way that the static environment objects described in the map match the real world in the vehicle's surrounding. Thus, for the real application the map just needs to be complete and consistent. For the evaluation here (using the ground truth estimate) it has to be ensured additionally that the map is accurate according to the reference sensor.

The position error is calculated as lateral, longitudinal and overall error. *Longitudinal* means only the component of the position error that is parallel to the ground truth vehicle orientation[2] is considered. For *Lateral* only the orthogonal component. Assuming the vehicle mainly driving in the roads direction, a small longitudinal error corresponds to a small position error along the road and is needed for example when estimating how far the ego vehicle is away from an intersection. A small lateral error corresponds to a small position error perpendicular to the lane orientation and is needed for estimating on which lane the ego vehicle is driving.

The error of the localization method is compared to the error of a baseline method. As baseline method a similar localization method but without considering the objects in the vehicle's surrounding is chosen. This method is known as *map matching* and used in many current GNSS navigation system solutions. It is implemented by removing the object evaluation part of the application. Thus, the basic temporal fusion and ego vehicle map evaluation is exactly the same in both methods and the impact of using the observed objects in the vehicle's surrounding becomes measurable.

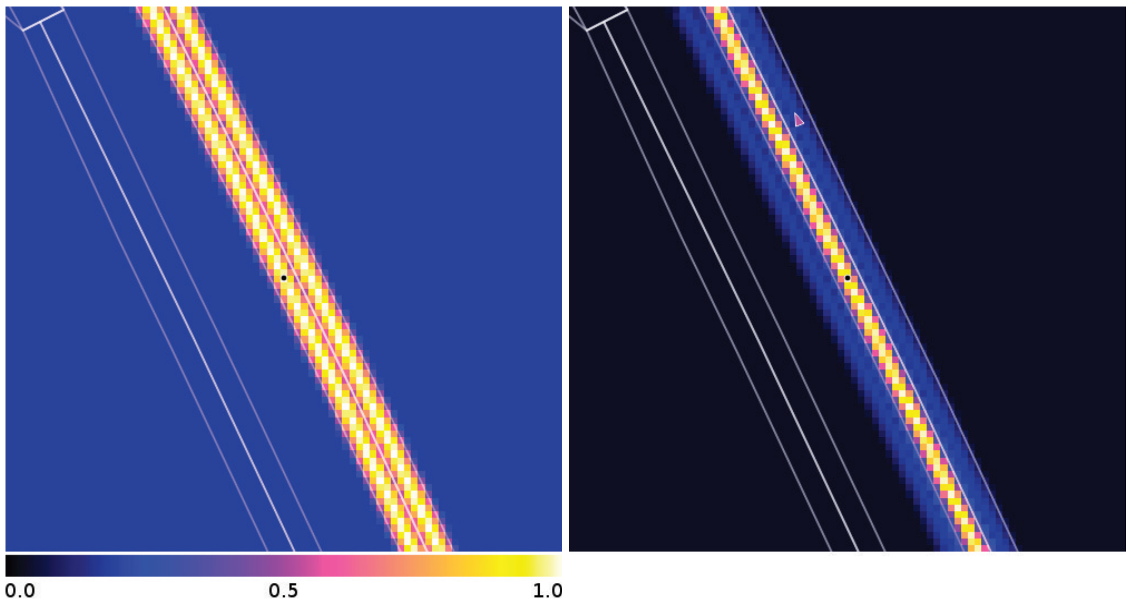
For visual impression the probability distribution of the ego vehicle's pose given the map sensor model on an area of 50 x 50 meters around the ego vehicle is depicted as heatmaps (Fig. 7.7, 7.8, 7.9, 7.11). Note, that this is an additional computation step run offline. Online, only a set of 250 particles is used to evaluate the probability distribution just in the most probable regions.

7.8.3 Single-Frame Evaluation of Map Evaluation Model

Inspecting the heatmaps of the ego vehicle position estimate at different situations corresponds to the expectations:

When driving on straight roads with multiple lanes, without considering the object constellation no precise estimation is possible (Fig. 7.7a). With observed vehicles on parallel lanes the estimation can be narrowed to one specific lane (Fig. 7.7b). This means the lateral error can be reduced when observed vehicles drive in the same direction as the ego vehicle but with a lateral offset. There is no improvement in longitudinal error.

[2]It could be argued that actually the lane's orientation should be used. This would introduce another requirement on the precision of the map data and a matching procedure. A solution without map dependency was chosen.



(a) (b)

Figure 7.7: Evaluation of the map evaluation model. The heatmap includes an area of 50 x 50 meters with a resolution of 1 x 1 meter. The colors represent the mapping score from 0 to 1.0 as depicted in the color bar. A low score means a low matching probability. The ego vehicle's ground truth pose is always at the center (black dot). Observed objects are represented as pink triangles. A desired result is a high matching score at the ego vehicle's ground truth position although ambiguities can be eliminated by temporal fusion. The higher lateral uncertainty without considering the object constellation in (a) is reduced when the observed object on the right lane (pink) is included in (b). [7]

In contrast, if the observed vehicles drive perpendicularly to the ego vehicle's direction (90° or 270°), the longitudinal error can be reduced (Fig. 7.8). On curved roads a mix of both effects is visible (Fig. 7.9).

7.8.4 Temporal Fusion

When the temporally local estimates are fused over time, the effects of longitudinal and lateral position error reduction are retained and the overall error decreases.

Nevertheless, for example if no objects are in the vehicle's surrounding for a long time, the lateral error increases (Fig. 7.10). If then an object appears, the error is quickly reduced.

Compared to the localization method without considering observed objects, the new localization method can reduce the longitudinal error in front of intersections. This is visible in Fig. 7.14 at minute 1:10 and following. The method without considering the object constellation can only reduce the longitudinal error when the ego vehicle takes a turn with about 90° (resulting in a longitudinal error of the size of the previous lateral error).

Also during lane change (Fig. 7.11) the observed object constellation can help: Although the lane change might be estimated correctly without considering objects, there is a part during lane change, where the ego vehicle itself is between the two lanes and thus is located on a less probable map position. Correct particles might be evaluated incorrectly and discarded. Considering the objects in the surrounding can help to keep the correct position with a high probability (Fig. 7.11d).

7.8.5 Overall Performance

In Fig. 7.12 the overall position error of the two methods during the test drive is compared. The method using the object constellation in the ego vehicle's surrounding clearly outperforms the reference method. Assuming a lane width of 3 meters, the position error has to be smaller than 1.5 meters to be called lane precise. This is achieved for the proposed method excepting one higher peak at 9:43 corresponding to the situation in Fig. 7.10.

The diagrams for the separate longitudinal and lateral errors (Fig. 7.14, Fig. 7.13) further support this observation. Although the localization without considering the objects is slightly better in some situations the method considering the objects is more robust and has a better overall result. Also the statistical values (Table 7.1) underline this.

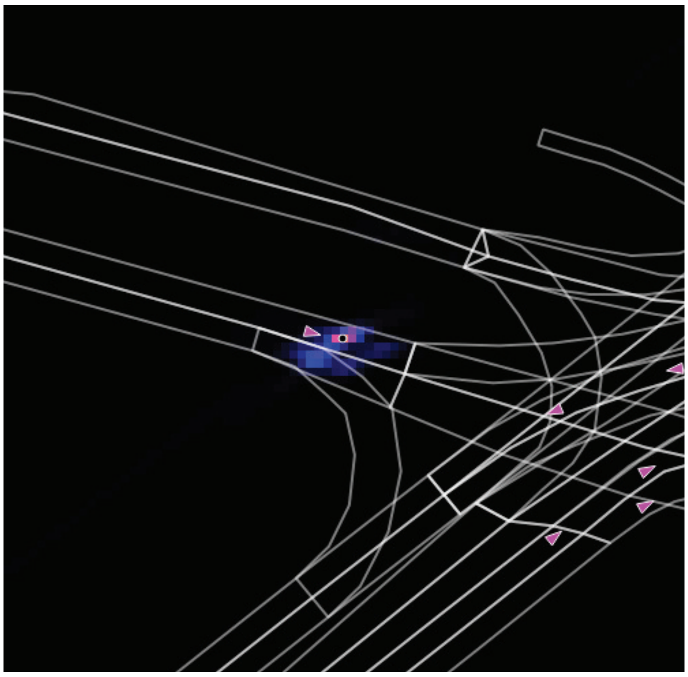


Figure 7.8: If perpendicular moving objects are present the map evaluation model returns a precise longitudinal estimation of the ego vehicle's pose with lateral uncertainty (pink and blue values) around the ground truth position (black dot). This corresponds to the principle depicted in Fig. 7.1. [7]

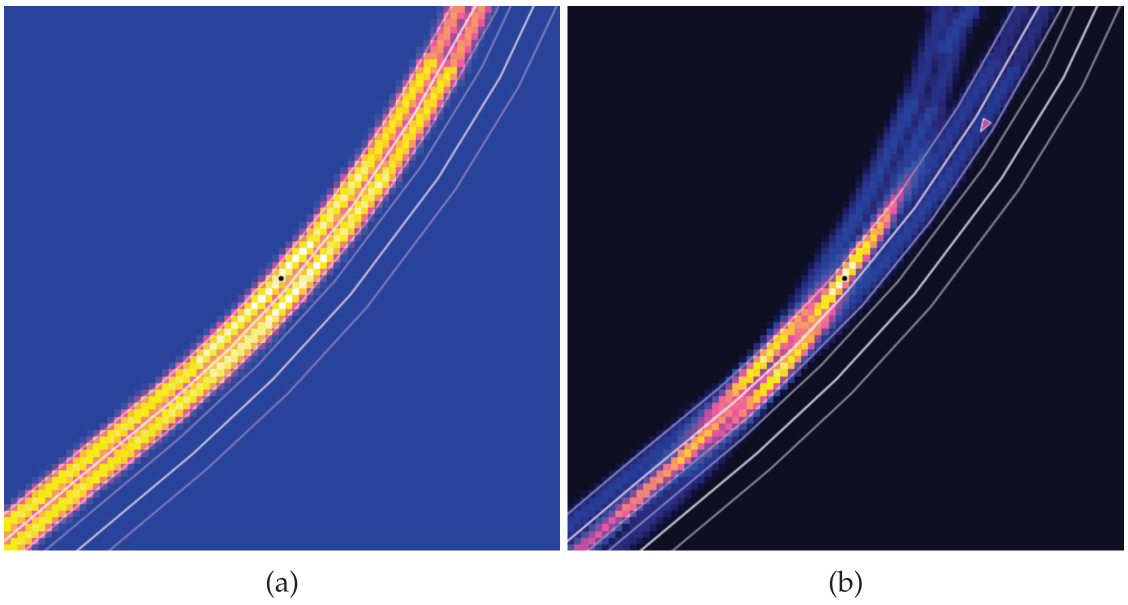


Figure 7.9: Also on slightly curved roads observing other objects improves estimating the ego vehicles pose. A mix of the individual effects (Fig. 7.7, 7.8) can be seen in (b). [7]

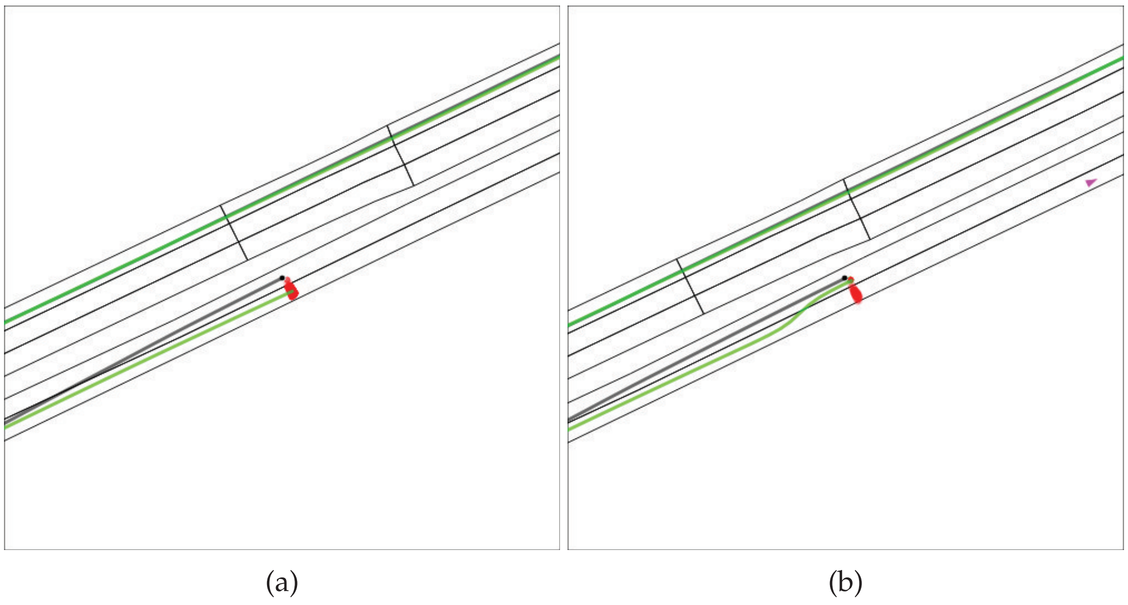


Figure 7.10: Without objects present a high lateral position error can still occur in the temporally fused output (a). After an object appears the error is reduced again (b). This example corresponds to the high peak of the overall and lateral position error in figures 7.12 and 7.13 at minute 9:43. [7]

Table 7.1: Statistical values for the position error. [7]

		without objects	with objects
mean	overall	2.1154	0.5250
	lateral	0.7093	0.4033
	longitudinal	1.7302	0.2731
std. dev.	overall	1.7296	0.2789
	lateral	0.9262	0.2716
	longitudinal	1.7641	0.2731
upper quartile	overall	4.3379	0.7062
	lateral	0.8718	0.6013
	longitudinal	4.0365	0.3754
maximum	overall	5.1647	2.4806
	lateral	3.6502	2.4496
	longitudinal	4.9896	0.9649

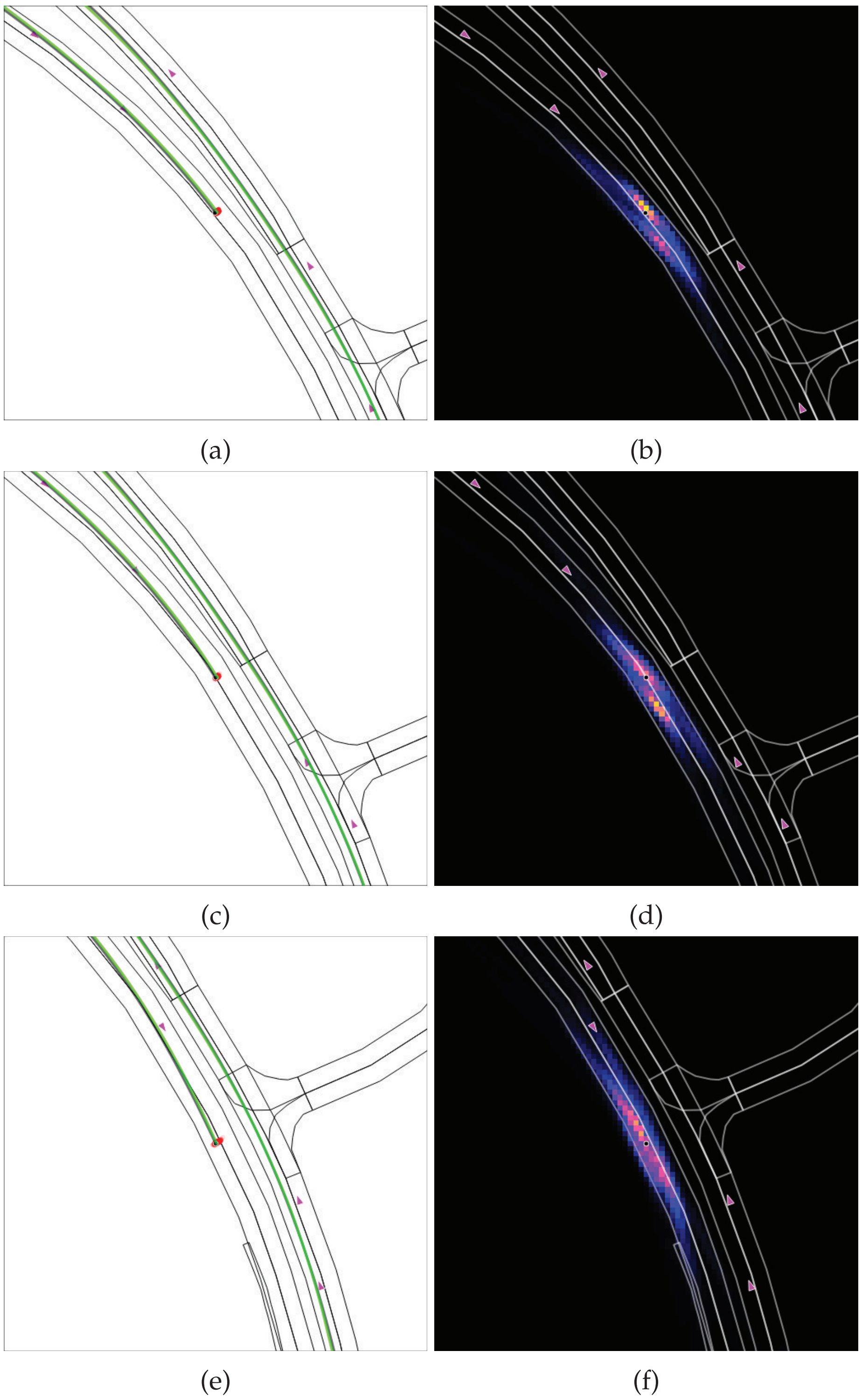


Figure 7.11: Localization results (a,c,e) and corresponding map evaluation model (b,d,f) during a lane change. Although the ego vehicle is between the lanes in (c), there is still a maximum in the current map evaluation model due to the observed objects in the vehicle's surrounding (d). [7]

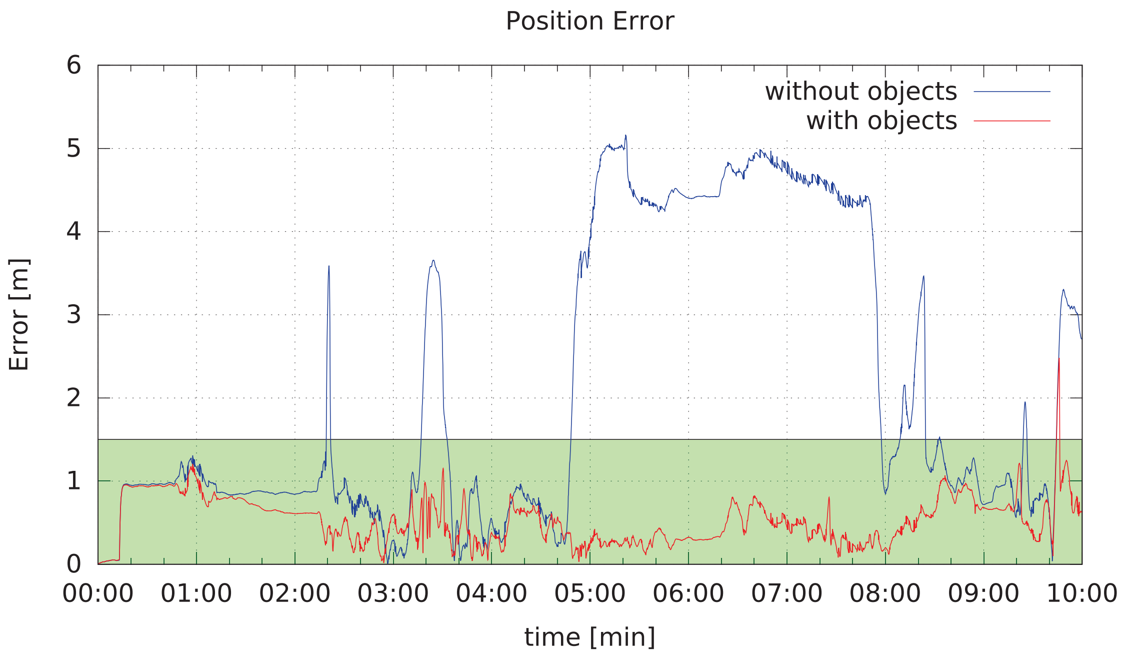


Figure 7.12: Overall position error. Depicted is the euclidean distance between the estimated position (with and without considering the objects in the vehicle's surrounding) and the reference position from the reference sensor. High position errors are decreased by considering the object constellation. The position error is always below 1.5 meters, meaning a lane-precise estimation is achieved. Except for a higher peak at minute $9{:}43$, where no recognized objects are in the vehicle's surrounding. This situation is depicted in Fig. 7.10. [7]

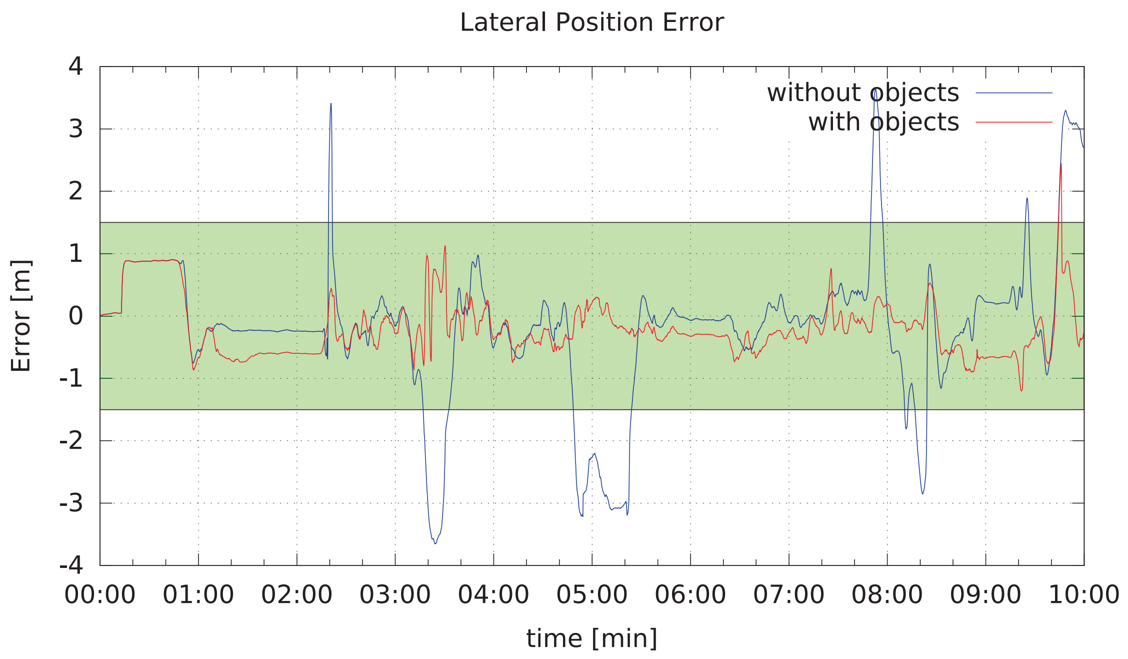


Figure 7.13: Lateral component of the overall position error (Fig. 7.12). localization without objects is slightly better at some times, but localization considering the objects in the vehicle's surrounding is more robust and always in the range of -1.5 to 1.5 meters, except a higher peak at minute $9{:}43$ due to no recognized objects (Fig. 7.10). [7]

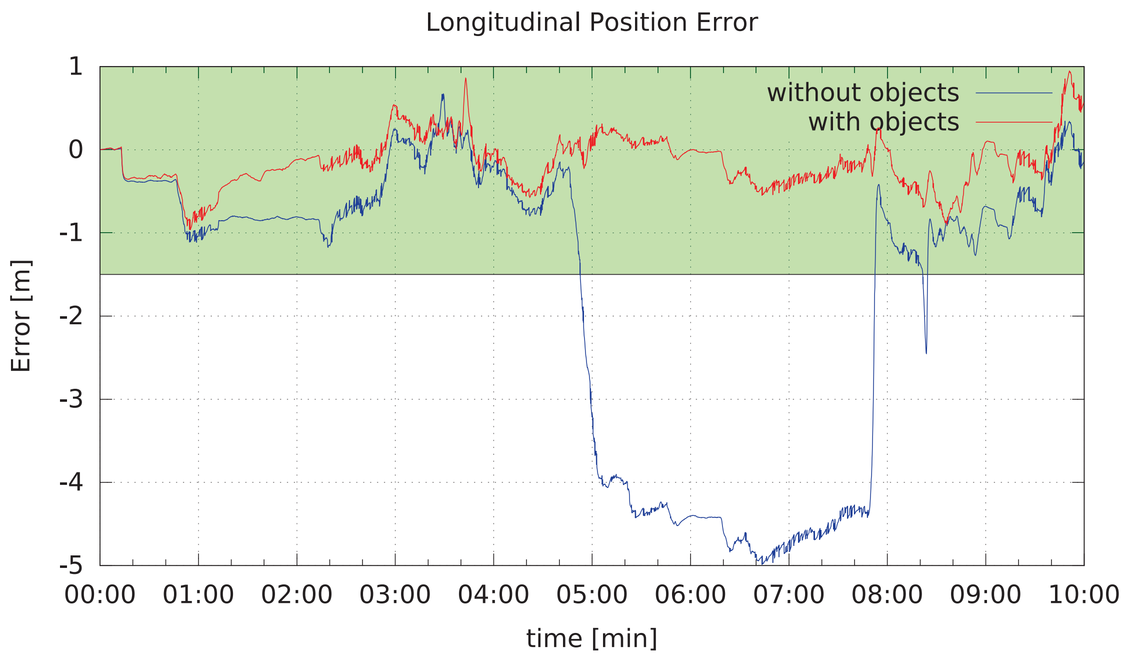


Figure 7.14: Longitudinal component of the overall position error (Fig. 7.12). Considering objects in the vehicle's surrounding results in an improved ego vehicle localization that is always between -1.5 and 1.5 meters around the reference position for this exemplary scenario. [7]

7.9 Conclusion

This chapter showed how an advanced method for ego vehicle localization can be modeled using the proposed language.

7.9.1 Summary

The key parts are:

- Compared to the related work the proposed localization method is new.
- From a real world oriented application description, the application ontology can be derived, based on the generic DSL for the traffic domain.
- Advanced dependency models can be integrated straightforwardly into the class descriptions.
- A particle filter implementation is realized as an alternative sub class of the state space model base class and is a good choice for complex probability distributions.
- The implementation as a specialized implementation allows staying close to generic filter implementation with added adaptations.

- Comparing the resulting application to a baseline implementation shows that considering objects in the ego vehicle's surrounding improves the overall localization result.

7.9.2 Used Language Properties

The key goal of presenting this application within this thesis is to exemplarily show the usage of properties (Chapter 4) of the proposed language.

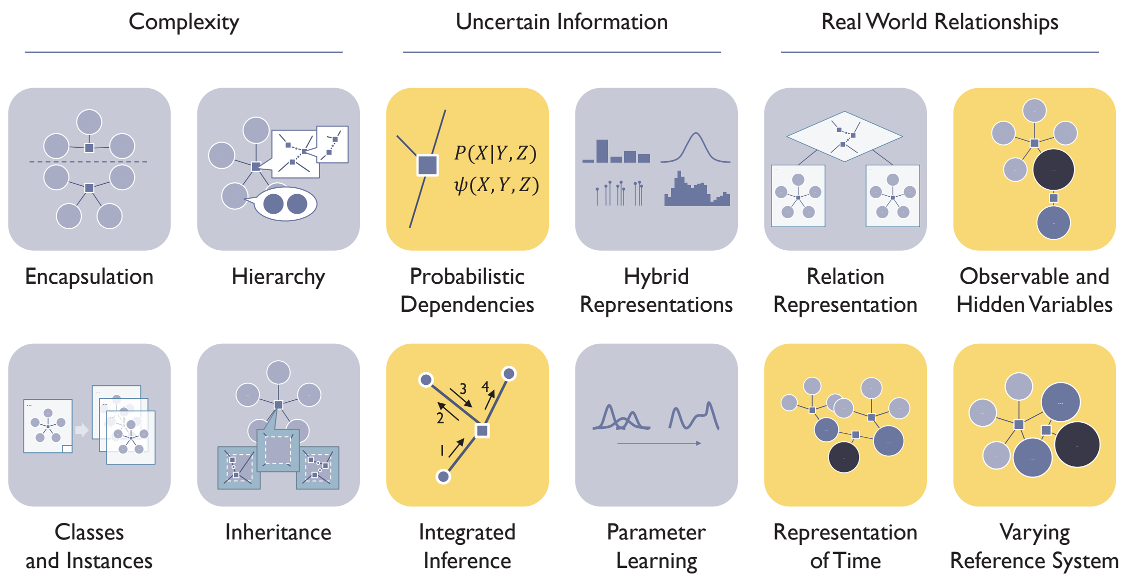


Figure 7.15: Some of the overall language properties are highlighted that are beneficial to the presented localization application.

In this application the following properties (Fig. 7.15) are beneficial:

- **Probabilistic Dependencies** (Sec. 4.2.1):
  By describing the dependencies probabilistically with uncertainties (Sec. 7.5), the uncertain behavior of other road users and uncertainties in map data can be incorporated.

- **Integrated Inference** (Sec. 4.2.3):
  The language's capability of describing inference methods in a class hierarchy allows picking of the most suitable inference method (Sec. 7.6) independently from the dependency modeling (Sec. 7.5).

- **Observable and Hidden Variables** (Sec. 4.3.2):
  The application has only a few observable variables: The odometry and the relative object measurements. All other variables are hidden and help estimating the ego vehicle's pose (Sec. 7.4).

- **Representation of Time** (Sec. 4.3.3):
  A key functionality of this application is the temporal fusion of only inaccurate position estimates. The modeling language highly supports the temporal aspect and allows direct derivation of proper models (Sec. 7.4).
- **Varying Reference Systems** (Sec. 4.3.4):
  While the object measurements are local, relative measurements, the ego vehicle's pose and the map are in a global coordinate system. These two perspectives are brought together using the dependency models (Sec. 7.5).

Other properties will be used more intensively in one of the other applications described in the subsequent chapters and thus will be highlighted there.

8 Road Layout Estimation

This chapter shows how an example application considering road layout estimation can benefit from the introduced OOFGML. A special road layout estimation method is created that estimates the most probable road layout from several sensor inputs. This means various observed environment features are brought into relation to derive the position of road elements such as lanes and intersections. The idea is based on a popular work of Daniel Töpfer [124] but adapted to the requirements for the small-scale vehicles in the Audi Autonomous Driving Cup (AADC). The OOFGML supports the whole development process from application description to algorithm implementation.

The goal of this chapter is to outline how the features of the OOFGML can be utilized to achieve the application: It is shown that all relevant aspects are describable in a consistent way. Nevertheless also the performance of the resulting application is discussed by observing the results when participating in the AADC.

This chapter is largely based on the previous publication [8]. Some more details considering the application can be found there.

8.1 Related Work

Road layout perception has always been a central research topic for autonomous vehicles. Before handling obstacles and deciding about future trajectories, the static environment in the ego vehicle's surrounding has to be localized precisely. Maps [27], [72], [47], [11] can simplify this task, but they can be inaccurate or outdated and require a very precise localization. Therefore, perceiving the static environment, especially the lanes vehicles can drive on is a key skill since years.

In ADAS, especially in Lane Keep Assistance (LKA) systems, at least the ego vehicle lane needs to be estimated. Therefore a lane model is used that describes the ego vehicle's lane or includes neighbour lanes and constellations. Common approaches [41], [46], [107] usually follow the first ideas of [42]: A geometric model is estimated using observed sensor cues, such as lane markings, curbs, etc.. The model can be fitted in the camera frame or the sensor cues are transformed into a bird eye view [88]. The later approach has the advantage of better understandable parameters for the geometric model and also subsequent processes usually use a local 2D coordinate system to handle vehicle behaviors.

While many approaches [34], [70], [31] use Random Sampling Consensus (RANSAC) [48] to properly match a lane model neglecting outliers in sensor data, more recent approaches [67], [124] use graphical models, e.g. Markov Random Fields. Besides estimating the ego vehicle's lane, these approaches have the potential to estimate whole road layouts, including merging and diverging lanes and intersections using multiple sensory inputs, such as lane markings, curbs, stop lines, traffic signs and lights. A well-chosen road model can then also be used to combine it with geometric map data or even to update such a map [62].

The approach proposed in this work is largely based on Markov Random Fields, so-called Compositional Hierarchical Models (CHMs) introduced by Daniel Töpfer [125], [124], [126].

8.2 Application Goal

The goal of the application is to estimate a consistent road layout in the vehicle surrounding using several environment observations (Fig. 8.1).

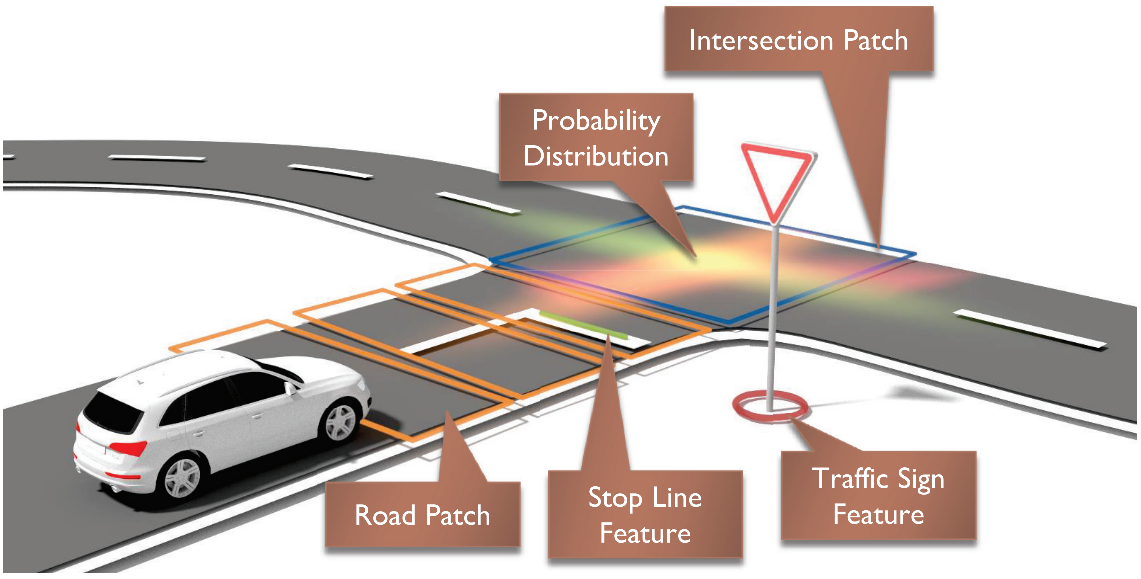


Figure 8.1: A probabilistic hierarchical model is used to estimate the road layout: Observable features like stop lines (green) or traffic signs (red) and road patches (orange) are brought into relation described by spatial constraints to infer higher level objects like intersections (blue). [8]

The basic idea, introduced by Töpfer [124] is to model all real world entities in a markov random field and design weak spatial constraints between these entities that describe their relative positions, e.g. a lane marking is usually at the boundary of a lane. Longitudinally the lanes are split into patches to deal with changes in lane width or curvature. The lane features build up patches, which build up lanes, which build up multilane roads and finally lead to whole road layouts including intersections. This generative process is called Compositional Hierarchical Models (CHMs) and with the aid of part-sharing [137] and depth-first message passing [125] Töpfer is able to estimate real world highway and urban scenes even on a per image evaluation without temporal fusion.

Due to the limited sensor in the AADC there are many occlusions and missing features especially in curves. Therefore temporal fusion is beneficial, integrated and a special grid-based message representation is used where useful. The temporal fusion introduces a dependency to the ego vehicle's localization (since the observing vehicle moves over time) and thus makes it necessary to treat the formulated problem with a more complete approach.

Key idea of this application is to perceive environment *features*, such as different longitudinal lane markings, stop lines and traffic signs, and use them to estimate different kinds of *patches* that are not directly observable but build up the road environment, such as road patches, intersection patches and parking lot patches. Spatial and temporal constraints have to be modeled between these elements to describe a consistent road layout.

8.3 Application Specific Ontology

From the DSL (Chapter 5) an application specific ontology has to be derived to represent the conditions of the application (Sec. 8.2). From the entity and relation class structure of the DSL for the traffic domain (Fig. 5.3) appropriate entities and relations have to be chosen to bring over to the application specific ontology (Fig. 8.2).

For this application several entities and relations have to be included: The `AutomatedVehicle` and the `RoadLayout` correspond to the real world entities. The `RoadLayout` includes the road hierarchy consisting of `LaneSegments` and `Lane Features`. These correspond to the aforementioned patches and features. The `detects` relation describes the fact that lane features are observed by the ego vehicle. The `changesTo` relation represents the temporal fusion of the ego vehicle pose estimate as well as the static road elements. Since the ego vehicle moves over time, an inertial sensor is used to estimate the local movement and filter it by a state space model, similarly to the GNSS example in Chapter 6.

Analogously to the previous example applications (Chapter 6, Chapter 7) this basic ontology is extended by sub classes and helper classes to fully define the estimation problem, approach and inference method.

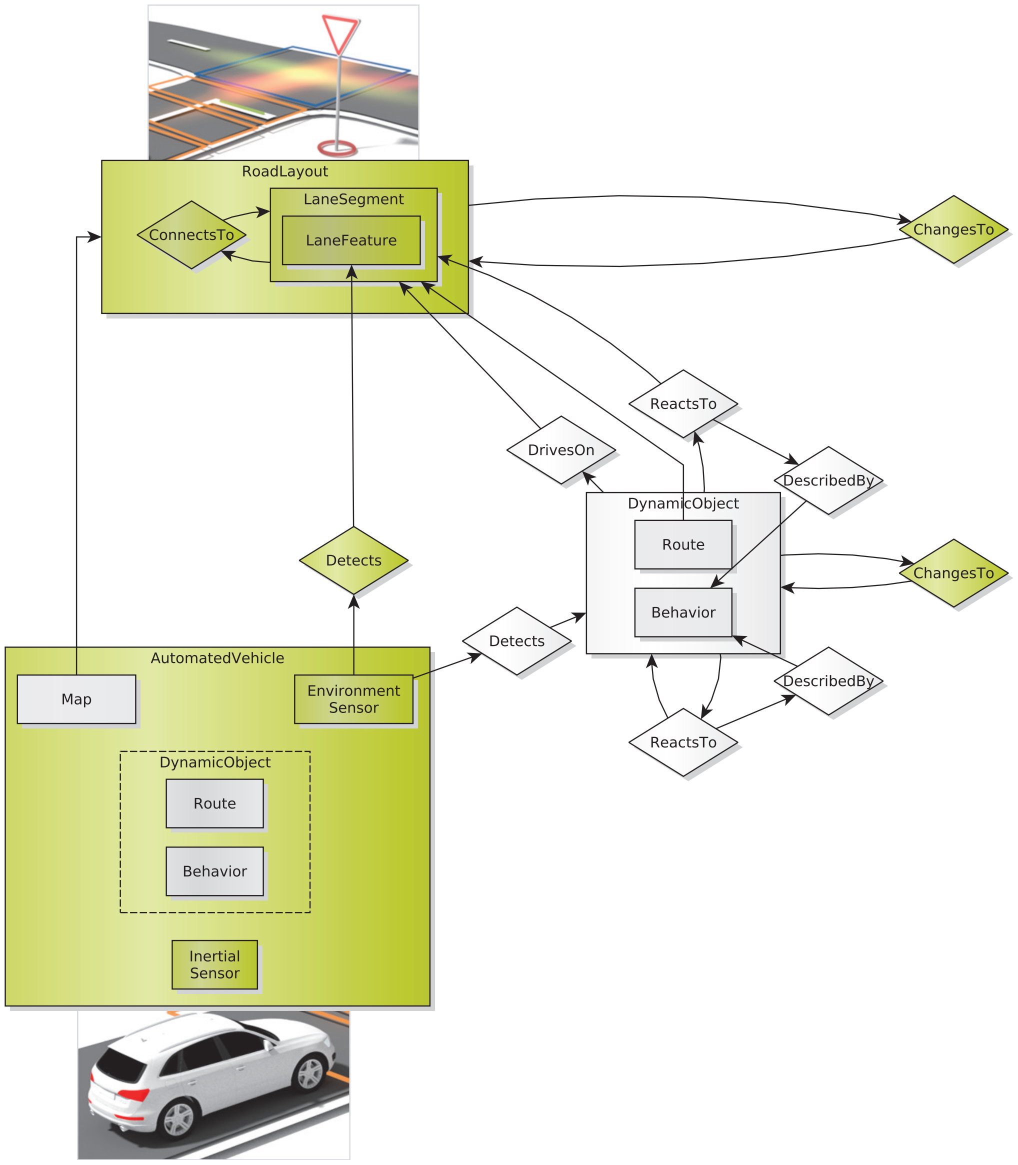


Figure 8.2: The necessary entities and relations comprise the ego vehicle, dynamic objects and the road layout.

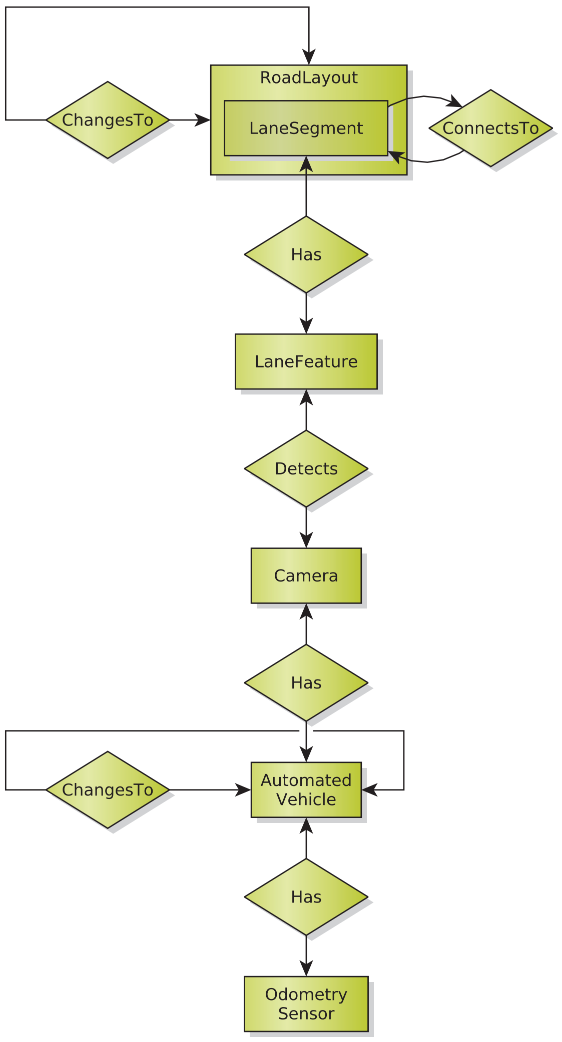


Figure 8.3: More detailed view on the application specific ontology. The temporal `ChangesTo` relations are acting on the dynamic vehicle state and the static road layout. Focus will be on the road layout estimation.

8.4 Classes and Attributes

The classes describing the entities and relations are equipped with attributes and dependency models to describe the real world properties (Fig. 8.3). The lower part of the figure including the ego vehicle localization and feature perception has already been elaborated in detail in the previous applications (Chapter 6, Chapter 7) and will be used similarly. Thus, the upper part concerning the road layout is focused. The important entities here are the `LaneFeatures` and the `LaneSegments`.

8.4.1 Lane Features

All lane features match the possible sensory evidences observable from the environment. They consist of a position $(x, y) \in \mathbb{R}^2$ and orientation $\phi \in [0, \pi)$ in a world coordinate system and a feature type

$$\tau_f \in T_f := \{\text{center\_line}, \text{side\_line}, \text{stop\_line}, \text{parking\_line}, \text{traffic\_sign}\} \tag{8.1}$$

to distinguish between different features:

$$f = (x, y, \phi, \tau_f) \tag{8.2}$$

8.4.2 Lane Segments

The lane segments are defined in the same coordinate system as the lane features. Each lane segment fully describes a hypothesis for a single road patch. Besides position $(x, y) \in \mathbb{R}^2$, orientation $\phi \in [0, \pi)$ and patch type

$$\tau_p \in T_p := \{\text{road}, \text{intersection}, \text{parking}\} \tag{8.3}$$

the lane segments include a width w and length l:

$$p = (x, y, \phi, \tau_p, w, l) \tag{8.4}$$

Note that for application in the AADC prior knowledge can be incorporated for width and length according to the three different types of patches:

- A *road patch* represents the full width of the uniform road with an adjustable fixed length. This is sufficient since during the AADC the number of lanes, as well as the width of the road, are fixed. Road patches are not only used for straight roads – multiple road patches are used to approximate curves.
- An *intersection patch* matches the width and length of the fixed intersection size.

- A *parking lot patch* matches the width and length of parking lots which are known to be parallel or perpendicular to the road.

These entity attributes are modeled in the OOFGML using already introduced generic classes (e.g. `Pose2D` for the pose of patches and features). Note that the specialization is realized by a hierarchy of sub classes. Details are neglected since they were already discussed in Chapter 6.

8.5 Dependency Model

Several factors describe dependencies between real world entities (Fig. 8.4). They can be classified as spatial constraints between features and patches or among patches and temporal constraints over time.

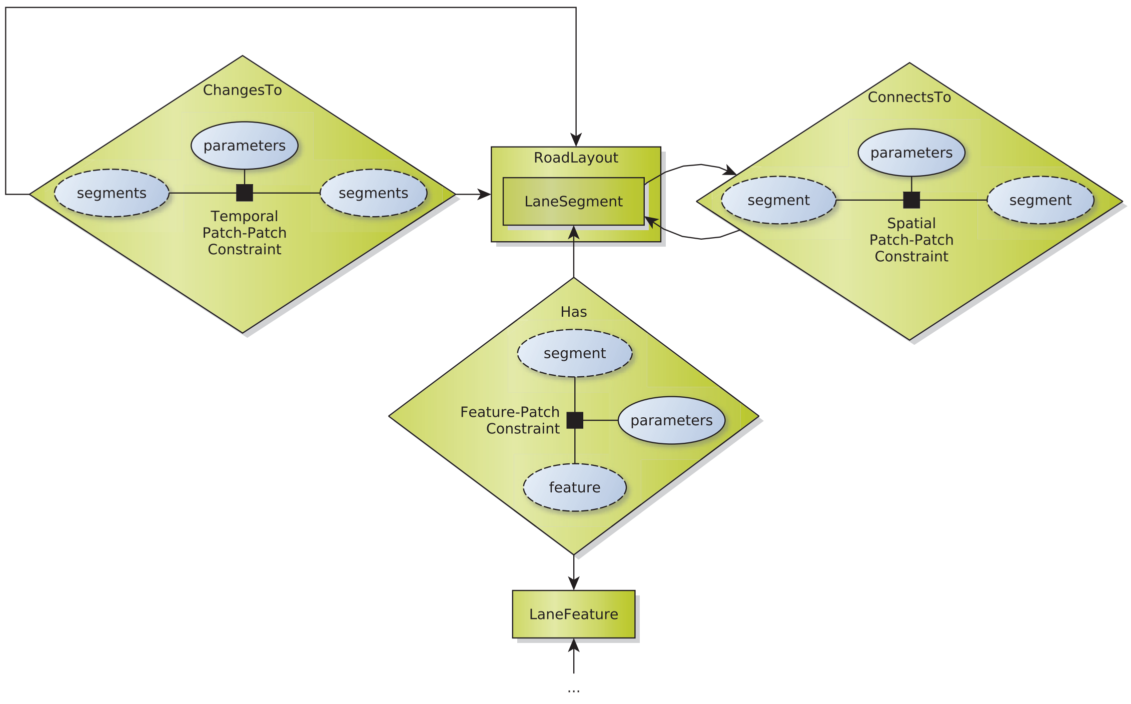


Figure 8.4: Several dependency models mainly represent spatial and temporal constraints between real world entities.

8.5.1 Spatial Feature-Patch Constraints

All feature-patch constraints describe the spatial dependencies between observable features and patches. They are represented by

$$\psi_{i,j}(f_i, p_j) = \mathcal{P}_{[\theta]}(f_i, p_j). \tag{8.5}$$

with a general probability distribution $\mathcal{P}_{[\theta]}$ modeling the relative positioning and uncertainties. This probability distribution has to be chosen to handle several correlations: Lateral features like center lines and side lines are connected to road patches with a tight lateral coupling while their longitudinal dependency has high variance (Fig. 8.5). Similarly, stop lines are connected to intersection patches and parking lines to parking space patches with high longitudinal precision. Traffic signs are located in corners of intersections but their orientation can vary. Therefore a spatial constraint in the form of a quarter circle is assigned to them, representing a large angular variance (Fig. 8.6).

All these dependencies can be defined as sub classes of the feature-patch spatial constraint factor.

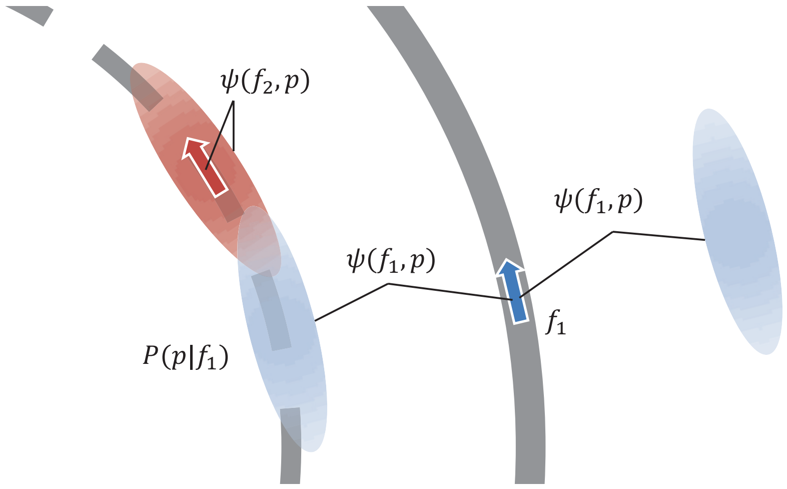


Figure 8.5: Spatial constraints ψ describe the relationship between features f and patches p. Exemplarily a side line feature f_1 (blue arrow) and a center line feature f_2 (red arrow) are shown. While the spatial constraint for the center line feature can be modeled with a single gaussian distribution, the one for the side line feature has a bimodal gaussian distribution. [8]

8.5.2 Spatial Patch-Patch Constraints

All patch-patch constraints are represented by a generic probability distribution

$$\psi_{i,j}(p_i, p_j) = \mathcal{P}_{[\theta]}(p_i, p_j). \tag{8.6}$$

Road-road constraints exist between consecutive road patches and ensure continuity between them. Road patches are also correlated to intersection patches: There is a higher probability for an intersection if patches are located at the arms of the intersection (Fig. 8.6). Parking spaces are connected similarly to road patches laterally.

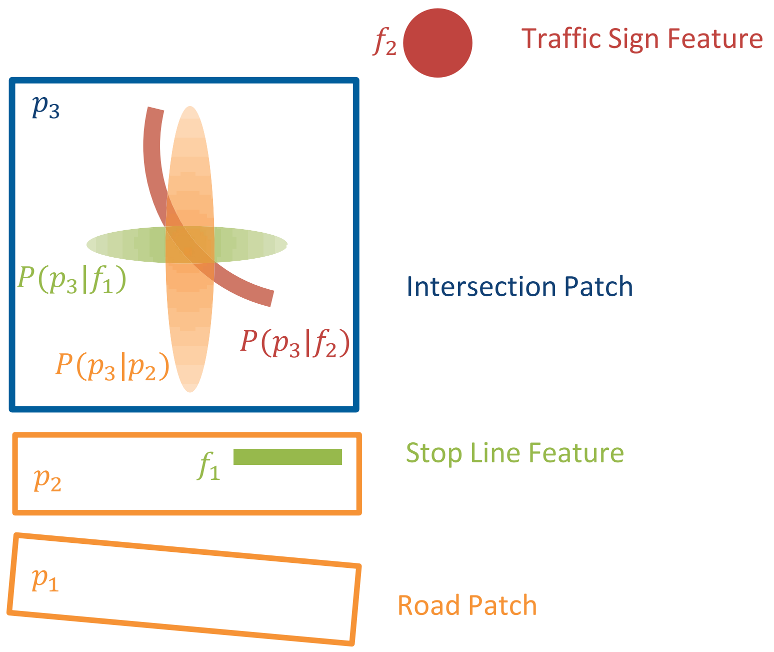


Figure 8.6: Spatial constraints for intersection patches on the example of Fig. 8.1. The combination of previously found road patches p_1, p_2, stop line features f_1 and traffic sign features f_2 give a precise estimation of the intersection patch $P(p_3|f_1, f_2, p_2)$. [8]

8.5.3 Temporal Patch-Patch Constraints

Temporal constraints realize the dependency between single time steps in the `changesTo` relation.

Because of the assumption that static environment elements do not move over time, the past patches are transmitted to the next time step by applying a temporal constraint that represents an identity transformation. This dependency can be modeled by a Gaussian potential function:

$$\psi_{T\,i,j}(p_i, p_j) = \mathcal{N}_{[p_i, \Sigma_{i,j}]}(p_j). \tag{8.7}$$

This results in a tracking of single road patches over time with a transition model representing a constant position process. Per step observation uncertainties are smoothed out.

8.6 Inference Methods

The defined model so far is usable for different applications. For example it could also be used for a localization using known map data, similarly to the application in Chapter 7. In the here described application the goal is to estimate the road layout. Therefore, a proper message passing schedule passes messages from the lower part up to the road layout at the top of the model.

To define the actual inference methods, the message representations have to be chosen. For the original CHM approach a sampling based method was chosen with a depth-first message passing schedule to overcome the large complexity in the hierarchical model. The temporal component in the given model further increases the complexity and a sampling based inference method does not have to be the best choice.

Instead a local choice for message representations is desired: In the lower part the ego vehicle pose tracking can be best estimated by linear gaussian models, estimating a pose variance over time. The feature extraction from camera images is usually implemented by a hypothesis generation with confidence values. In the upper part local lane segment hypotheses (patches) have to be created using the constraints to past lane segments and current lane features. Looking at this step in detail reveals a large association problem (Fig. 8.7). A grid representation of the lane segment space is chosen to overcome this problem (which can also be seen as a general hough transformation). The actual road segments in the road layout are represented as single values and are sampled from the grid space.

The grid space is used as follows: Since width and height of patches are given by prior knowledge (Sec. 8.4.2), the unknowns to estimate for a patch hypothesis are the position and angle as well as the type of the patch:

$$p \in X \times Y \times \Phi \times T_p \tag{8.8}$$

All four dimensions are discretized in a way that a proper resolution around the ego vehicle is achieved (see [8] for more details). The spatial and temporal constraints are applied on the features and patches and their results are projected into the grid (Fig. 8.8). Actual lane segments are sampled from the grid space by searching for maximas. The grid representation allows further performance improvements by using rectangular block distributions with separate blurring and working in the log-space [8].

8.7 Implementation Opportunities

One of the five implementation opportunities described in Sec. 3.4 has to be chosen. The road layout estimation application is more complex than the previous applications (Chapter 6, Chapter 7). Different message representations were chosen for different parts of the model and they can be separated straightforwardly. A *modular implementation* is an obvious approach.

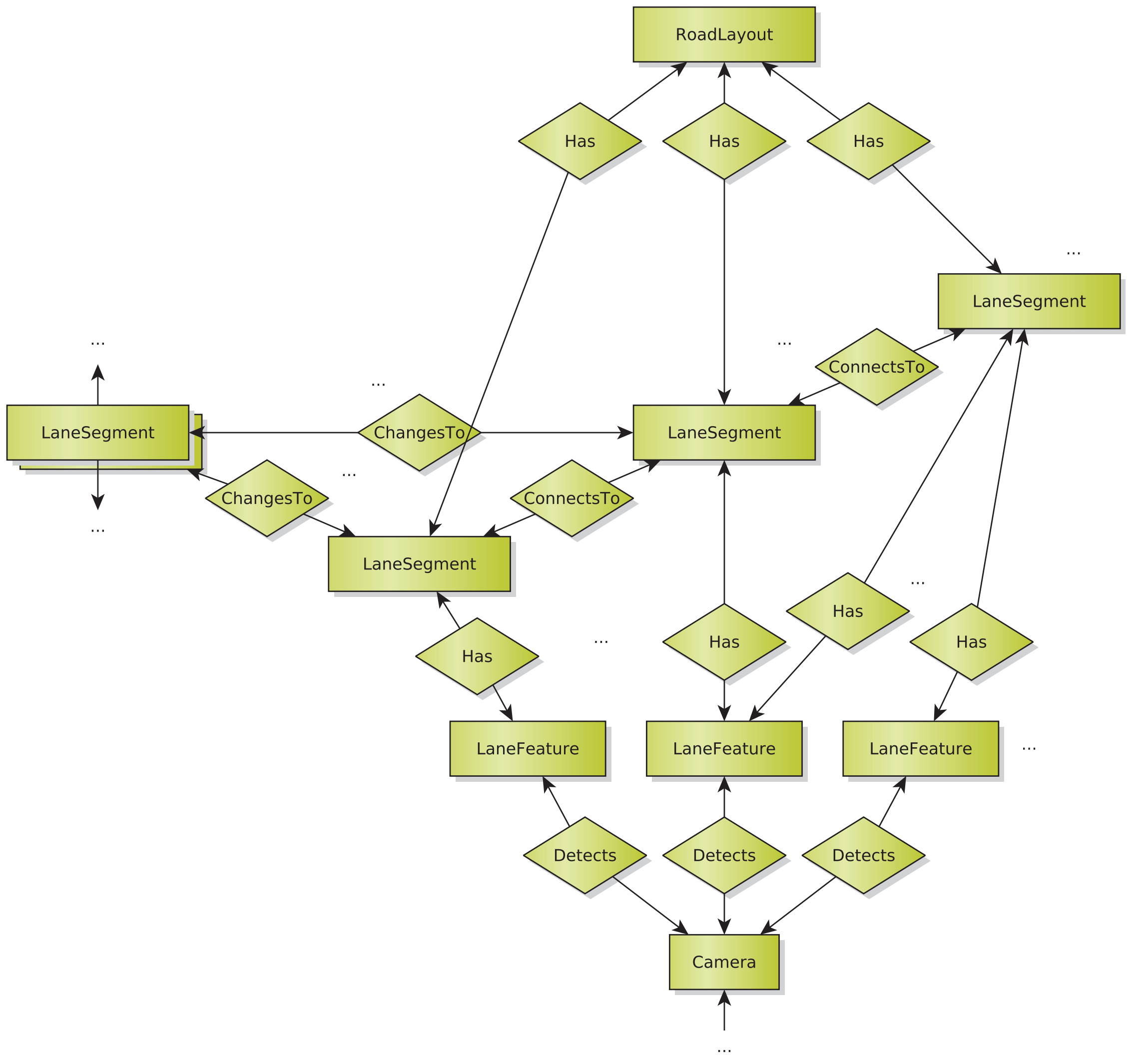


Figure 8.7: A detailed view on the lane segments reveals an association problem. Every `LaneFeature` could have a relation to every `LaneSegment` hypothesis. Same holds for spatial and temporal patch-patch constraints.

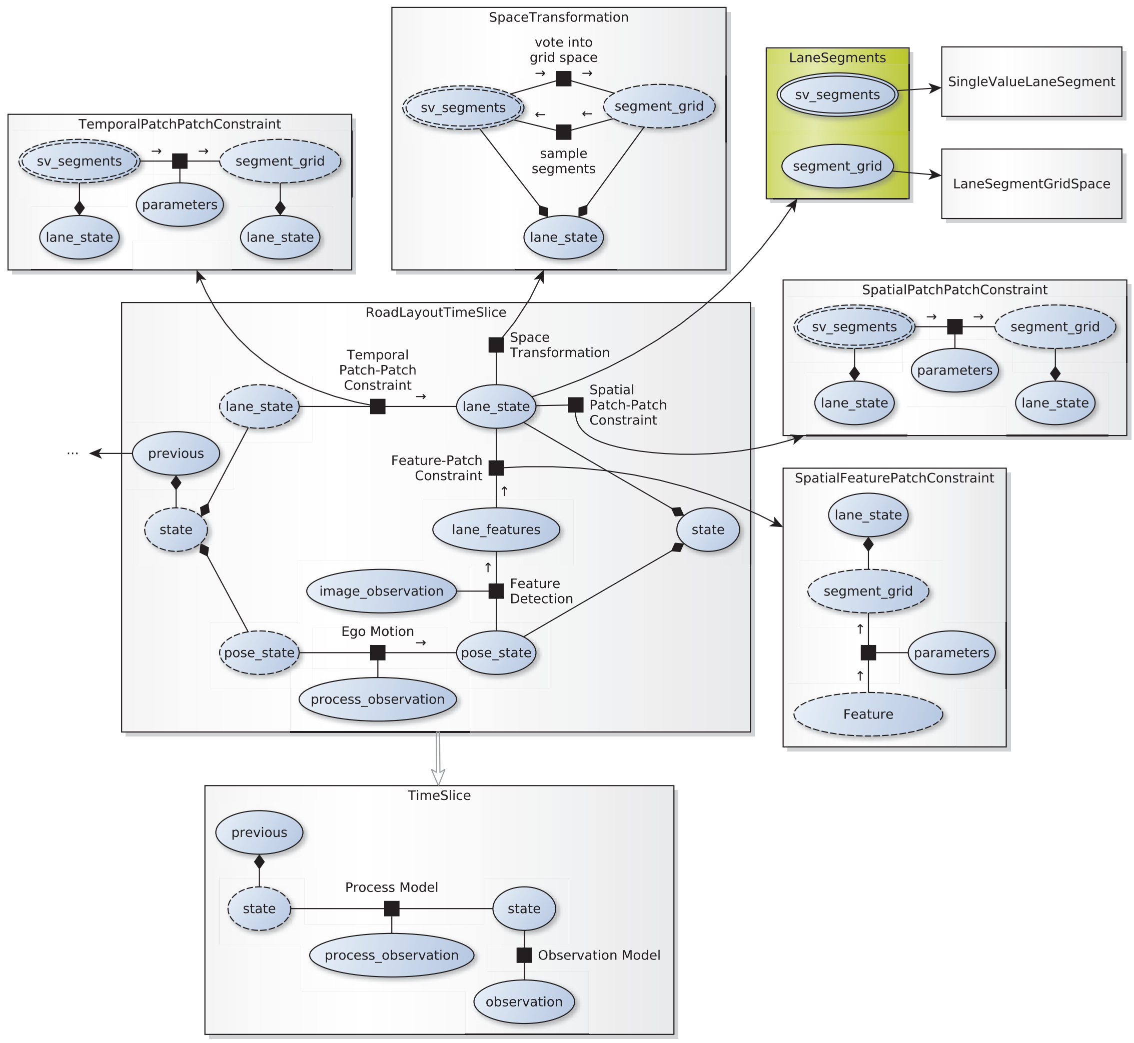


Figure 8.8: The conversion from and to the grid space can be represented in the OOFGML. The central `RoadLayoutTimeSlice` inherits from a generic `TimeSlice`. All dependencies are defined in separate classes. Key element ist the `LaneSegments` class (green) with two probability distribution representations. Inference directions are depicted by small arrows.

For the experiments in this chapter the ego vehicle pose estimation is implemented separately using an established filter implementation. The feature extraction is managed using well-known computer vision algorithms. Finally, the grid-based fusion is a specialized implementation.

The OOFGML coherently describes the overall model which is then implemented in smaller modules.

8.8 Evaluation

The approach was evaluated at the Audi Autonomous Driving Cup 2016. The student team[1] implemented a full system that enables autonomous driving capabilities for the small-scale vehicles. For the robust environment perception they followed the approach described in this chapter and in [8]. In the finals they were able to test the approach on a test track with various advanced challenges like glare light, missing road markings, a tunnel, snow and additional markings at a pedestrian crossing.

The results can be seen as a first qualitative evaluation of the approach. For profound evidence further, also quantitative evaluation on a larger dataset using real-scale vehicles is necessary.

8.8.1 Experiment Configuration

The AADC is an international student competition realized annually. Teams of 5 students participate in the cup. A fixed hardware setup is provided and the students have a 6 months period to implement a full autonomous driving functionality that can handle several tasks set in the cup.

Basic tasks like emergency stopping, overtaking or handling intersections are already required in the base competition. In the final the three best teams face additional unforeseen challenges like missing lane markings, tunnels or new situations like pedestrian crossings.

The cars are equipped with several sensors and actuators that simulate real-scale sensors for the small scale vehicles (e.g. ultrasonic sensors instead of radar sensors). The environment consists of road tiles of the size 1x1 meters that are arranged to build straight and curved roads, intersections and parking spaces.

Team KACADU created a modular system architecture that includes the here proposed robust road layout perception modeled using the OOFGML (Fig. 8.9).

[1]team KACADU consists of Vitali Kaiser, Jan-Markus Gomer, Micha Pfeiffer, Peter Zimmer and David Zimmerer

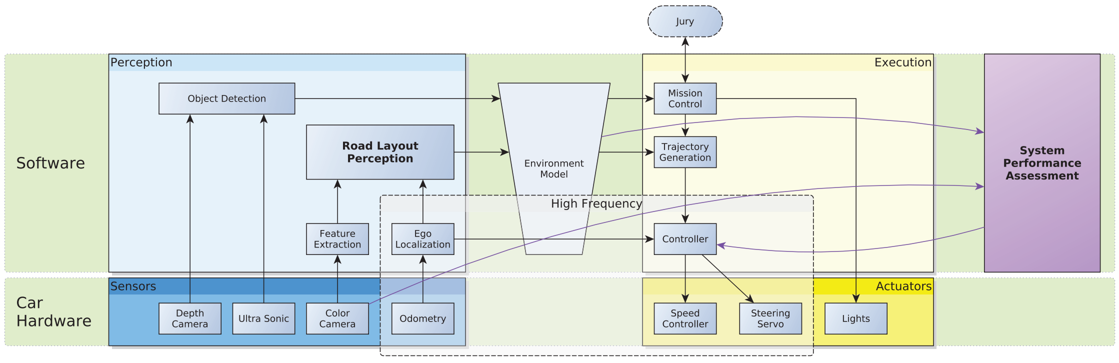


Figure 8.9: The system architecture follows the approach of an *intelligent agent* (Sec. 2.2): Perception provides an environment model that can be used by execution modules. The here proposed approach using the OOFGML includes the three modules Road Layout perception, Feature Extraction and Ego Localization. For robustness, a *system performance assessment* surveils the components to react in case of failures. This is not part of this thesis and further described in [8].

The feature extraction is implemented in two sub modules:

- For **line features** a chain of steps including a birds eye transformation and haar-feature detection is used (Fig. 8.10)
- For the marker-equipped **traffic signs** the ArUco library [94] is used. It delivers the class and 3D pose of each traffic sign in the coordinate system of the camera.

With this setup, team KACADU was able to perform in the Audi Autonomous Driving Cup 2016 and won the overall second place with an additional first place award for the scientific presentation that largely handled the robust road layout perception. The final run (with the advanced challenges) was recorded during the competition and evaluated afterwards. A video summarizes the most challenging tasks and is available for download[2]. Fig. 8.11 shows an example frame with visualizations of the top view, grid space and resulting environment model.

8.8.2 Basic Performance

Fig. 8.12 highlights the grid-based feature fusion. A left turn is chosen since because of the narrow camera field of view this is one of the most challenging tasks: Only the right lane boundary is visible. Without temporal fusion it is hard to decide whether this is the right or left boundary of the road. In the shown part of the grid space the patch votes placed by the detected features form a bimodality with large longitudinal variance. Due to the previously detected road patches

[2]Video: `url.fzi.de/aadc2016`

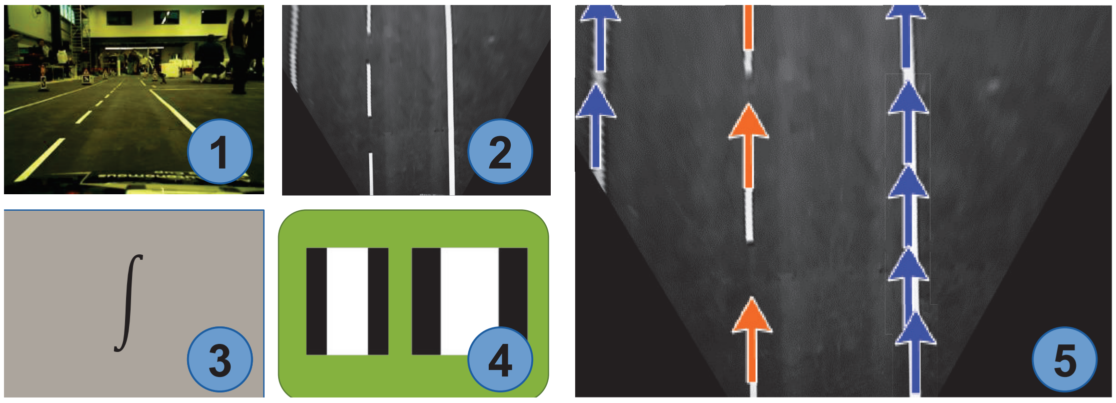


Figure 8.10: The line feature extraction strategy. 1: The input image. 2: Inverse Perspective Mapping (IPM). 3: Integral images are calculated from the IPM. 4: Haar-Features as applied to the image, for a thin center line and a thicker side line, 5: Examples of the found side line (blue) and center line (red) features displayed on top of the IPM. [8]

the ambiguity can be solved. This is a basic capability that is needed many times during a drive on a common track.

8.8.3 Advanced Challenging Situations

In the final competition several new challenging situations were given that require a robust road layout perception. In the already mentioned video some of these challenges are shown:

Besides the glare light and the sensor failure due to snow, that were only solvable by an action of the performance assessment module, there is also the tunnel and missing lane markings. At the tunnel, a correct estimation of the road is possible even though there are only a little number of detected features. The temporal fusion is able to track the previously estimated road over time and extend it using the little amount of new features. Similarly missing lane markings on intersections do not affect the performance. At the pedestrian crossing the white big zebra bars are not detected as lane markings and do not reduce the perception performance.

8.9 Conclusion

In this chapter it is shown how an advanced method for road layout estimation can be modeled using the proposed language.

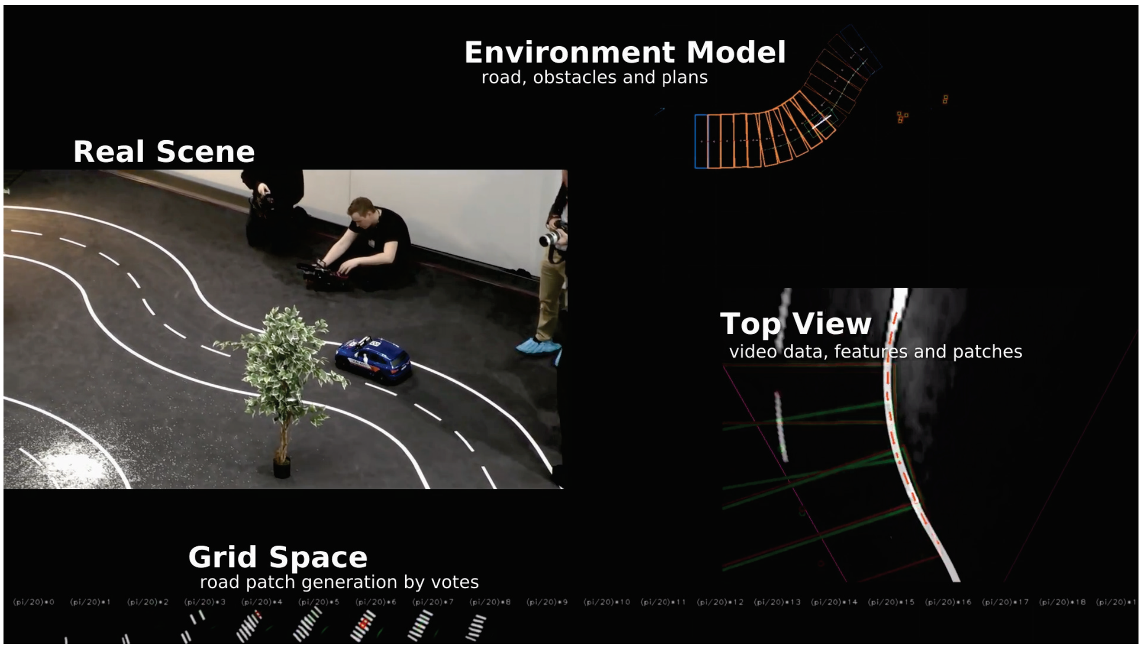

Environment Model
road, obstacles and plans
Real Scene
Top View
video data, features and patches
Grid Space
road patch generation by votes

Figure 8.11: Exemplary frame of the video. The video data recorded by the vehicle and used for perception is displayed in the lower right. Inverse perspective transform is already applied. Furthermore lane features (red lines) and patches (red and green boxes) are displayed. In the upper right the local map of the current state is shown. Road patches (red boxes) as well as the ego vehicles position (white arrow and green box) and the planned trajectory (white line strip) are displayed. In the bottom a part of the grid space is shown: Several x-y-grids of different orientations for the road patches. An excerpt of the grid is discussed in detail in Fig. 8.12. [8]

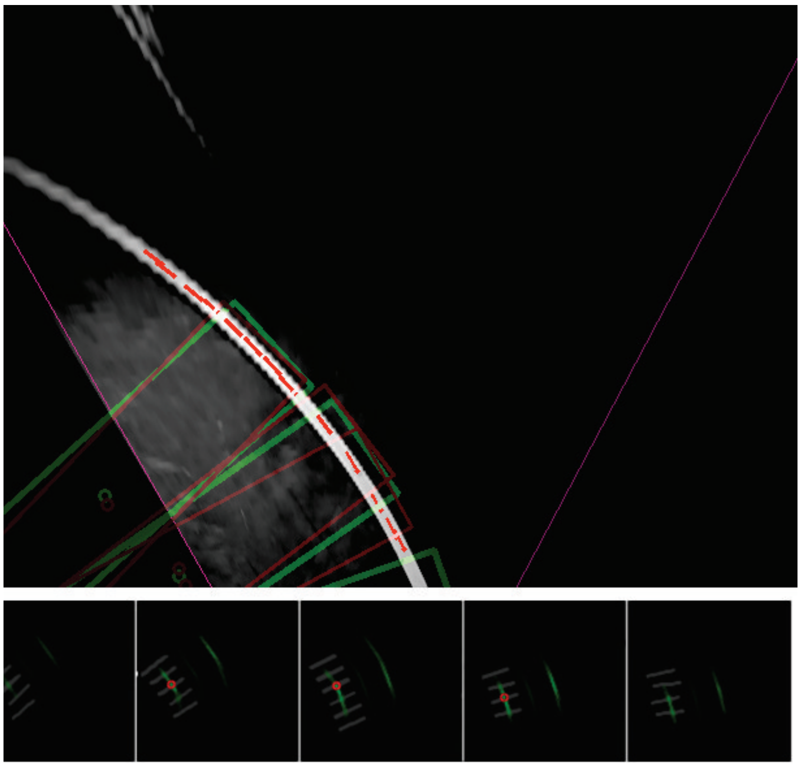


Figure 8.12: Road detection on a curved road. The upper image shows the top-view generated from the input image. Only the right outer marking can be seen. These markings are detected using the Haar-Line filters which generate features (red lines on the lane marking). The lower image shows a portion of the voting space for straight patches ($\tau_p = \text{road}$). Gray votes are generated by previously found patches, green votes are generated by the line features. The maxima which are found when sampling from the grid space are marked by small red circles. Note that the patch votes locate new patch hypotheses mainly in the longitudinal position while feature votes locate them in the lateral position. Patches are shown as green boxes in the top-view. Even though only a very small portion of the road is visible, the algorithm correctly approximates the curve. [8]

8.9.1 Summary

The key points are:

- Compared to the related work the proposed road perception method is new.
- From a real world oriented application description, the application ontology can be derived, based on the generic DSL for the traffic domain.
- Spatial constraints between real world entities are described as dependencies between attributes.
- Message representations can be chosen locally in the overall model.
- The local message representations and different inference methods motivate a modular implementation of the coherently described model.
- The resulting implementation was integrated in a complete automated driving system and performed well at the Audi Autonomous Driving Cup showing the proof of the concept.

8.9.2 Used Language Properties

The key goal of presenting this application within this thesis is to exemplarily show the usage of properties (Chapter 4) of the proposed language.

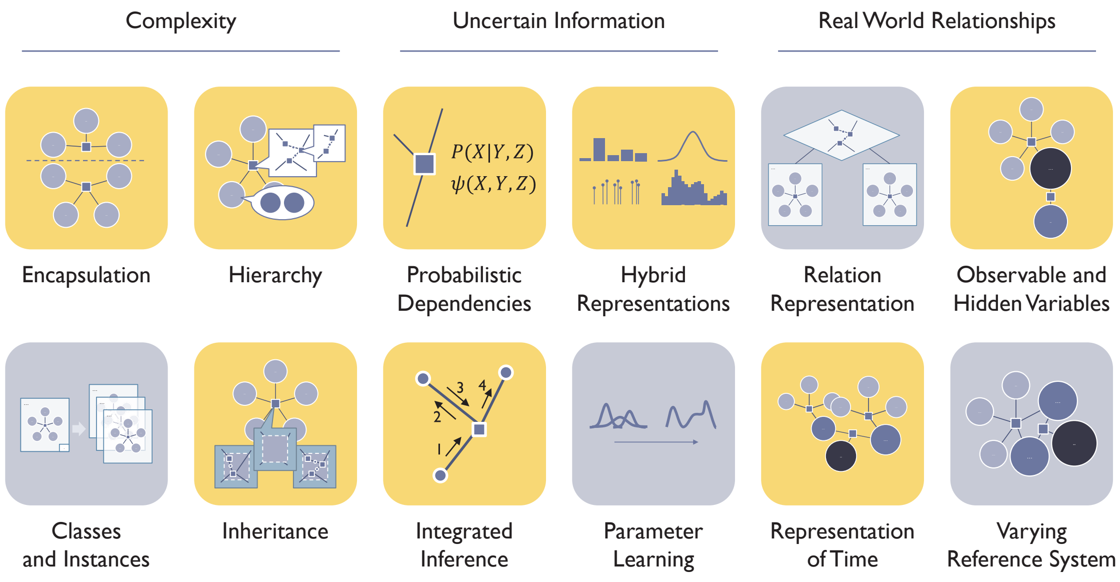


Figure 8.13: Some of the language properties are highlighted that are beneficial to the road layout estimation application.

In this application the following properties (Fig. 8.13) are beneficial:

- **Encapsulation** (Sec. 4.1.1):
  Since the overall model is splittable at class and dependency borders the implementation can be separated into distinct independent modules such as pose estimation, visual feature extraction and grid-based fusion (Sec. 8.7).
- **Hierarchy** (Sec. 4.1.2):
  The hierarchical modeling helps defining the overall principle (Sec. 8.4) and inference schedules (Sec. 8.6) on a higher level. The spatial and temporal dependencies are defined in separate classes describing the factors (Sec. 8.5).
- **Inheritance** (Sec. 4.1.4):
  The inheritance property is beneficial at several places. Especially the spatial constraints can be defined in very different ways (Sec. 8.5) but used as generic dependency in the overall estimation (Sec. 8.6).
- **Probabilistic Dependencies** (Sec. 4.2.1):
  A key component of this application is to define the dependencies probabilistically to incorporate uncertainties. Complex probability distributions (Sec. 8.5) can be incorporated using the grid representation (Sec. 8.6).
- **Hybrid Probability Distribution Representations** (Sec. 4.2.2):
  Different attributes of the model are represented differently. The language allows easy combination and conversion between them. Especially the two representations of the `RoadSegments` class (Sec. 8.6) are incorporated within the same class and accessable from factors.
- **Integrated Inference** (Sec. 4.2.3):
  The language's capability of describing inference schedules on a high level allows focusing on relevant message passing directions. This gives support to several designing decisions, from message representations to module separation (Sec. 8.6).
- **Observable and Hidden Variables** (Sec. 4.3.2):
  The application has several observable variables: The ego vehicle's odometry and many observed object measurements, such as different lane markings and traffic signs. The actual road layout is estimated from these features (Sec. 8.3).
- **Representation of Time** (Sec. 4.3.3):
  A key functionality of this application is the temporal fusion to overcome sensor noise and difficultly observable situations (Sec. 8.8.3). The temporal aspects of the modeling language are used in two areas: At the vehicle pose estimation and at the road layout estimation (Sec. 8.4).

The application in Chapter 9 also profits from the remaining properties, especially the machine learning capabilities and the massive usage of relations.

9 Route, Behavior and Trajectory Estimation

This final application chapter handles route, behavior and trajectory estimation of traffic participants, especially for future time where no evidences by observations have been gathered. The result will be a traffic participant prediction. Since the inclusion of ego vehicle localization and feature extraction from sensor data have already been shown in Chapter 8, focus is on the estimation and prediction using given object measurements and how to utilize the OOFGML there. On this level of abstraction, the approach is not only applicable for a single automated vehicle observing the environment but also for multiple vehicles or intelligent infrastructure [2].

The goal of this chapter is to outline how the features of the OOFGML can be utilized to achieve the application: It is shown that all relevant aspects are describable in a consistent way. Nevertheless also the performance of the resulting application is evaluated by comparing it to baseline methods, e.g. prediction by basic Kalman filter motion models.

Two different approaches for behavior estimation are followed: A single-shot approach and a temporal model approach. Starting with an initial view on the related work (Sec. 9.1), followed by the general application goal (Sec. 9.2) and ontology definition (Sec. 9.3), applicable for both approaches. Then the OOFGML-aided specialization for the single-shot approach is handled in detail (Sec. 9.4) before the temporal model approach is focused in same detail (Sec. 9.5). In Sec. 9.6 several aspects of the traffic participant prediction problem are evaluated.

This whole chapter is largely based on the previous publications [6] and [9]. Some more details considering the applications can be found there.

9.1 Related Work

Prediction of traffic participant's behaviors is an important research topic with various facets. While basic prediction with little model knowledge is already applied inside simple filters for years, more long-term and semantic predictions are the focus of research in recent years. To drive fully autonomously in urban traffic a precise long-term prediction is essential.

Precise prediction can be achieved by directly utilizing sensor data [26], [132]. These are dependent on the actual sensor. One step to more generality is to handle abstract data but to learn for a specific environment like one specific intersection [133], [91].

To reach more generality, also the infrastructure can be considered explicitly. Then the influence of the road geometry can restrict the traffic participant's future trajectory probability distribution, e.g. at the level of lanes [27] or corridors [5]. The infrastructure information can also include information about traffic lights and right-of-way. Often this information is loaded from a predefined map but there are also approaches that estimate the road layout, even of complex intersections, online [53]. All infrastructure information is prone to uncertainties, either because the map is not precise or up-to-date or because it is estimated by uncertain observations from perceiving sensors (Chapter 8).

Additionally, there are uncertainties in the traffic participant's behavior which cannot be modeled precisely by human experts. Promising approaches use machine learning to parametrize general models, e.g. by Case Based Reasoning [59] or by learning generic behavior models from unlabeled observations in a Dynamic Bayesian Network [58]. This way, street-dependent behaviors like braking in front of turns or intersections and object-dependent behaviors influenced by interactions between vehicles like braking because of a slower vehicle are considered when estimating predictions.

To handle all uncertainties in a generic model, probabilistic methods like Bayesian Networks [105] [71] [106] or Dempster Shafer Theory [103] [128] are applied. Especially generic state space models (Sec. 2.6.1) tackle the prediction problem perfectly with their sequential estimation character and are already used in many works [57] [136] [82] [58]. Usually they do not consider the object-oriented character of the real world explicitly and assume simplifications like a predefined map without uncertainties.

In other works, object-orientation and relations (FOPL, Sec. 2.7) have been the focus to attain a higher level of scene understanding. Schamm et al. [115] showed the potential of applying these methods to traffic situations. The proposed model derives condensed information (like a time-to-collision) from uncertain preprocessed sensor data. The result is usable for advanced decisions, e.g. risk assessment.

In the application in this chapter it is shown how the OOFGML can handle the different known challenges in one common description language and how vehicle prediction can profit from this.

9.2 Application Goal

The goal of this application is to estimate a consistent scene understanding of the ego vehicle's surrounding. This understanding shall include semantic information about the traffic participants' inner states such as their current behavior, planned route and future trajectories. In traffic situations the interaction between different traffic participants has a large impact on the actual individual behavior. These have to be taken into account to allow a precise scene understanding (Fig. 9.1).

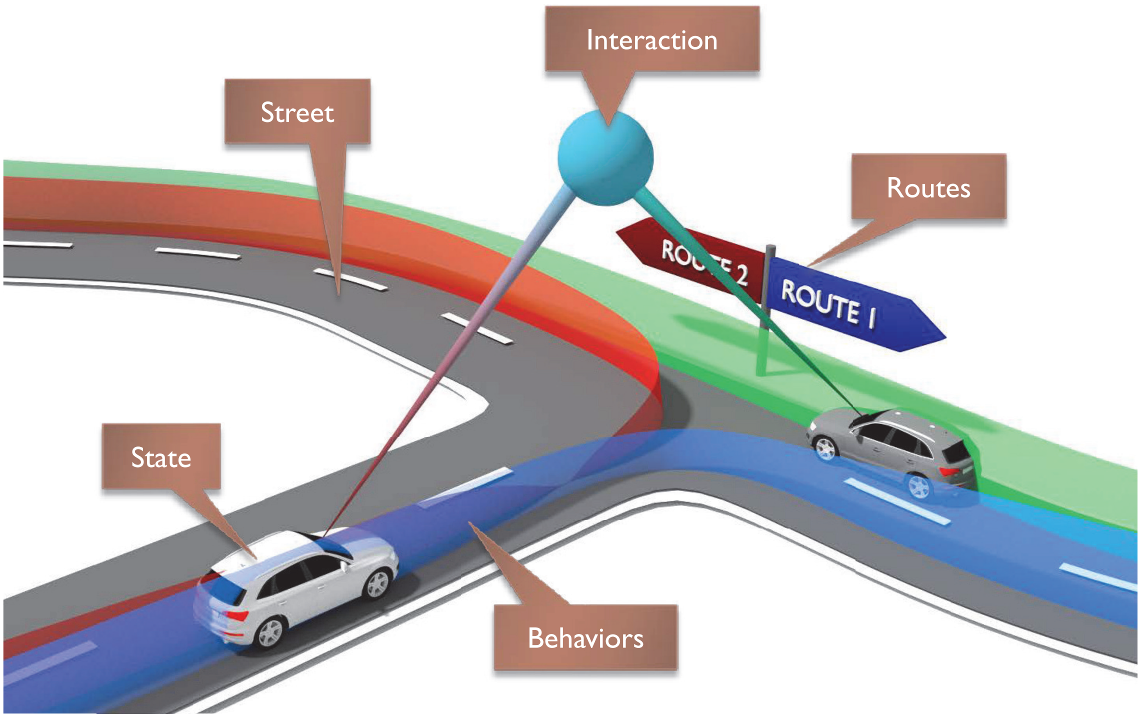


Figure 9.1: Interactions have a large impact on vehicle prediction: The constellation of the left and the right vehicle induces a braking behavior for the left vehicle if it turns left, because then their routes intersect and they have to react to each other (red velocity profile). Observing a braking behavior of the left vehicle can be used to predict a left-turn of this vehicle. [9]

The basic idea starts with the separation of discrete and continuous attributes (Fig. 9.2): While routes and behaviors can be modeled as a discrete set, there is an infinite number of possible trajectories representing a specific behavior. The trajectories can be estimated by combining several states over time that are measured using noisy measurements. A basic Bayesian model can be constructed to represent these dependencies (Fig. 9.3).

Such a Bayesian model can be used to estimate a coherent understanding of an observed traffic scene, including (per traffic participant) a route and behavior es-

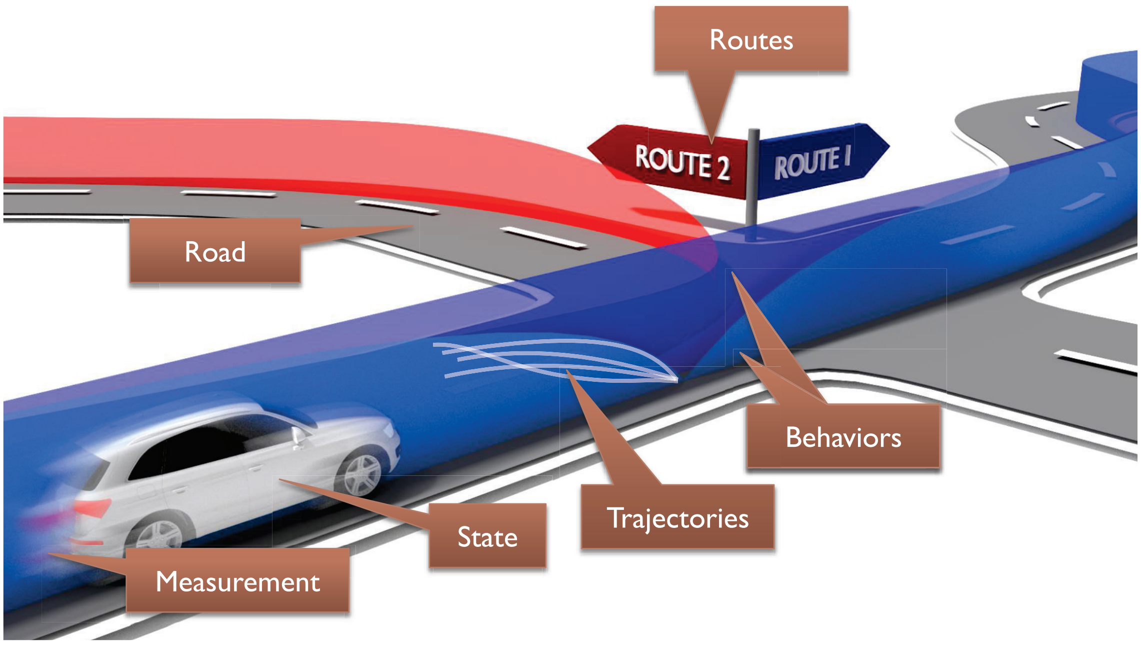


Figure 9.2: A more detailed view on the relevant elements: Different behaviors can be possible on a single route. Each behavior can be realized by an infinite number of possible trajectories, which are visualized here as velocity profiles over the track of the vehicle. Adapted from [6].

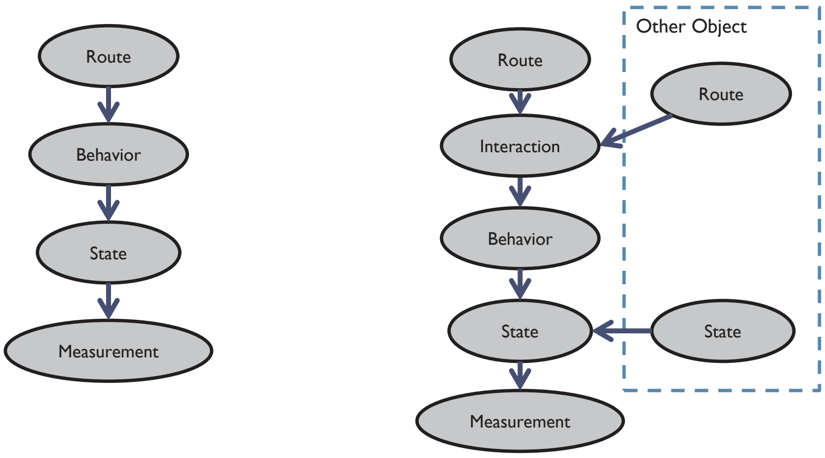


Figure 9.3: The basic dependencies in traffic scenes can be desribed by a Bayesian model. The route induces possible behaviors which result in possible vehicle states that can be measured (left). Also interactions between vehicles can be integrated into a basic model (right). Adapted from [9].

timation, a trajectory prediction as well as a more precise current state estimation. Having observed many vehicle behaviors the model can also be used to learn difficult to parametrize dependencies such as the actual form of the trajectory given a specific behavior, e.g. braking early or later in front of an intersection.

The idea for the temporal aspect of the application is inspired by the concepts of [56] summarized in Fig. 9.4. The past state of the dynamic object and the state of its environment (other dynamic objects, road layout) are combined to a context-aware state of the dynamic object. This holds all information to derive a probability distribution over possible actions which will lead to the new state of the dynamic object. Since the state can be observed from measurements, these observations give evidence on what actions have been taken and what part of the constellation leads to this decision. This is the principle how behaviors and interactions are estimated from traffic participant state observations.

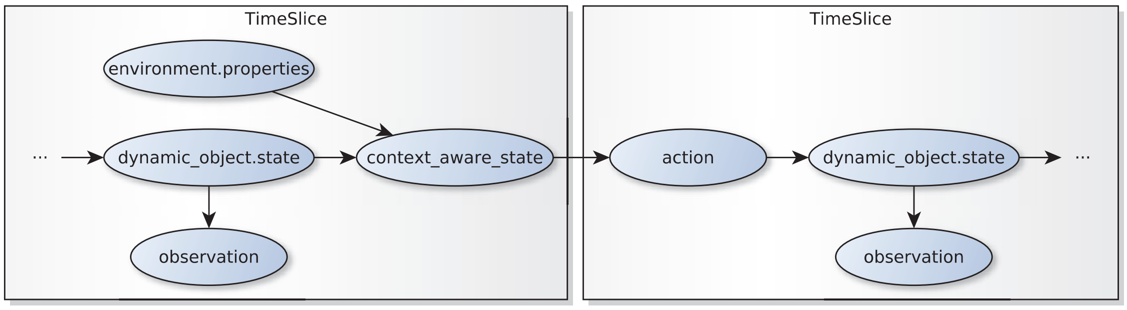


Figure 9.4: Bayesian temporal estimation principle inspired by Gindele [56]. The object states are combined to context-aware states from which actions are derived that result in a new object state. Object states can be observed by noisy sensor measurements.

9.3 Application Specific Ontology

From the general DSL for the traffic domain (Chapter 5) an application specific ontology has to be derived to represent the conditions of the application (Sec. 9.2). From the entity and relation class structure of the DSL (Fig. 5.3) appropriate entities and relations have to be chosen to be included in the application specific ontology (Fig. 9.5).

For this application several entities and relations have to be included: The `AutomatedVehicle` and the `RoadLayout` correspond to the real world entities. The `RoadLayout` includes the road hierarchy consisting of `LaneSegments`. Lane Features do not have to be considered if the road layout is loaded from prerecorded map data. The focus of this application lies on the details of the `Dynamic Object` class. There, several relations describe the relationship between the `Route` and `Behavior` of different dynamic objects and their environment.

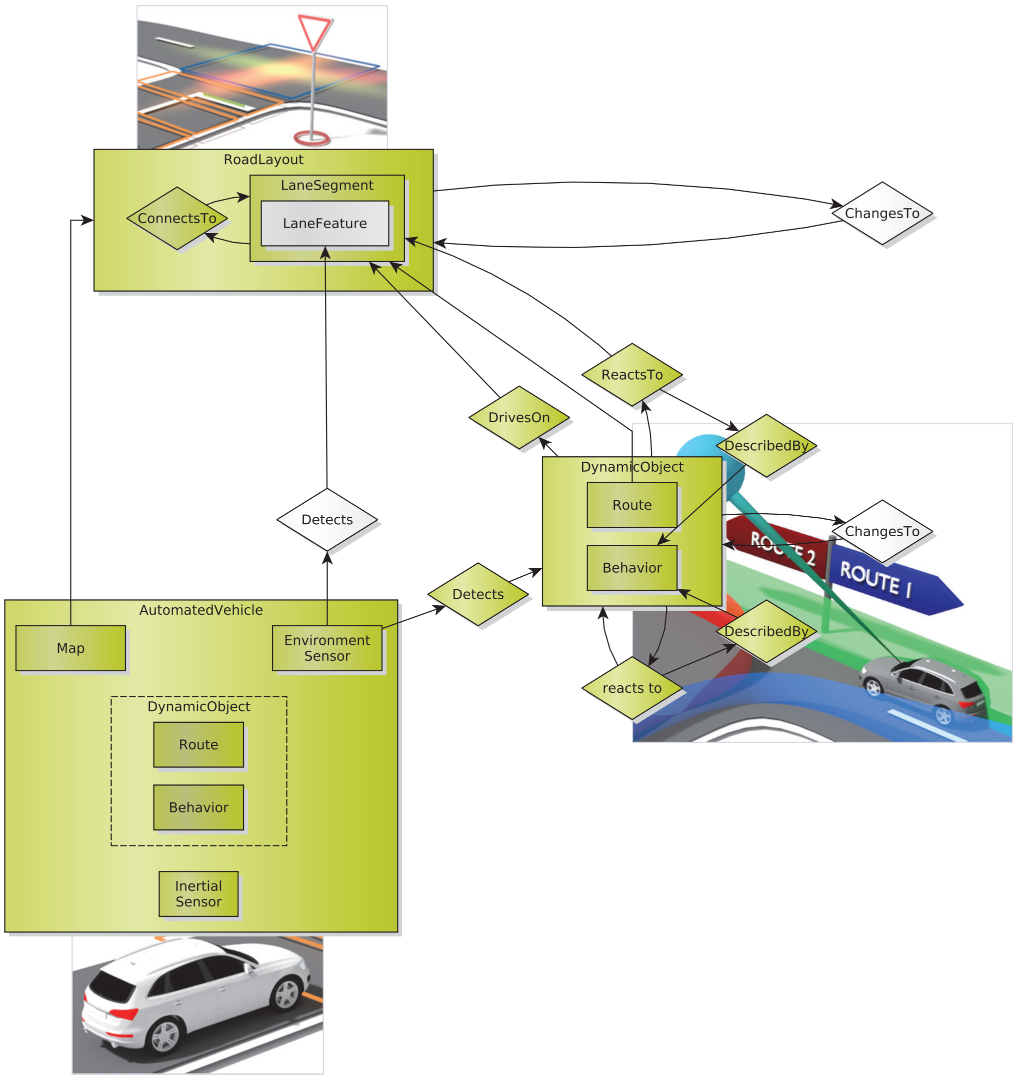


Figure 9.5: The necessary entities and relations comprise the ego vehicle, the road layout and the dynamic objects with a focus on routes and behaviors and the relations between these. Note that `reactsTo` is modeled as a relation but is included in the relation `decribedBy`. This is possible by replacing the `reactsTo` relation by an entity. The relation is kept because of simplicity.

9.3.1 Routes, Behaviors and Interactions

To understand the relevant relations, they are discussed seperately for infrastructure (lane segments) related behaviors and dynamic object related behaviors. Fig. 9.6 shows a more detailed view with explicitly depicted `has` relations for a single object and the two behavior sub classes. Reconcile that the goal is to estimate what dynamic object reacts to what scene object. The probabilities of the relationships are unknown, so each object can possibly react to any object. Thus, to start estimating, any possibly probable relationship has to be modeled.

In Fig. 9.6a the `reactsTo` relation describes that a `DynamicObject` reacts to a `LaneSegment`. If the existence probability of this relation is high, there has to be a behavior with high probability that matches this reaction. The `describedBy` relation connects the `reactsTo` relation to all behaviors that (would) realize this reaction. The `has` relation of the `DynamicObject` connecting the `Behavior` includes the existence probability that can be estimated by observations. The existence probability of the reaction is then a sum over all existence probabilities of behaviors describing this reaction. Simultaneously an existence probability of routes can be estimated as a sum over all reactions that are related to lane segments in the route. Additionally, to define the behaviors, the lane segment properties have to be considered, which can be accessed by the relation chain through the `reactsTo` relation.

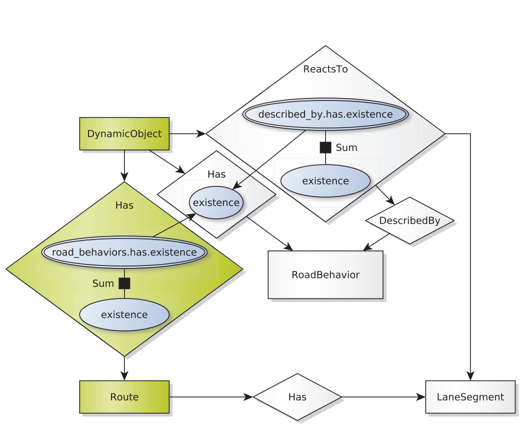
The behaviors related to dynamic objects are modeled similarly (Fig. 9.6b). There, the `reactsTo` relation relates to another `DynamicObject` (instead of a `Lane Segment`) but is again described by one or many behaviors. The properties of the behavior are dependent on the relative position of the dynamic objects and the relation between the routes they are driving on. The ontology structure allows again the summation for route probability and reaction probability.

In Fig. 9.6c it is shown how the existence probability of vehicles on routes is actually derived from both, the `InteractionBehaviors` and the `RoadBehaviors`.

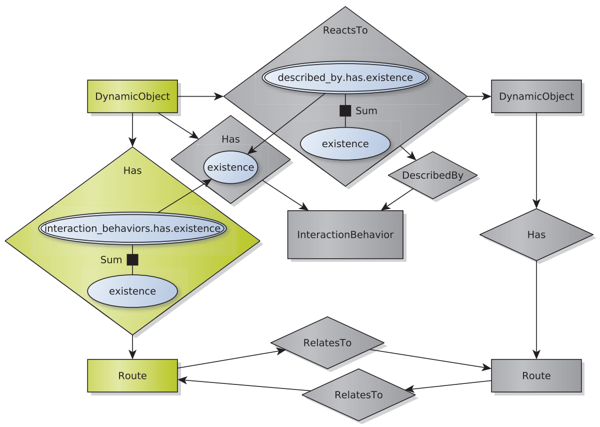
9.3.2 Relative Attributes

The relations among dynamic objects, between dynamic objects and road elements and among road elements allow to integrate attributes that are related to both involved entities (subject and object). Examples are given in Fig. 9.7.

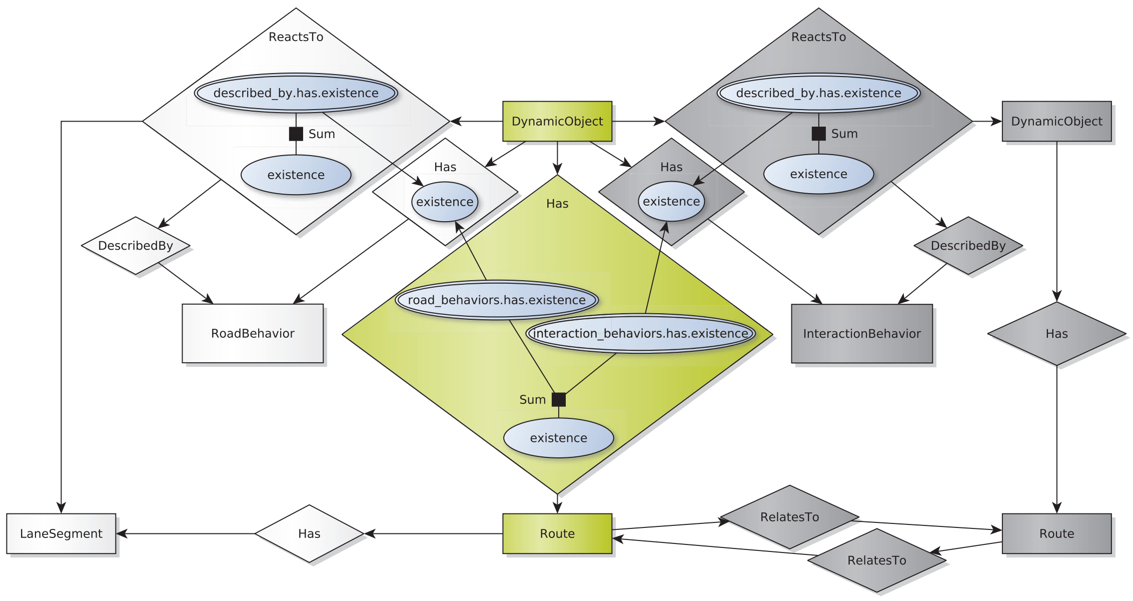
The relative attributes can be offered in sub class definitions of the relations but will be used depending on the application and the desired precision.



(a) Sub class `RoadBehavior` for reactions to `LaneSegments`.



(b) Sub class `InteractionBehavior` for reactions to `DynamicObjects`.



(c) Both sub classes combined.

Figure 9.6: The two behavior sub classes for reactions to infrastructure (`Lane Segment`) and reactions to other traffic participants (`DynamicObject`). The overall existence probability of a relation that a `Dynamic Object Has` a given `Route` is calculated by all reactions to dynamic objects and infrastructure on this route.

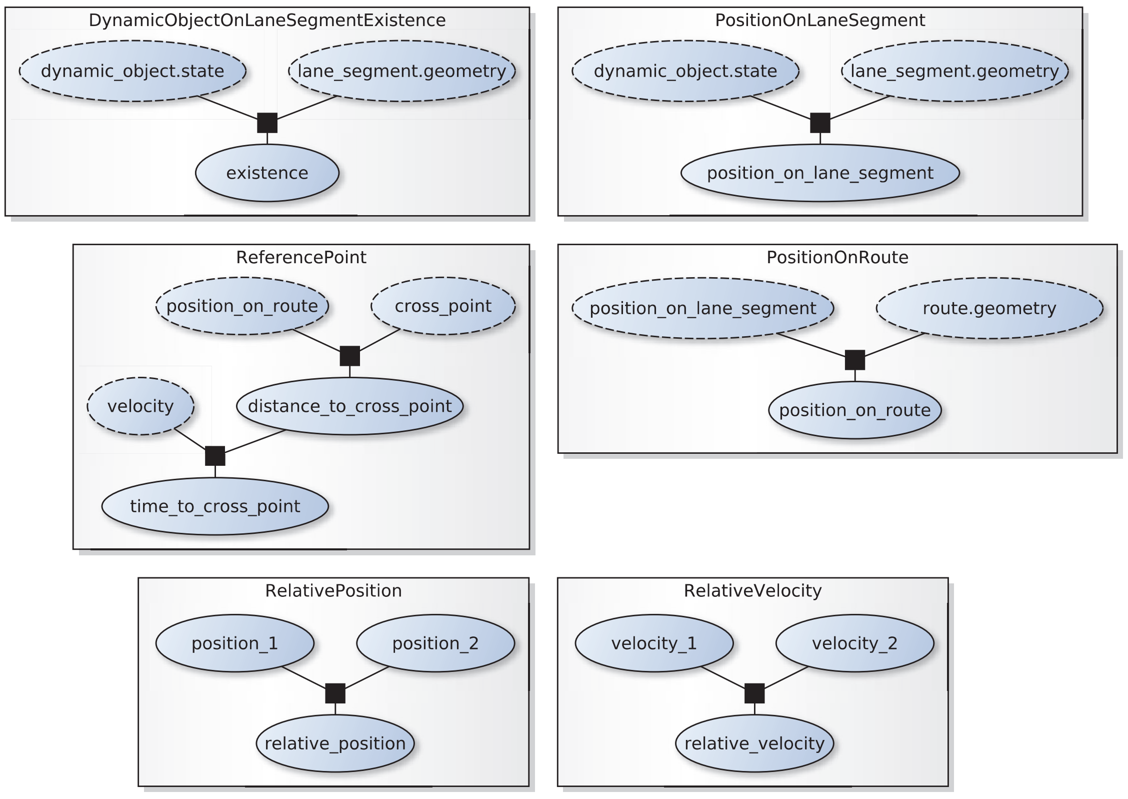


Figure 9.7: Examples of relative attributes depending on the combination of two entities.

9.3.3 Gating for Complexity Reduction

The ontology only describes the relations between entities in the application. The actual dependencies of class attributes can be modeled very differently, depending on the desired precision or focus.

For example, because of all the possible behaviors, routes and interactions a precise estimation considering any possible constellation would have an exploding complexity: For a situation with n_O dynamic objects and n_R routes (Fig. 9.8) there is a possibility for $n_O * n_R * n_R * (n_O - 1)$ interactions. An object will not interact with itself ($\Rightarrow n_O - 1$) but objects can interact on the same route, e.g. when following each other.

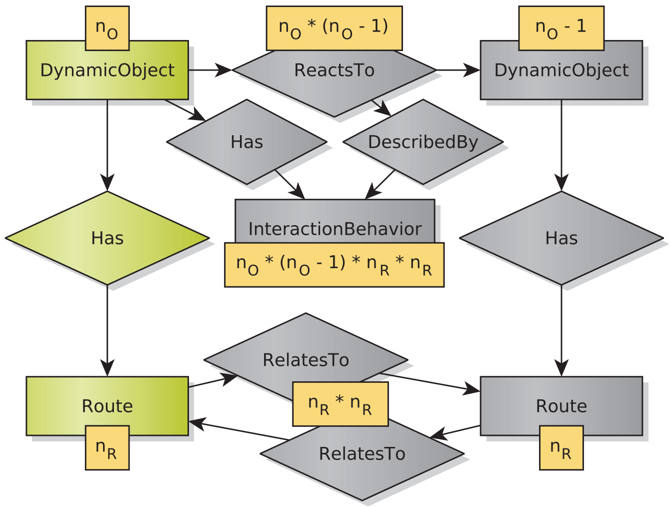


Figure 9.8: If all possible behaviors are instantiated a scene with n_O dynamic objects and n_R routes leads to $n_O * n_R * n_R * (n_O - 1)$ interactions.

An exemplary 3-way bidirectional intersection (no one-way streets) has 6 different routes. A situation with 4 vehicles leads to $4*6*6*3 = 432$ interactions (1728 interactions on a 4-way intersection). The higher the number of routes and the higher the number of vehicles, the more this becomes a crucial problem, while the majority of interactions has a very low probability of becoming probable. If the precise calculation of the interaction probability is expensive it is obvious that this will lead to a high computational overload.

An efficient inference algorithm detects the configurations that tend to a low joint probability early, and tries to avoid them. The technique of Gating known from multi target tracking (Sec. 2.6.6) can be applied here (Fig. 9.9).

A map matching algorithm can give a prior on the route existence by comparing the vehicles' lateral distance and orientation difference to the involved lane segment geometry [5]. This is a simple implementation of the `DynamicObject-drivesOn-LaneSegment` and `DynamicObject-has-Route` relations. Instantiating this for all vehicle-route combinations allows estimating a basic probabil-

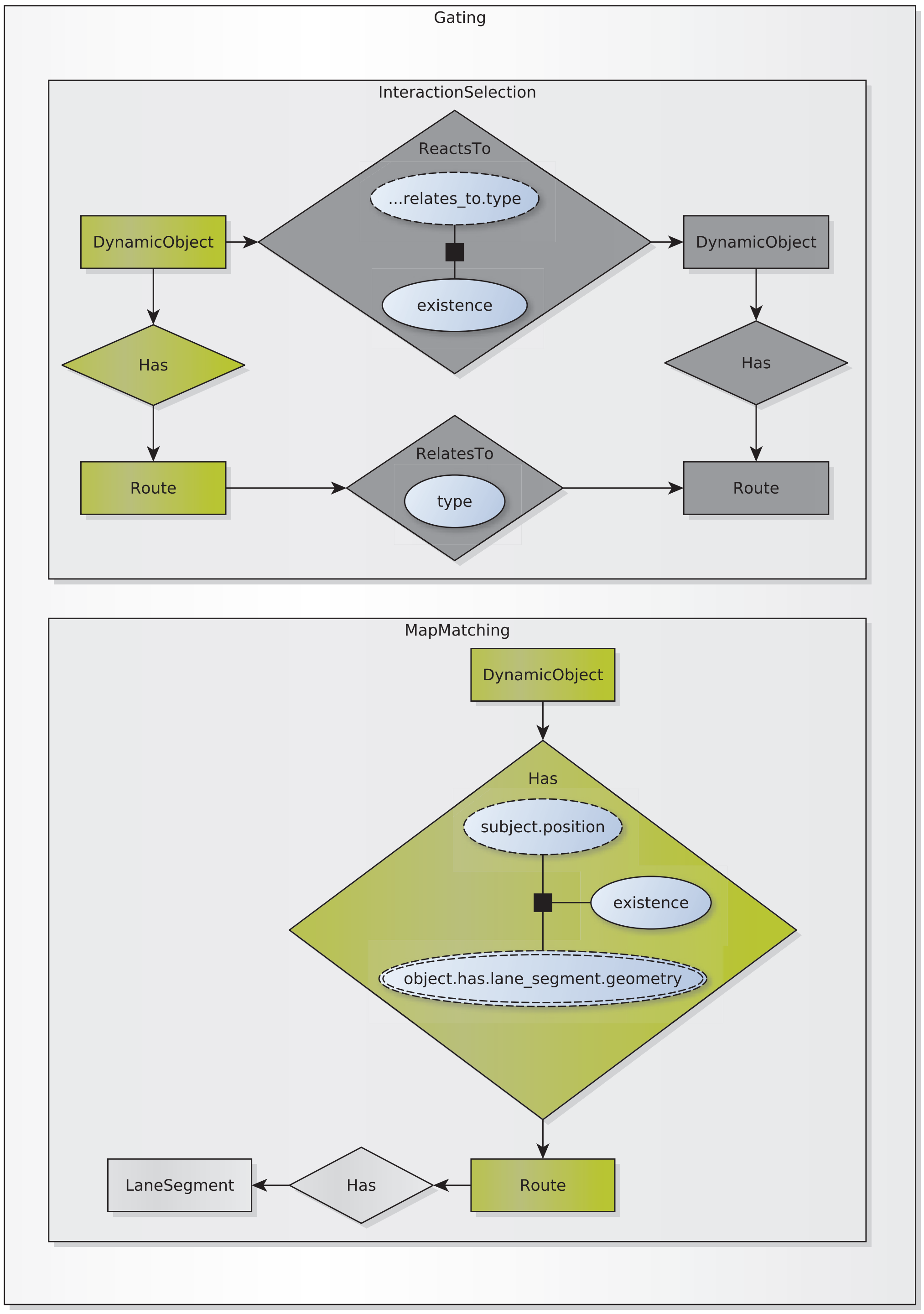


Figure 9.9: Sub models for gating: Map matching and interaction selection. Thresholds on the resulting `existence` probabilities allow discarding unlikely relations.

ity for all route existence probabilities. Using thresholds on these results allows discarding many unlikely hypotheses. In the case of the 3-way intersection the 6 possible routes usually reduce to 1 or 2. For actual interaction estimation only this subset of vehicle route relations has to be instantiated at all. Analogously from the relation types between routes (Fig. 9.10), a prior on the existence of interactions can be derived and thus the number of `ReactsTo` instances and induced `Behaviors` can be reduced.

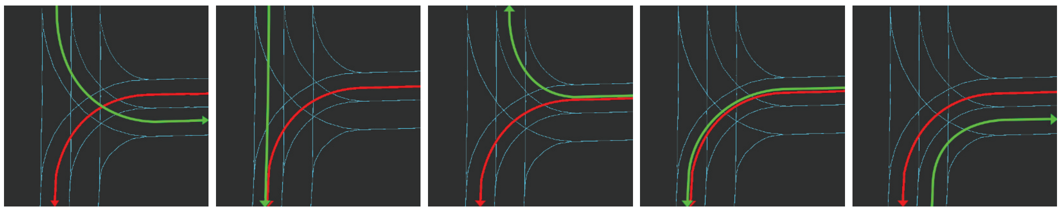
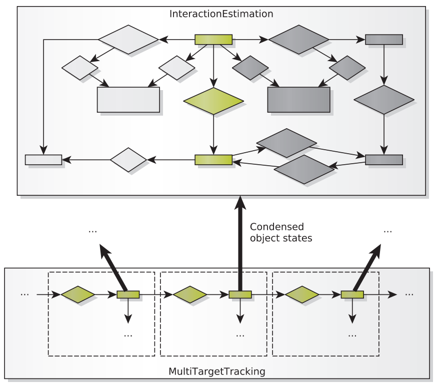

(a) Cross (b) Merge (c) Diverge (d) Follow (e) None

Figure 9.10: The five types of route relations *Cross*, *Merge*, *Diverge*, *Follow* and *None* are used to reduce instances of the `reactsTo` relation. The routes (green, red) can be derived from the road layout (cyan). [9]

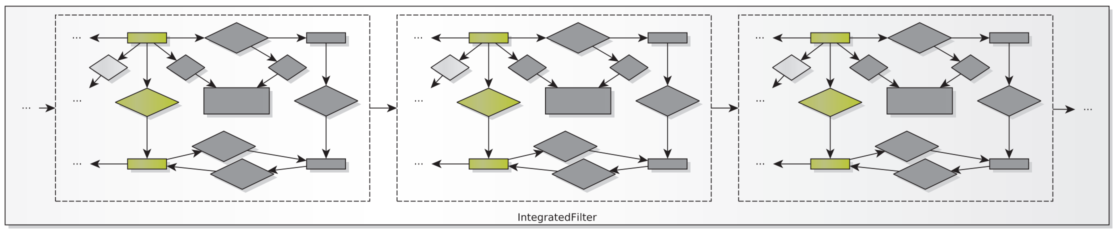
9.3.4 Single Shot and Integrated Temporal Model Approach

After having reduced the number of instances to instantiate, the actual precise interaction estimation can be solved in different ways. The main basic difference is if the temporal estimation is integrated into the interaction estimation or separated (Fig. 9.11). In the separated case, inference can be implemented in a modular way, separating the temporal tracking of object hypotheses from a static single shot evaluation of the compressed object states. In the integrated case, the temporal filtering is directly part of the interaction estimation. The interaction estimation is then handled in the temporal `changesTo` relation using environment related transition models which lets the state estimation also profit from the higher scene understanding.

The two approaches are discussed separately in Sec. 9.4 and Sec. 9.5. Each approach introduces attributes and sub classes to the basic ontology to fully define the estimation problem, approach and inference method.



(a) Single-shot approach: Interaction estimation is performed on a situation description that is estimated by a preceding multi target tracking.



(b) Integrated temporal approach: Interaction estimation is directly integrated into the temporal filtering process.

Figure 9.11: Principles of the *single shot* and *temporal* approach.

9.4 Single-Shot Approach

The goal of this approach is to estimate routes of traffic participants from single compressed object states. Since there is little temporal information and no temporal smoothing, the approach largely depends on a profound interpretation of the relations between objects and the corresponding road layout. An accurate parametrization of behavior models is crucial to this approach. They will be learned from observations.

9.4.1 Behavior Estimation

The basic assumption in this approach is that the object observations include attributes that depict the state of the object as well as attributes that correspond to the actions. From the object state possible reasonable actions can be inferred (depending on route and interaction hypotheses) that can then be compared to the observed action (Fig. 9.12). It is also assumed that these observations are only slowly changing over time. Thus the full state of one time slice (observed state and observed action) can be used instead of two time frames (comp. Fig. 9.4). This makes the interaction estimation independent of the temporal tracking and can behave in a single-shot manner.

The context-aware state can consist of several relative attributes that are derived from the relational structure (Sec. 9.3.2). For example for an interaction-related behavior it can be useful to consider the velocities of the interacting objects, their distance and time to a crossing point of their routes and the right of way at this crossing point (Fig. 9.12). The reasonable action can be expressed as the object's velocity or acceleration.

On the one hand the potential function of the factor connecting the context-aware state to the reasonable action is hard to parametrize by expert knowledge. On the other hand there are parts like for example the relative velocity between vehicles that can easily be described by a mathematical expression. Thus, it is desired to select explicitly which dependencies are parametrized by experts and which are learned from observations.

Subsequently the full class and attributes structure (Sec. 9.4.2) is built up before learning is focused (Sec. 9.4.3).

9.4.2 Classes and Attributes

For each entity and relation in Fig. 9.5 an OOFGML class and sub classes are defined with attributes and dependencies. An excerpt of the overall class structure is depicted in Fig. 9.13 focusing on the core functionality that is implemented

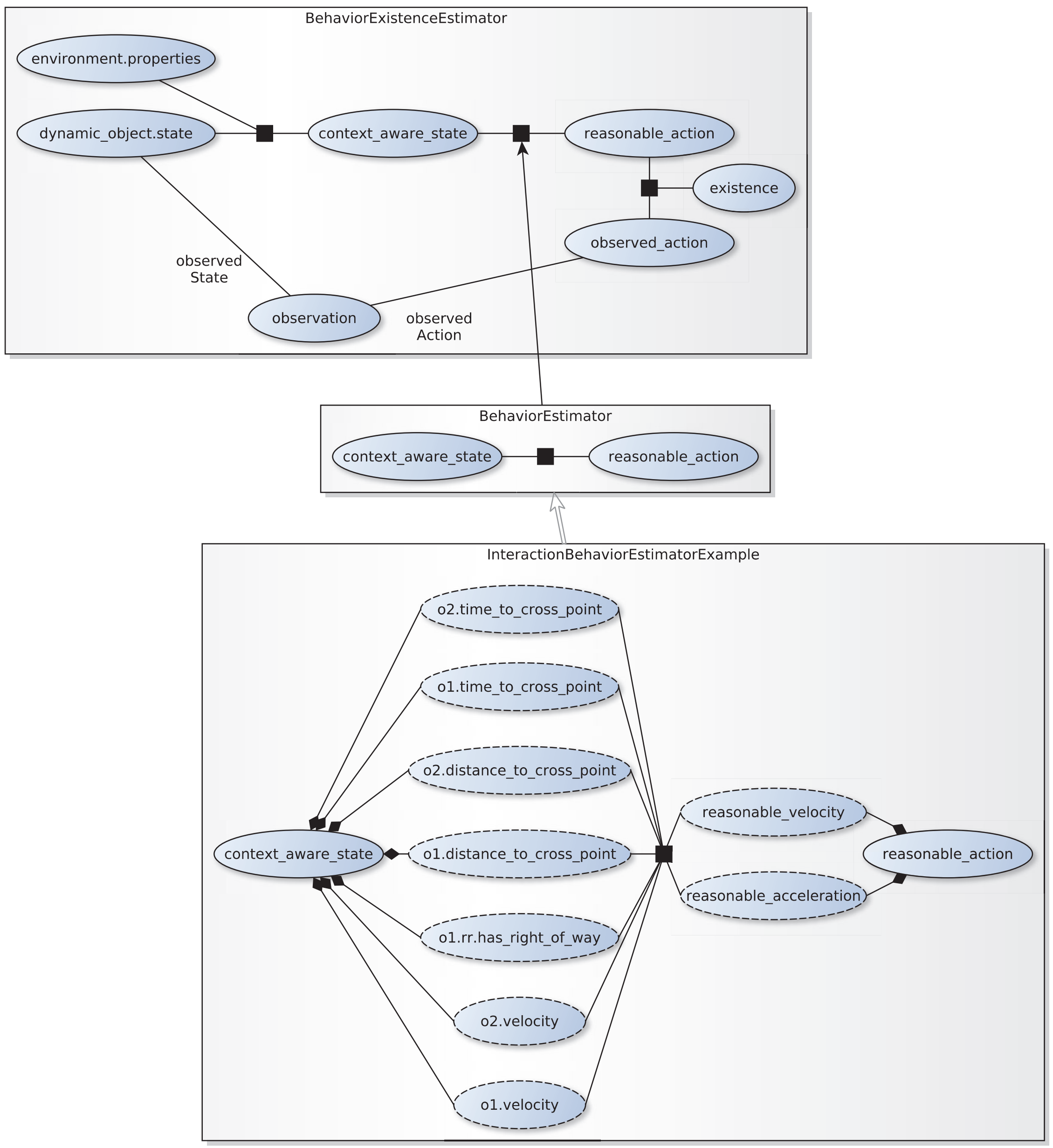


Figure 9.12: Core behavior estimation idea in the single-shot approach. Context-aware states are created from object state and environment, reasonable actions are derived and compared against observed actions from the same observations (top). The key component is the BehaviorEstimator which has a complex dependency model depending on many object and relation related attributes (bottom).

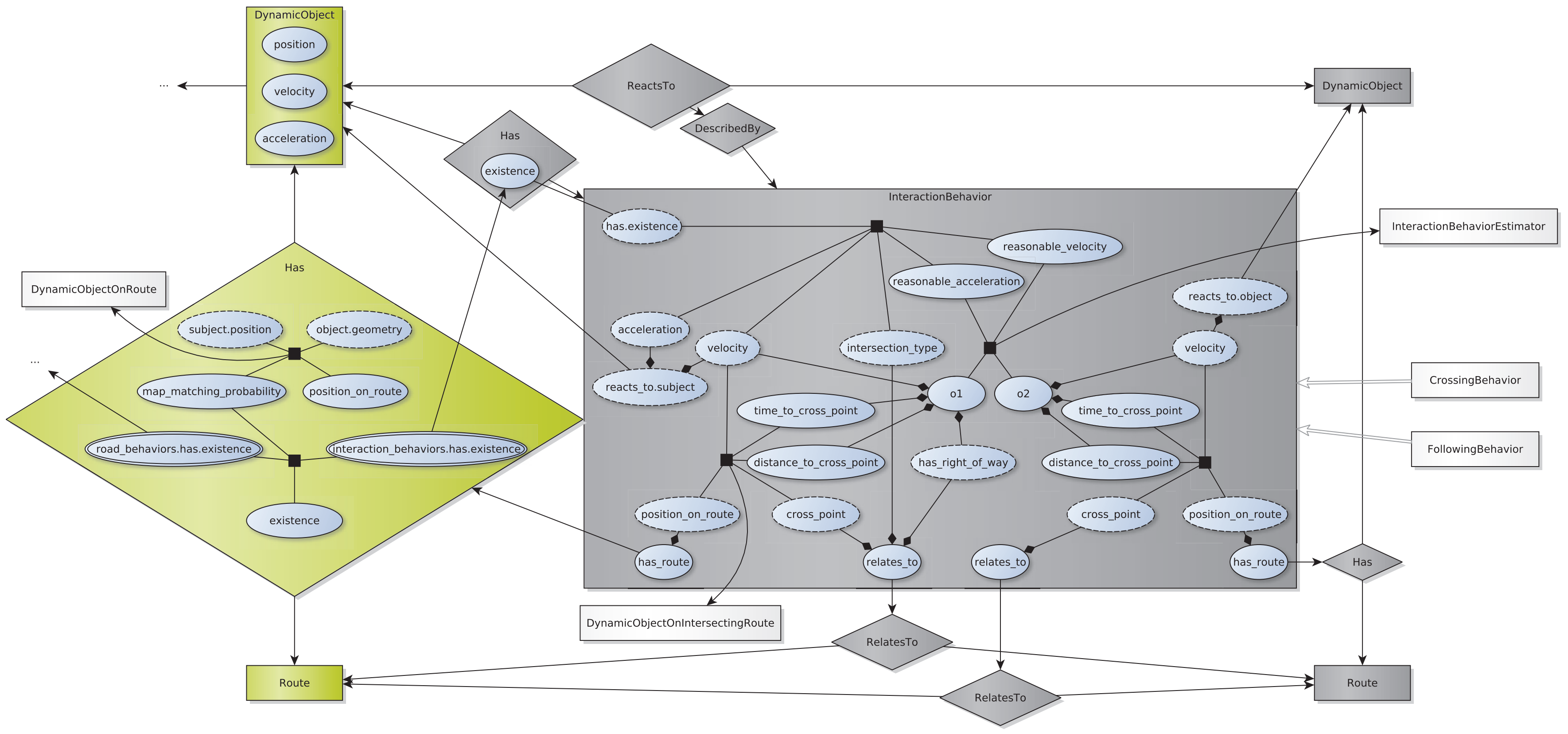


Figure 9.13: Core behavior estimation functionality around the `InteractionBehavior` class. Details of the `Interaction BehaviorEstimator` and sub classes are depicted in Fig. 9.14.

around the `InteractionBehavior` class. The resulting behavior existence probability is handled as described in Fig. 9.6 to derive route and interaction probabilities.

The core behavior estimation is handled in the factor that refers to the `InteractionBehaviorEstimator` class. Sub classes of the `InteractionBehavior` class (`CrossingBehavior` and `FollowingBehavior`) refer to sub classes of the `InteractionBehaviorEstimator` class. These are depicted in Fig. 9.14.

9.4.3 Learned Dependency Model

The behavior estimator base class infers reasonable actions from the context-aware state vector. Different discrete interaction behaviors are defined for following another object and for crossing the route of another object as sub classes (Fig. 9.14). This makes sense since following another vehicle means adapting the velocity to keep a desired distance to the other vehicle over a longer time while crossing another vehicle's route means adapting the velocity to not be at the intersection point at the same time. In both cases a reasonable velocity as well as a reasonable acceleration is estimated in two separate factors.

The dependency model for the following behavior is inspired by algorithms for adaptive cruise control, e.g. the reasonable acceleration is calculated from the position and velocity difference between the objects. The factors for inferring the difference are defined by models described in Sec. 9.3.2 while the actual reasonable acceleration and velocity are learned from observations.

The dependency model for the crossing behavior uses the distance and time to the intersection point and is then completely learned from observations.

To learn the model parameters, observations are recorded and automatically labeled with actually driven routes. Then every data set, consisting of a single time frame with given road layout, observed dynamic object state and ground-truth driven route is shown to the learning algorithm (in the sense of Fig. 4.8): Matching instances of the application-specific OOFGML model are instantiated, evidences applied and the parameters for the desired dependency models are iteratively learned via Expectation Maximization [36]. The parameters are hold fixed among multiple instances of the same class and stored in the class definition.

This way, a few discrete behavior models are learned from real traffic data and can then be instantiated multiple times according to the constellation of a new unknown traffic scene.

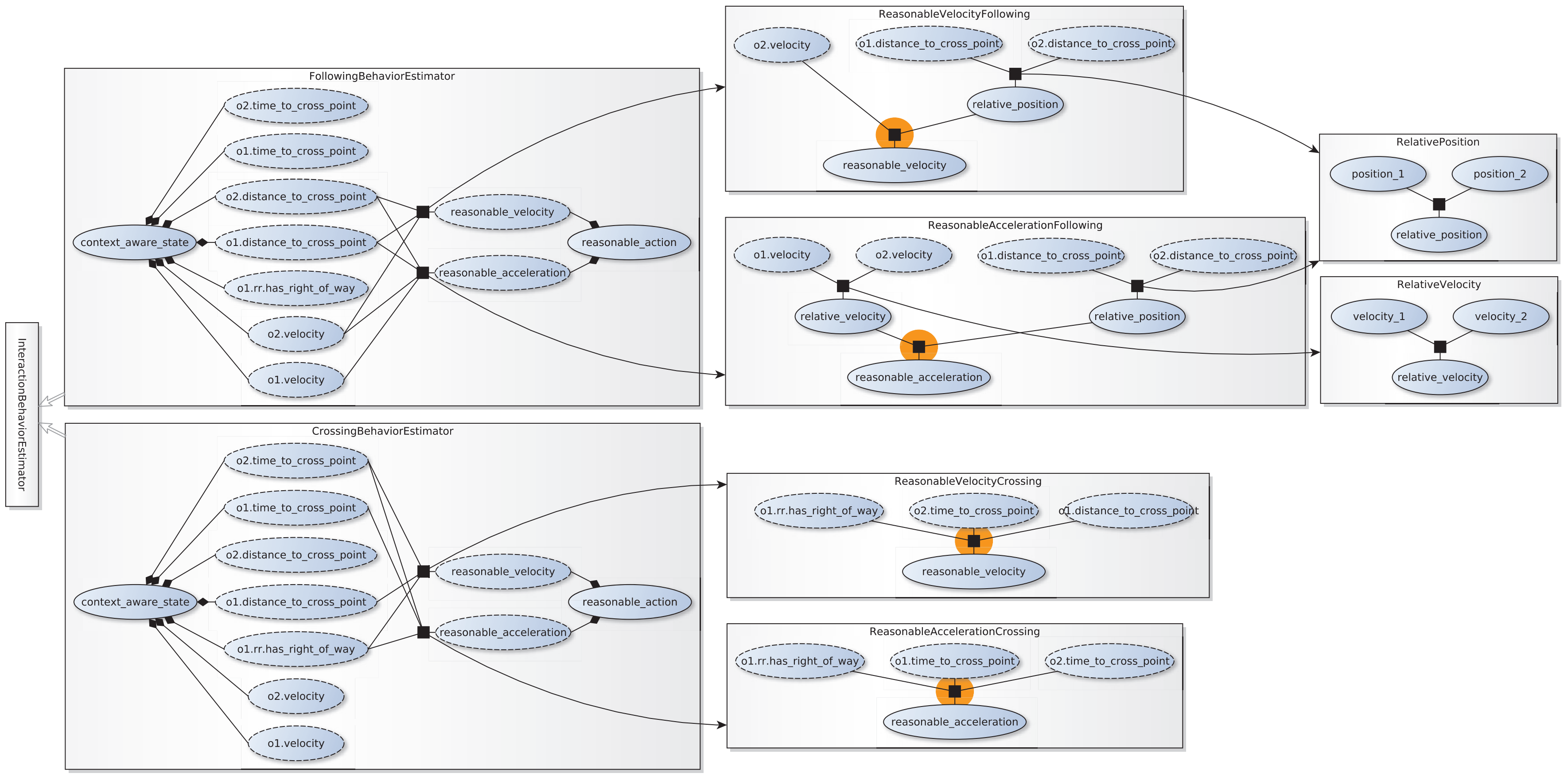


Figure 9.14: Details of the `InteractionBehaviorEstimator` class. The two sub classes for following and crossing behaviors model the derivation of reasonable velocity and acceleration differently. The factors which parameters are learned from observations are highlighted in orange.

9.4.4 Inference Methods

Inference in this application is split into three steps:

1. The dynamic objects in the ego vehicle's surrounding are temporally tracked by a multi target tracking (comp. Sec. 2.6.6).
2. A gating step including map matching and interaction selection reduces combinatorial complexity (comp. Sec. 9.3.3).
3. The actual inference of interactions and probable routes or learning of behavior model parameters is performed on a reduced subset of instances using the full detailed model (Sec. 9.4.2).

These steps can be modeled in a central class holding a set of processing scripts that instantiate the corresponding sub models (Fig. 9.15).

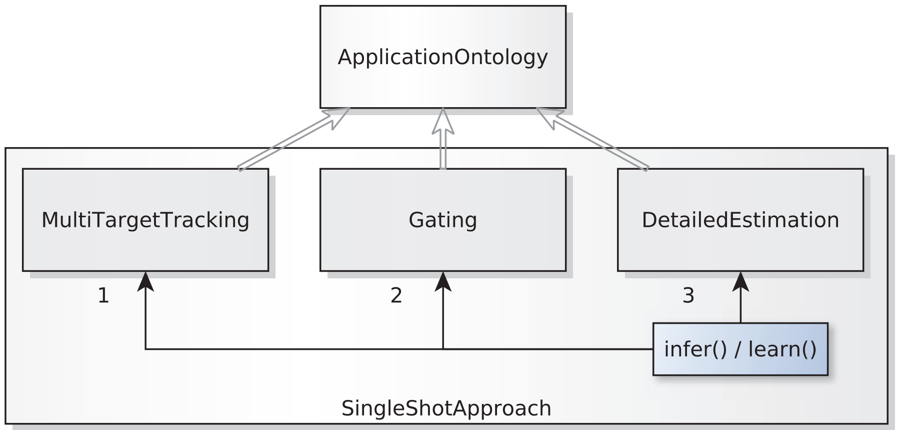


Figure 9.15: Inference and learning is split into three steps that are handled from a central class `SingleShotApproach`. The numbers describe the sequence of instantiation and inference. All estimation models inherit from the basic application ontology classes.

Together with the learning step the processing from sensor data to predicted routes is described using the OOFGML as schematically depicted in Fig. 9.16.

9.4.5 Implementation Opportunities

One of the five implementation opportunities described in Sec. 3.4 has to be chosen. Because of the three step inference (Sec. 9.4.4) it is obvious to choose a *modular implementation* similarly to the road layout estimation application (Chapter 8). For the multi target tracking an *established filter implementation* is chosen. The gating is solved in a *specialized implementation* specialized for the two tasks. Finally the actual interaction estimation is implemented as a *FOPL*.

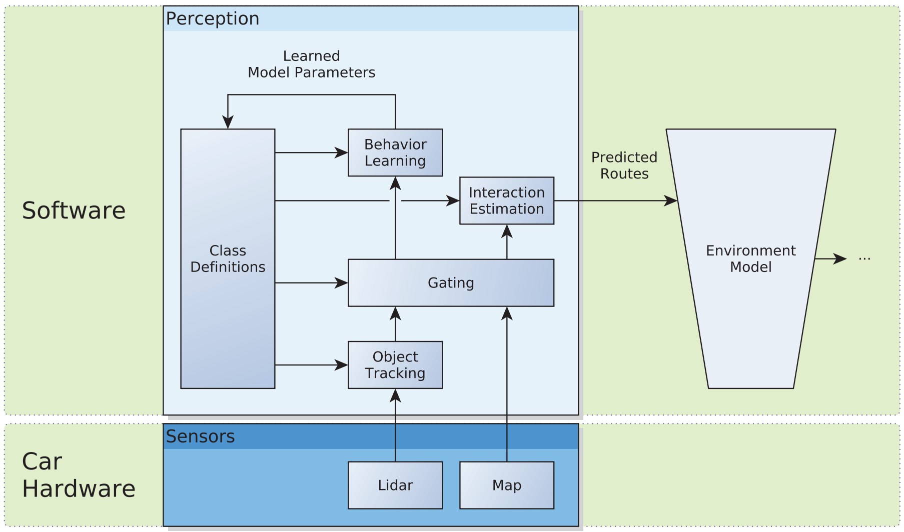


Figure 9.16: System overview depicted as part of an automated vehicle (comp. Fig. 2.3). Sensor data (e.g. from *Lidar*) and predefined *maps* are processed by the three inference methods. The OOFGML class definitions are the basis for all three steps. *Behavior Learning* updates the model parameters while *Interaction Estimation* provides the predicted routes as part of the environment model.

For the FOPL an existing implementation of the OPRML based on the discrete inference library libDAI [93] is chosen. LibDAI offers various generic inference algorithms, such as Loopy Belief Propagation [79], Fractional Belief Propagation [131] and Generalized Belief Propagation [134]. Additionally, parameter learning is supported by Expectation Maximization [36]. The implementation has the limitations of only being able to use discrete random variables and a limited number of aggregate slots (that can be mapped to multi-reference attributes in the OOFGML (Sec. 3.5)). Therefore several adaptations are added when implementing the application-specific OOFGML model (Fig. 9.13) in the OPRML (Fig. 9.17):

- Low-dimensional discrete state spaces are defined for all attributes.
- All continuous evidences are discretized using fuzzy logic.
- The class structure is adapted to reduce the number of classes and instances where reasonable, especially:
  - The road behavior is integrated into the route, more precisely in the `has` relation between the dynamic object and the route.
  - Following and crossing behaviors are combined in one Behavior class.
  - The `reactsTo` relation is integrated in the behavior class.
  - The `has` relation between dynamic object and behavior is integrated in the behavior class.

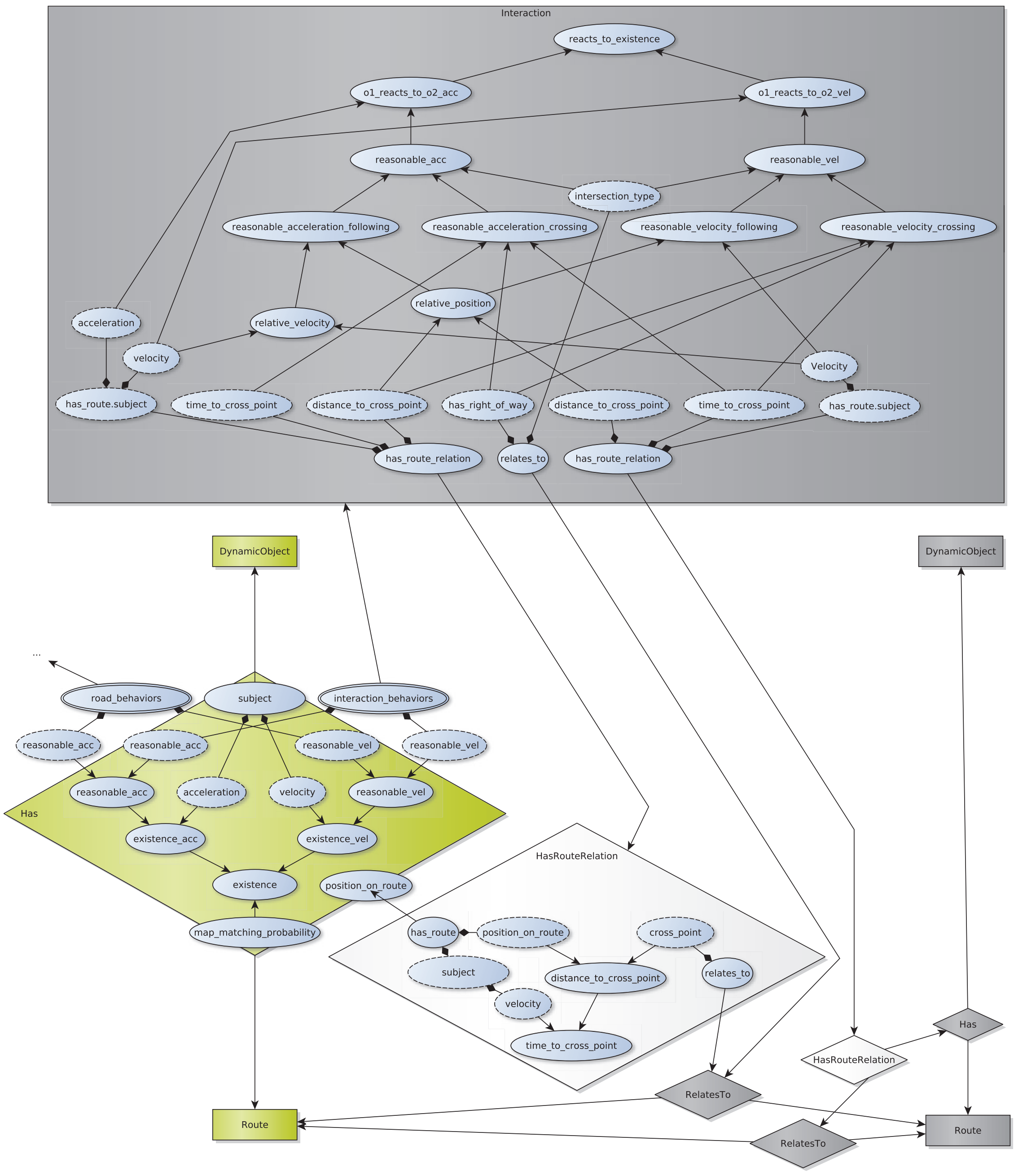


Figure 9.17: The OOFGML based definition (Fig. 9.13) is adapted to an object-oriented probabilistic relational model (OPRM) implementation. The dependencies in the factor graph are now replaced by arrows between attributes describing a Bayesian Network as described in Sec. 2.4.

- Behavior existence is not aggregated at the level of the existence attribute but at the level of the reasonable action.
- The calculation of distance and time to the crossing point is shared among behaviors and thus separated into a class related to a has-route relation and the relation to another route.
- All dependencies are described as causalities, resulting in a Bayesian Network (Sec. 2.4.2), which is the basis for the OPRM.

This implementation of the single shot approach is used in the evaluation in Sec. 9.6.1. The more integrated temporal filtering approach is described in Sec. 9.5.

9.5 Integrated Temporal Model Approach

The goal of this approach is to simultaneously estimate the route, current state and future state of traffic participants in a single model. The evidences come directly from noisy sensor input and the multi target tracking is integrated into the estimation of behaviors, interactions and routes. The idea is that also basic state estimation and prediction can profit from high-level knowledge.

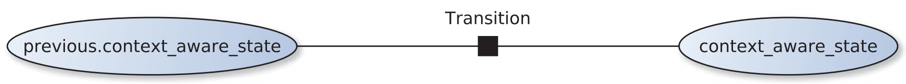
9.5.1 Behavior Estimation

The basic idea of this approach is that the state of the dynamic object in the current time slice can be derived by a transition from the context-aware state of the past time slice (Fig. 9.18a, comp. Fig. 9.4). All different actions that can be taken result in a mode of the next state variable. It is assumed that each of these modes belongs to one discrete behavior. The existence probability of the behavior matches the weight of the mode in the state. The state can therefore be seen as a gaussian mixture, similar to $\boldsymbol{W}_t$ and $\boldsymbol{V}_t$ in Fig. 2.15. Now each behavior describes a separate transition model (Fig. 9.18b) which puts the whole state transition into the class of switching transition model filters (Sec. 2.6.5).

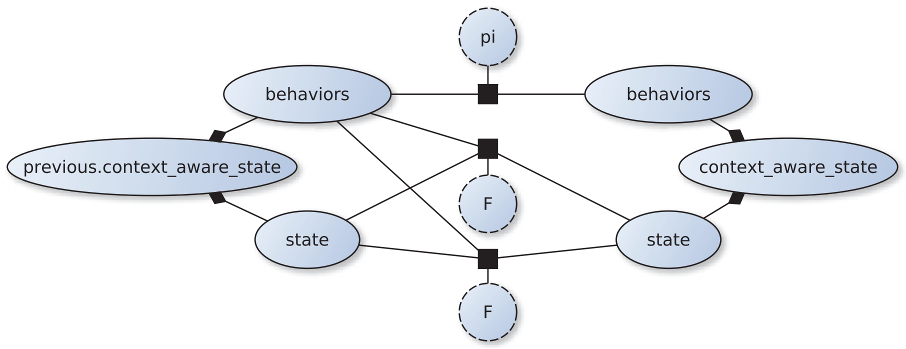
As described in Sec. 2.6.5 such a switching transition model can estimate the state and the mode probabilities from observations simultaneously and the IMM filter is an efficient implementation of such a model.

In the object-oriented description of this approach mode probability, state variable and the transition model descriptions are grouped in the behaviors of each dynamic object (Fig. 9.18c). The relation between objects give prior knowledge on possibly probable behaviors and prepare relative attributes for the context-aware state as described in Sec. 9.3.

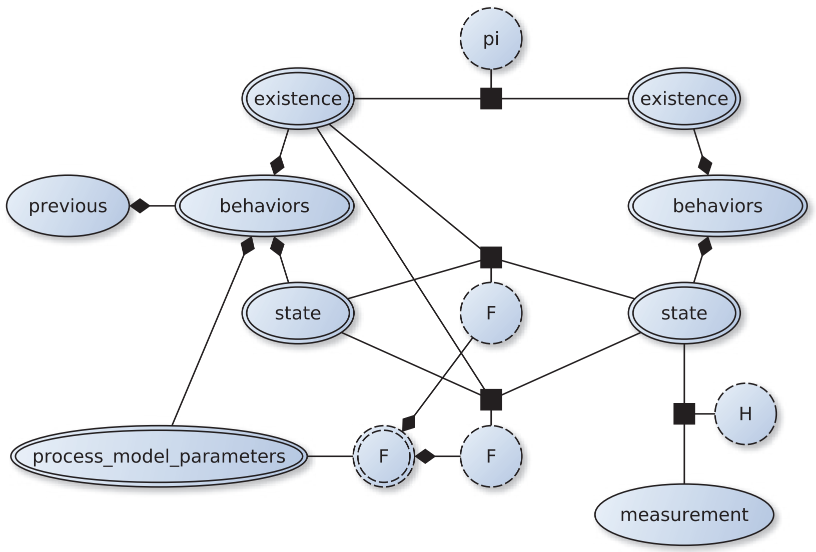
Transition model properties can be derived from the context-aware state. For example a curved road induces a reduced target velocity at a given point in the curve or another dynamic object motivates a transition model that adapts the



(a) The state at one timeslice can be derived via a transition function from the previous time slice.



(b) The `context_aware_state` can be seen as Gaussian mixture where every mode results from a separate transition model (with parameters F) corresponding to one behavior.



(c) In an object-oriented manner transition model, state variable and mode probability are grouped in the `Behavior` class.

Figure 9.18: Principle of behavior estimation in the integrated temporal model approach.

velocity to hold a desired distance to the object. Exemplary transition models are given in Sec. 9.5.3.

9.5.2 Classes and Attributes

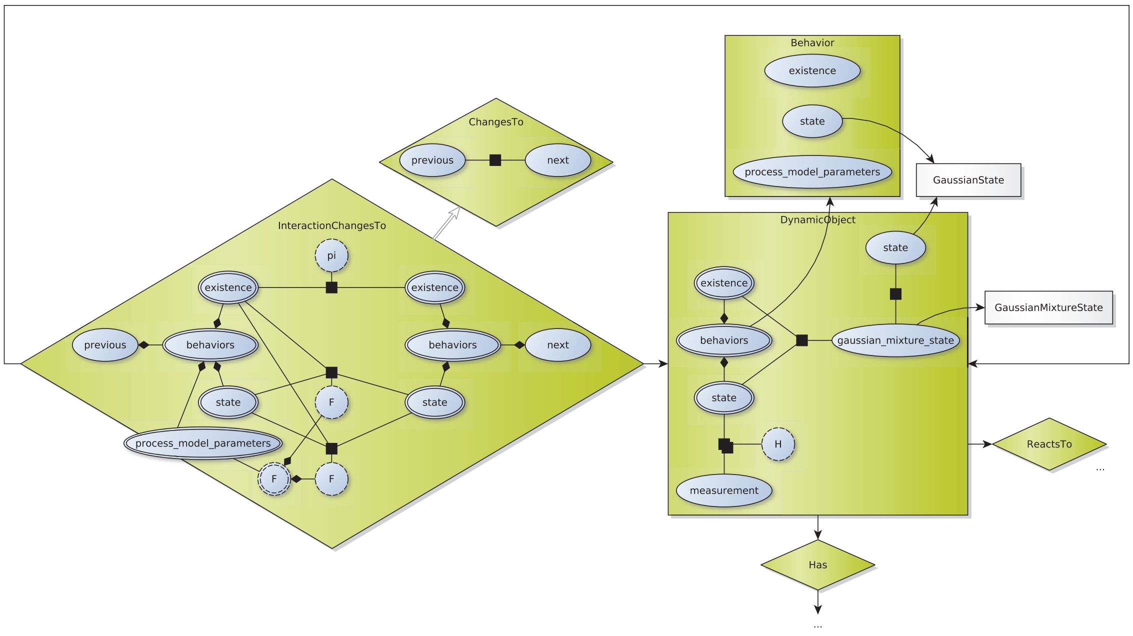


Figure 9.19: Behavior estimation approach integrated into the application ontology. Key functionality is placed in the ChangesTo relation of every DynamicObject entity. The objects' context is incorporated in the process_model_parameters as described in Fig. 9.20.

The ontology from Fig. 9.5 is used with a focus on the changesTo relation. Fig. 9.19 shows the classes around this relation. Every dynamic object in the time slices is embedded in the relational class structure of Fig. 9.6 with the existence probability estimation of routes and interactions. Visible in Fig. 9.19 is the grouping of state hypotheses in the behavior class, the conversion to gaussian mixture representations and gaussian representations of the state attribute as well as the transition model including the different mode-dependent continuous transition models and the discrete transition of the mode distribution. The inference via IMM filter is described in Sec. 9.5.4.

9.5.3 Dependency Model

The important part of the dependency model is the definition of the different behavior-dependent transition models. They are defined by the process_model_ parameters attribute which can depend on relative attributes induced by the ontology as depicted in Fig. 9.20.

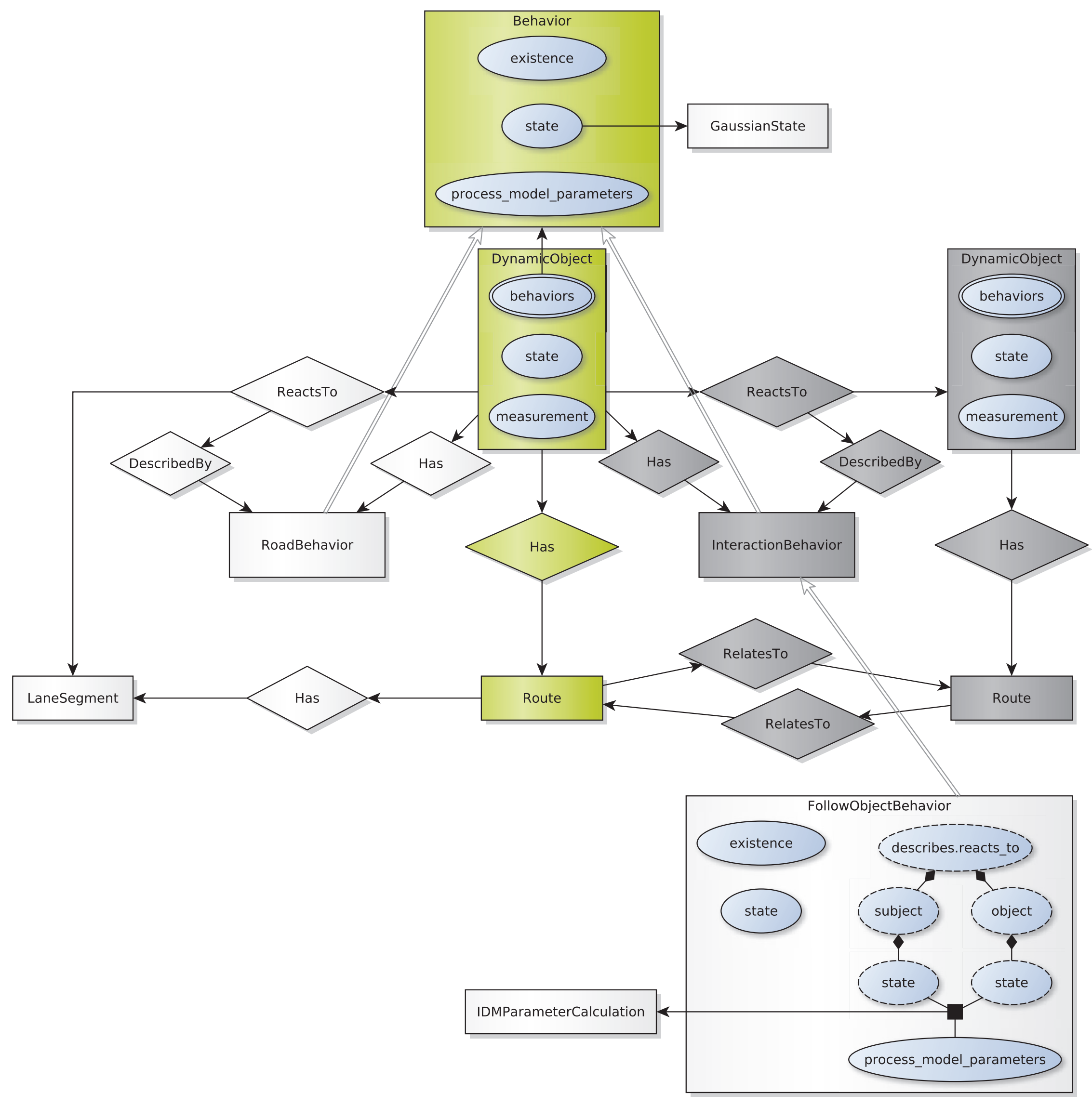


Figure 9.20: process_model_parameters are derived from relations given by the application ontology. The FollowObjectBehavior implements the behavior for following another vehicle using the intelligent driver model in IDMParameterCalculation.

For this work focus lies on longitudinal behaviors, such as braking in front of an intersection or adapting the velocity to follow another vehicle. The state of the dynamic objects can be described by the longitudinal position s_t and velocity v_t:

$$y_t = \begin{pmatrix} s_t \\ v_t \end{pmatrix} \tag{9.1}$$

This is the basic input to the transition models which are all of this type:

$$y_t = f(y_{t-1}, \theta) \tag{9.2}$$

with model parameters θ which can include additional (context-aware) state attributes.

Transition models are defined to represent the following longitudinal behaviors:

1. *constant-ride*
2. *brake-for-turn*
3. *brake-for-stop*
4. *follow-object*
5. *unknown-behavior*

The *constant-ride* and *unknown-behavior* behavior will be implemented by the constant velocity transition model, the *brake-for-turn* and *brake-for-stop* behavior will be implemented by the target velocity transition model and the *follow-object* behavior will be implemented by the intelligent driver model.

These transition models are defined in the following.

Constant Velocity Model

The Constant Velocity (CV) model is used to describe the *constant-ride* and the *unknown-behavior* behavior. It is known from many basic filter applications and is based on the physical formula

$$s = s_0 + v \cdot \Delta t. \tag{9.3}$$

The transition matrix F from Equ. 2.96 is set to

$$F_{CV} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \tag{9.4}$$

and the process noise covariance $\Sigma_{F_{CV}}$ is defined as

$$\Sigma_{F_{CV}} = \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix} \cdot q \cdot \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix}^T . \tag{9.5}$$

Then q can be seen as the allowed acceleration of the vehicle. If the CV model is used as single model to fit the whole process, this value must be quite high to match every acceleration and braking maneuver. Since in this approach this model is applied to the *constant-ride* behavior, the noise must be much smaller to match the small velocity changes that happen while driving without interactions. With a high variance it is used for the *unknown-behavior* behavior similar to the zero mean, high-variance Gaussian outlier process in [124].

The complete **CV transition model** can be written as

$$y_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \cdot y_{t-1} + w_t \tag{9.6}$$

with $w_t \sim \mathcal{N}_{[0,\Sigma_{F_{CV}}]}(w)$. Since it is a linear model the update rules from Table 2.2 can be applied during inference.

Target Velocity Model

For the *brake-to-turn* and *brake-to-stop* behavior the Target Velocity (TV) model is introduced. Key idea is to calculate the necessary acceleration a_t for reaching a target state x_T. The target state consists of a target velocity v_T at a specific target position s_T. The target velocity is 0 in the *brake-to-stop* case and equals a specific curve velocity in the *brake-to-turn* case. v_T and s_T can be loaded from map data and calculated as relative attributes using the OOFGML ontology (Sec. 9.3.2).

Assuming constant acceleration from current time slice t to the future target state x_T leads to

$$a_t = \frac{v_{t-1}^2 - v_T^2}{2 \cdot (s_T - s_{t-1})}. \tag{9.7}$$

This calculated currently best acceleration a_t can be used as control input. With the transition matrix F_{CV} from the CV model (Equ. 9.4) the **TV transition model** can be written as

$$y_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \cdot y_{t-1} + \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix} \cdot a_t + w_t \tag{9.8}$$

with $w_t \sim \mathcal{N}_{[0,\Sigma_{F_{TV}}]}(w)$ and $\Sigma_{F_{TV}}$ defined analogously to the CV model (Equ. 9.5). q can be used to allow derivation from the estimated acceleration a_t.

Note that y_t depends on the velocity of the previous time slice quadratically via a_t. Hence, the system becomes nonlinear. Therefore, instead of simply using the update rule 3 of Table 2.2 the Jacobian matrix J_t is taken for the covariance update:

$$\mu_{y_t} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \cdot \mu_{y_{t-1}} + \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix} \cdot \frac{\mu^2_{v_{t-1}} - v_T^2}{2 \cdot (s_T - \mu_{s_{t-1}})} \tag{9.9}$$

$$\Sigma_{y_t} = J_t \Sigma_{y_{t-1}} J_t^T \tag{9.10}$$

with

$$J_t = \begin{pmatrix} \frac{\partial \hat{s}_t}{\partial s_{t-1}} & \frac{\partial \hat{s}_t}{\partial v_{t-1}} \\ \frac{\partial \hat{v}_t}{\partial s_{t-1}} & \frac{\partial \hat{v}_t}{\partial v_{t-1}} \end{pmatrix} \tag{9.11}$$

$$\frac{\partial \hat{s}_t}{\partial s_{t-1}} = 1 + \frac{(v_{t-1}^2 - v_T^2)}{4 \cdot (s_T - s_{t-1})^2} \cdot \Delta t \tag{9.12}$$

$$\frac{\partial \hat{s}_t}{\partial v_{t-1}} = \Delta t + \frac{v_{t-1}}{2 \cdot (s_T - s_{t-1})} \cdot \Delta t^2 \tag{9.13}$$

$$\frac{\partial \hat{v}_t}{\partial s_{t-1}} = \frac{v_{t-1}^2 - v_T^2}{2 \cdot (s_T - s_{t-1})^2} \cdot \Delta t \tag{9.14}$$

$$\frac{\partial \hat{v}_t}{\partial v_{t-1}} = 1 + \frac{v_{t-1}^2}{2 \cdot (s_T - s_{t-1})} \cdot \Delta t. \tag{9.15}$$

Intelligent Driver Model

For the *follow-object* behavior an adapted Intelligent Driver Model (IDM) is used. The IDM was originally introduced by Treiber et al. [127] for microscopic simulations. It is a longitudinal car following model that allows a (simulated) vehicle to adapt its velocity to a leading vehicle. It is one way to consider the environment when deciding about the motion of a dynamic object.

Therefore the following quantities are used from the context-aware state representation:

$s_A(t)$	The longitudinal position of the dynamic object itself
$v_A(t)$	The longitudinal velocity of the dynamic object itself
$\Delta s_A(t)$	The position difference $\Delta s_A(t) = s_B(t) - s_A(t)$ to the object in front
$\Delta v_A(t)$	The velocity difference $\Delta v_A(t) = v_B(t) - v_A(t)$ to the object in front

Additionally these parameters describe the driving style and physical constraints of the dynamic object:

v_0	The target velocity
b	A desired deceleration of the dynamic object
a_{max}	The maximum acceleration of the dynamic object
s_0	A minimum distance between the two objects
T	A desired time head away between the two objects
δ	An acceleration exponent. The higher this exponent, the higher the slope of the acceleration of the dynamic object

Standard values for these parameters including parameters to drive in town or on highway are given in [33, p.22] .

The original IDM is defined by

$$\begin{aligned} a_{IDM}(t) &= IDM\big(s_A(t), v_A(t), s_B(t), v_B(t)\big) && (9.16)\\ &= a_{max}\left[1 - \left(\frac{v_A(t)}{v_0}\right)^{\delta} - \left(\frac{s_A^*(t)}{\Delta s_A(t)}\right)^{2}\right] && (9.17)\end{aligned}$$

with a target distance $s_A^*(t)$, given by

$$s_A^*(t) = s_0 + \max\left(0, T \cdot v_A(t) - \frac{v_A(t) \cdot \Delta v_A(t)}{2\sqrt{a_{max} \cdot b}}\right) \tag{9.18}$$

This definition delivers proper acceleration values for physically reasonable input. When using it in a transition model also hypotheses that are not physically reasonable will be evaluated. If the distance between the objects is close to zero, the acceleration gets unrealistically high. Similarly the model is not defined for negative relative velocities. Therefore, the model has to be slightly adapted before using it as a transition model.

The unrealistically high outputs are overcome by limiting $a_{IDM}(t)$ to a maximum acceleration a_{max} and deceleration b_{max}:

$$a_t = \min\Big(a_{max}, \max\big(a_{IDM}(t), -b_{max}\big)\Big) \tag{9.19}$$

This calculated currently best acceleration a_t is used analogously to the target velocity model to build up the **IDM transition model**:

$$y_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \cdot y_{t-1} + \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix} \cdot a_t + w_t \tag{9.20}$$

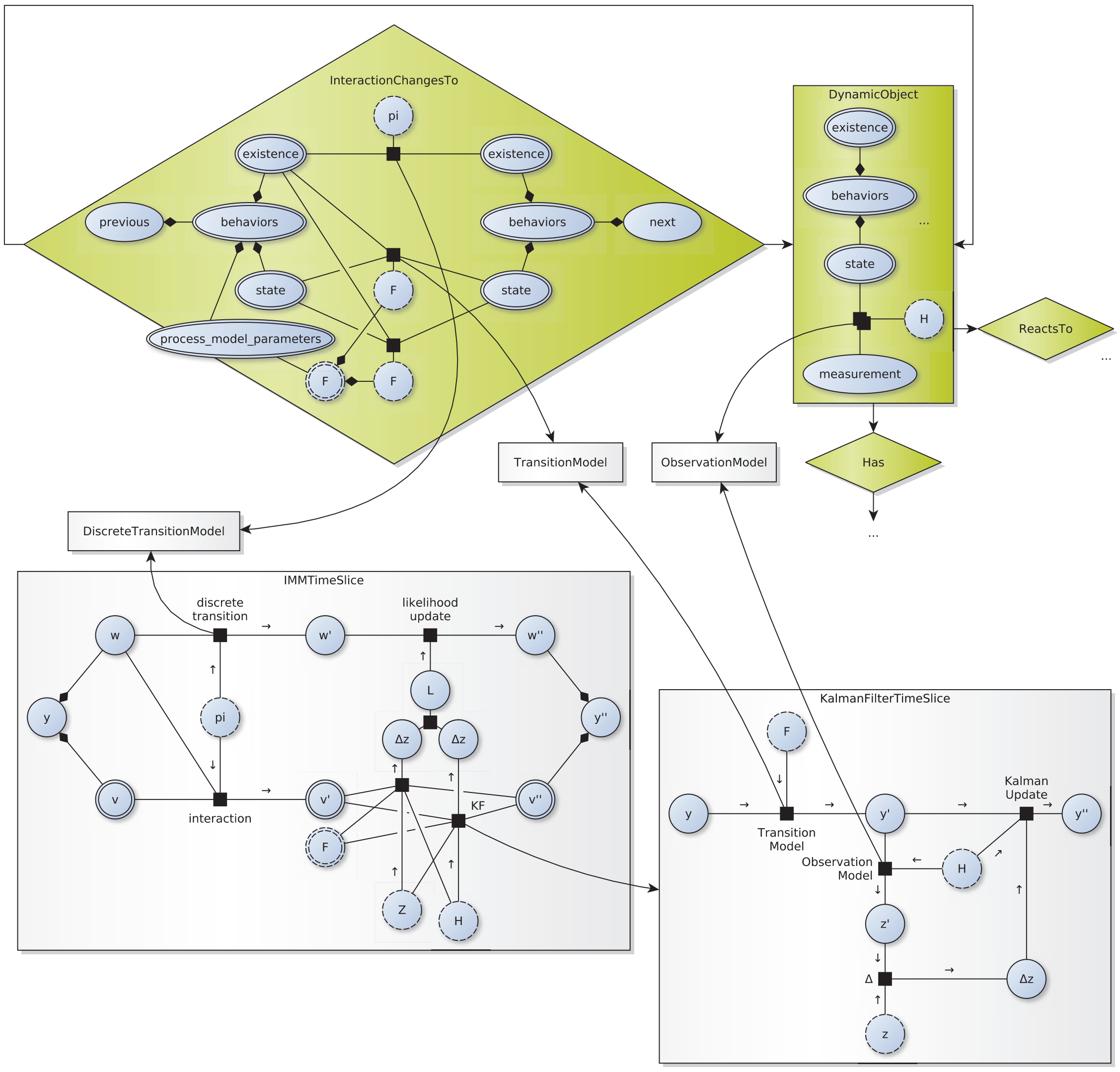


Figure 9.21: Additional classes define the inference method that is based on the IMM filter principle. Single letter attribute names have been chosen to emphasize the correlation to Sec. 2.6, especially `IMMTimeSlice` corresponds to the FFG in Fig. 2.15.

9.5.4 Inference Methods

As already mentioned, the behavior-mode-dependent temporal component can be seen as a switching transition model filter (Sec. 2.6.5). Efficient inference can be achieved by using the principle of the IMM filter. Inference classes have to be created analogously to the Kalman filter classes in Sec. 6.2.5. Fig. 9.21 shows how corresponding inference classes can be stacked around the existing ontology to describe the IMM filter inference schedule. Observation model, continuous transition models and behavior mode distribution transition model are mapped to the already defined dependency model.

The overall inference in this approach is split into two steps:

1. A gating step reduces combinatorial complexity when receiving new measurements (comp. Sec. 9.3.3).
2. The actual inference of interactions and probable routes is performed on a reduced subset of instances using the full detailed model based on multiple IMM filters (Sec. 9.5.2).

These steps can be modeled in a central class holding a set of processing scripts that instantiate the corresponding sub models analogously to the single shot approach in Sec. 9.4.4.

9.5.5 Implementation Opportunities

One of the five implementation opportunities described in Sec. 3.4 has to be chosen. For this application a *specialized implementation* that matches the special requirements is chosen. The gating is solved similarly as in the single shot approach (Sec. 9.4). The actual interaction estimation is implemented in an object-oriented fashion around the established filter principle of the IMM filter.

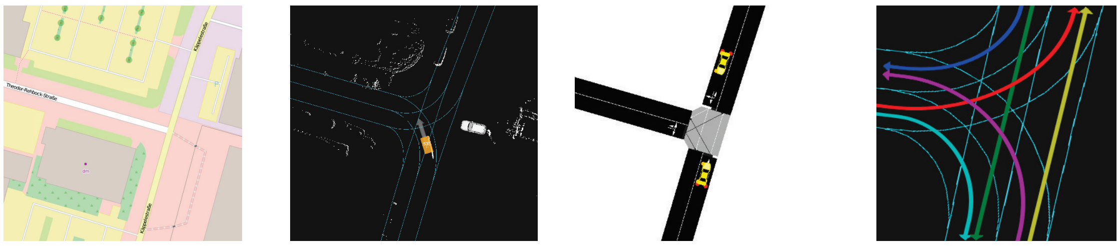
This implementation of the temporal filter approach is used in the evaluation in Sec. 9.6.2 and Sec. 9.6.3.

9.6 Evaluation

The two approaches for route, behavior and trajectory estimation are chosen to show the capabilities and applicability of the proposed OOFGML. Nevertheless, also a look at the performance in the application's field of research is taken.

The approaches are evaluated on different experiment setups to show their capability of predicting traffic participants' routes, behaviors and future states as well as estimating their current state. The following sections focus on different aspects of the evaluated systems.

9.6.1 Route Prediction from Object Constellations



(a) Chosen 3-way intersection in OpenStreetMap. The service way on the right was usable to place the observation car.

(b) Real Sensor data from CoCar (white) including laser raw data (white points) and processed object information with boundary box (orange) and estimated velocity (gray arrow).

(c) The very same scene modeled in the open source traffic simulation package SUMO [78] for easily creating a large learning database.

(d) Color-coded routes derived from the lanelet-based map [27] (thin cyan boundaries).

Figure 9.22: Experiment setup for route prediction from object constellations. [9]

The route prediction from object constellations was evaluated using the single shot approach (Sec. 9.4) at a three-way bidirectional intersection consisting of a straight priority road and a side road. The three-way intersection is depicted in Fig. 9.22. Sensor data for learning behavior models was recorded from real sensors of the test vehicle CoCar (cognitive car) [74] of the FZI Research Center for Information Technology as well as from the open source traffic simulation package SUMO [78]. Following [98] all recordings are divided into training and test data in the ratio 3 : 1. Note that the data has to be separated on whole vehicle track basis since the high correlation between consecutive time frames would lead to falsely good results. The experiment setup and evaluation results are described in detail in [9]. An excerpt is given here.

The output of the OPRM includes the route probabilities as well as the interaction probabilities. The interactions are visualized as cyan connections between objects and give an impression which interaction has a high impact on the route decision (Fig. 9.23). To evaluate the route prediction, a threshold-based classifier is applied, that delivers *unsure* if no route probability reaches the threshold. If a route probability reaches the threshold the route is compared to the ground truth route to obtain *true* and *false* classifications. Results during approaching the intersection are given in Fig. 9.24. Comparing the results without considering interactions (Fig. 9.24a) to the results with interaction detection (Fig. 9.24b) the following findings can be gathered which are also presented in a video[1].

Interactions Allowing More Decisions

The number of correct route predictions has increased, especially in the range of 10 - 0 meters in front of the intersection (Fig. 9.24):

Interaction-dependent behavior models can help predicting the correct route in situations where road-dependent behaviors are very similar for different routes. Such a situation is given when a vehicle approaches an intersection from the side street (Fig. 9.25), where the road-dependent behaviors for driving left or right are very similar.

Looking at the internally estimated existence values for the instantiated route and interaction classes, it can be stated that the improved estimation correlates with the detected interactions (Fig. 9.26b).

This is substantiated by a separate experiment only concerning vehicles approaching the intersection from the side road with traffic on one of the main road's lanes (Fig. 9.28a).

[1]Video: `url.fzi.de/interaction`

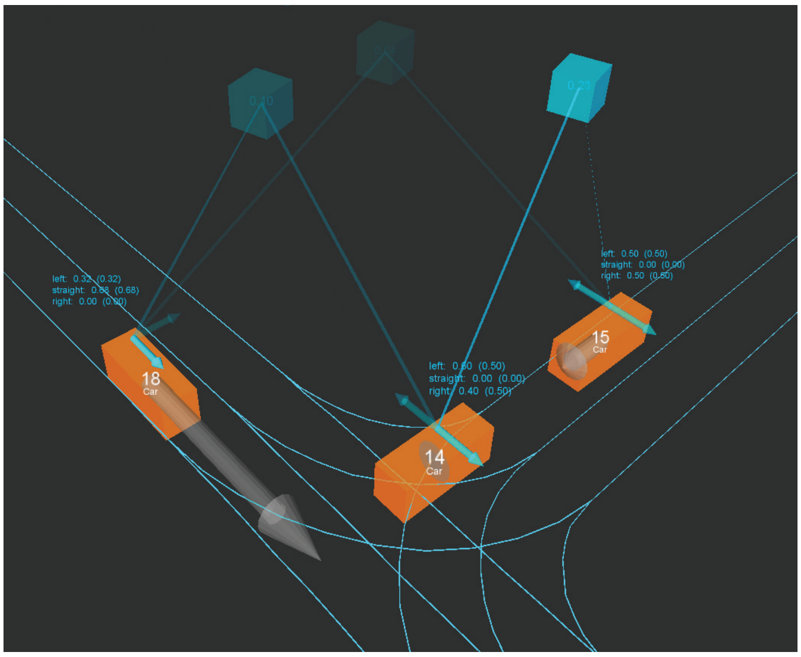
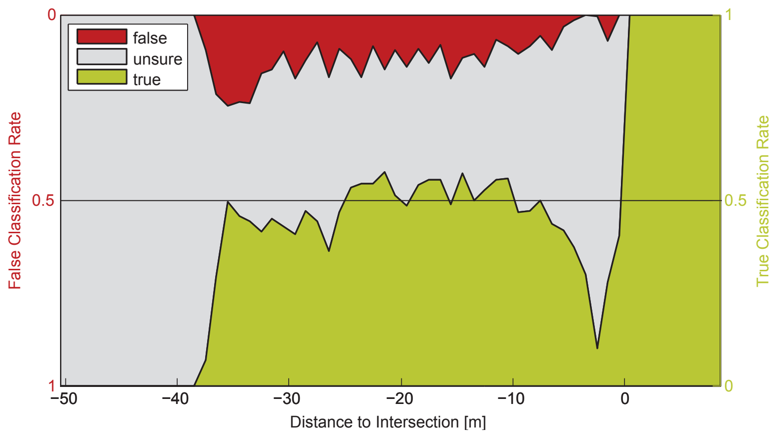
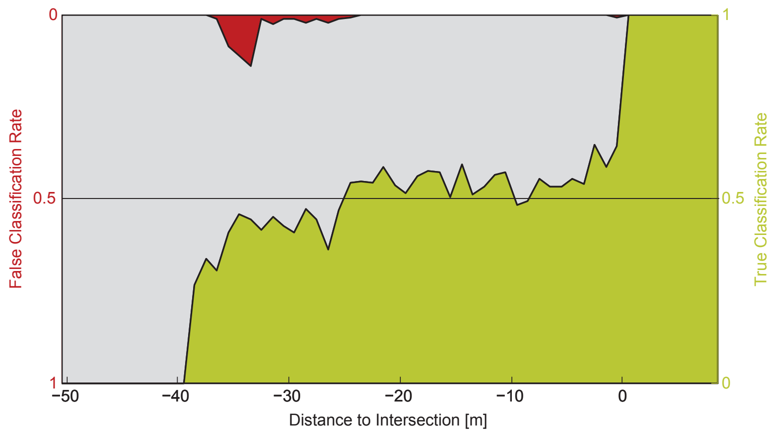


Figure 9.23: Exemplary interaction detection and route prediction at an intersection. The cyan cubes show the estimated interaction potential. Vehicle 14 is predicted as turning left because this is the only blocked route while it is waiting. The route of vehicle 15 cannot be predicted because the braking behavior is caused by the preceding vehicle. [9]



(a) without interaction detection



(b) with interaction detection

Figure 9.24: Classification results by using a classifier with threshold of 53% depending on the distance to the intersection. Detecting interactions, more route predictions can be made in 10 - 0 meters to the intersection and false predictions can be reduced. [9]

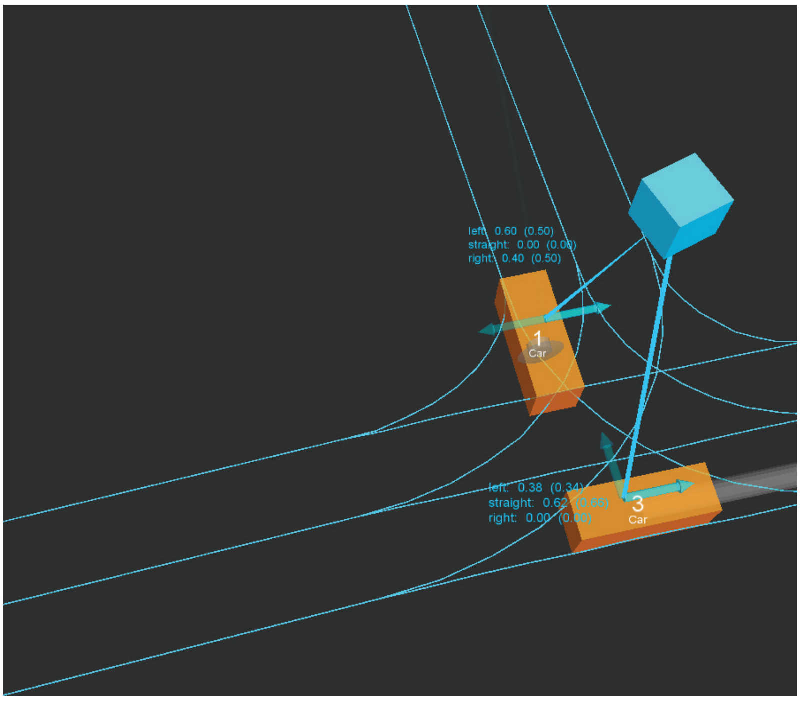
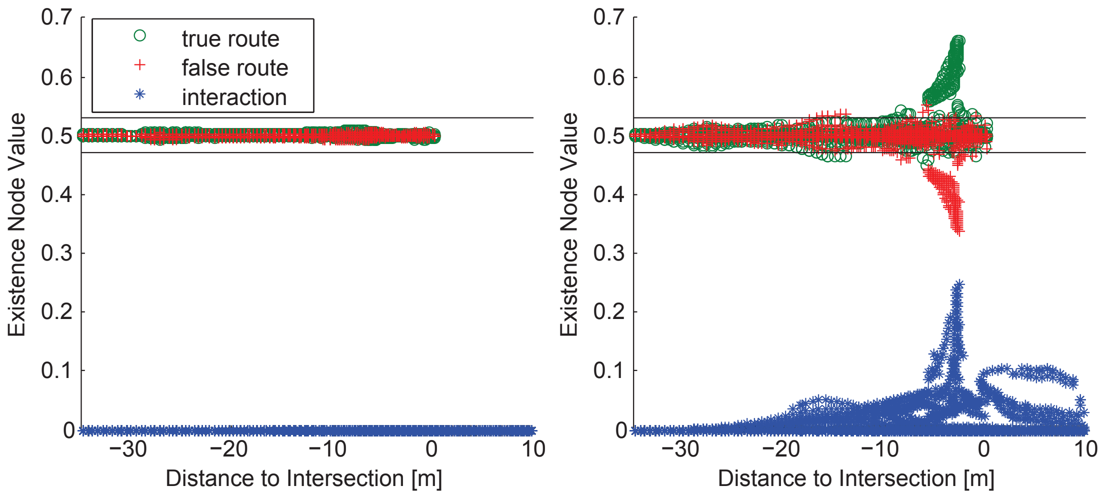


Figure 9.25: When a vehicle (1) approaches the intersection from the side street, road-dependent behaviors for driving left or right are very similar. Interactions (cyan cube) with other passing (route-blocking) vehicles can help estimate the correct route (small cyan arrows). [9]



(a) without interaction detection

(b) with interaction detection

Figure 9.26: Values of Route Existence nodes, color-coded by correct routes (green) and false routes (red). The value of the Interaction Existence node (blue) shows that the better result with interaction-dependent behavior models (correct routes above threshold, false routes below threshold) is correlated to the occurance of interactions. Adapted from [9]

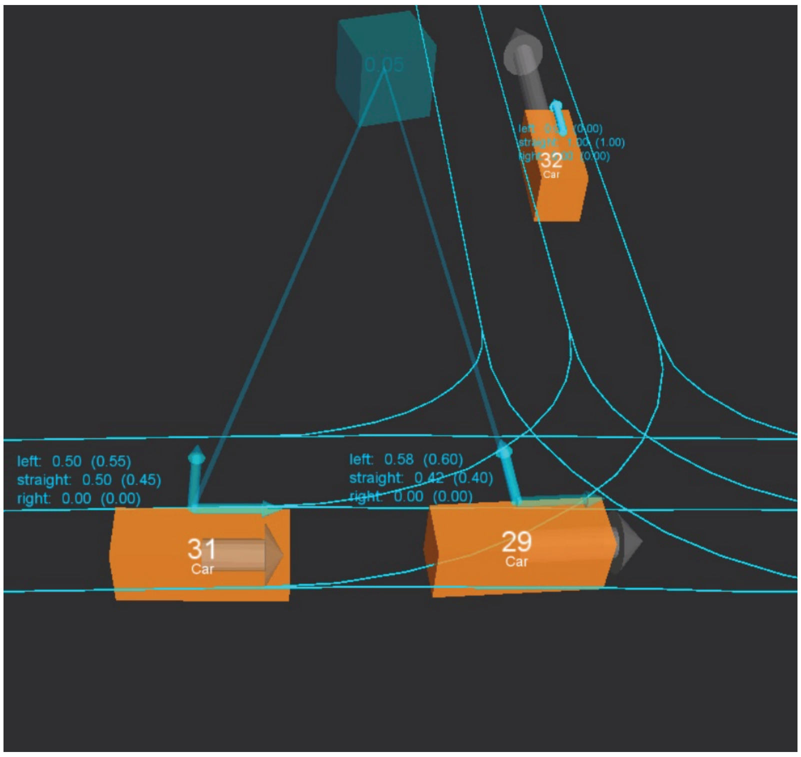


Figure 9.27: When a vehicle (31) is braking because of a preceding vehicle it can be falsely interpreted as braking for a turn if no interactions are considered. The detected interaction is visualized by the cyan cube. [9]

Table 9.1: Results of the four experiment configurations depicted in Fig. 9.28. Threshold-based route decision within 5 meters in front of the intersection, without (w/o) and with (w) interaction-dependent behavior models. [9]

Scenario	**Interactions**	**false**	**unsure**	**true**
left/right	**w/o**	0%	100%	0%
	w	1%	49%	50%
left/straight	**w/o**	0%	11%	89%
	w	0%	2%	98%
random	**w/o**	2%	72%	26%
	w	0%	48%	52%
real	**w/o**	6%	18%	76%
	w	4%	21%	75%

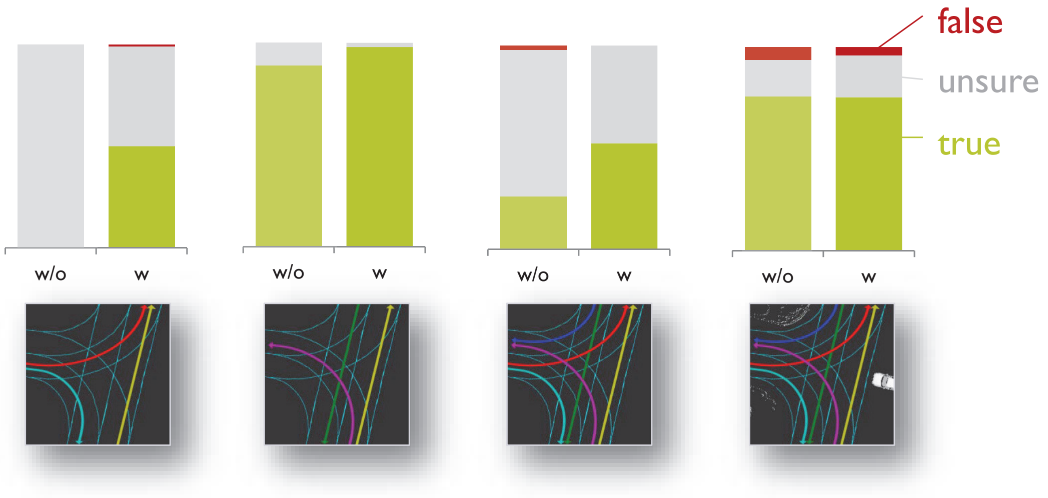


Figure 9.28: Four experiment configurations and their results for a threshold-based route decision within 5 meters in front of the intersection, without (w/o) and with (w) interaction-dependent behavior models. From left to right: a) Entering the main street left or right, b) leaving the main street with opposing traffic or staying straight, c) random traffic on every route (details in Fig. 9.24), d) random traffic on real sensor data. [9]

Interactions Avoiding False Decisions

A second result is that if interactions are considered, false route decisions can be reduced in a large range before the intersection (Fig. 9.24).

When using only road-dependent behavior models, behaviors that are caused by interactions can lead to false predictions. Such a situation is given in Fig. 9.27: If a vehicle brakes in front of an intersection because of another vehicle, the braking behavior misleadingly results in a high probability for driving a turn. With interaction-dependent behavior models this can be correctly explained by the interaction: Both routes stay probable and fewer false decisions are made by the threshold-based route decision.

9.6.2 Route Prediction with Interaction-dependent Motion Models

The route prediction using interaction-dependent transition models was evaluated using the test vehicle CoCar (cognitive car) [74] of the FZI Research Center for Information Technology at a 4-way intersection. The high precision internal state estimation sensor delivers accurate position measurements that can be used as ground truth information. At the intersection 4 different maneuvers have been recorded multiple times:

- 5 x Turn right
- 5 x Turn left
- 5 x Straight
- 5 x Stop

The temporal filter approach (Sec. 9.5) was used with 3 different behavior models:

- *constant-ride* (straight)
- *brake-for-turn* (turn)
- *brake-for-stop* (stop)

The estimated behavior is compared to the actually driven behavior (Fig. 9.29). One second before reaching the intersection the three behaviors right, straight and stop are correctly detected. Only turning left is mixed with the straight behavior. At this intersection the left turn can be driven with high velocity and thus the fixed target velocity of the turn behavior does not map this behavior precisely.

It can be seen that the environment-dependent motion models can be used to estimate discrete behaviors of traffic participants if the behaviors are represented sufficiently.

		Actually driven Behavior			
		right	left	straight	stop
Estimated Behavior	turn	5	1	0	0
	straight	0	4	5	0
	stop	0	0	0	5

Figure 9.29: Results of the route prediction. The route is predicted 1 second before the intersection. Only driving the left turn is confused with driving straight. Adapted from [6].

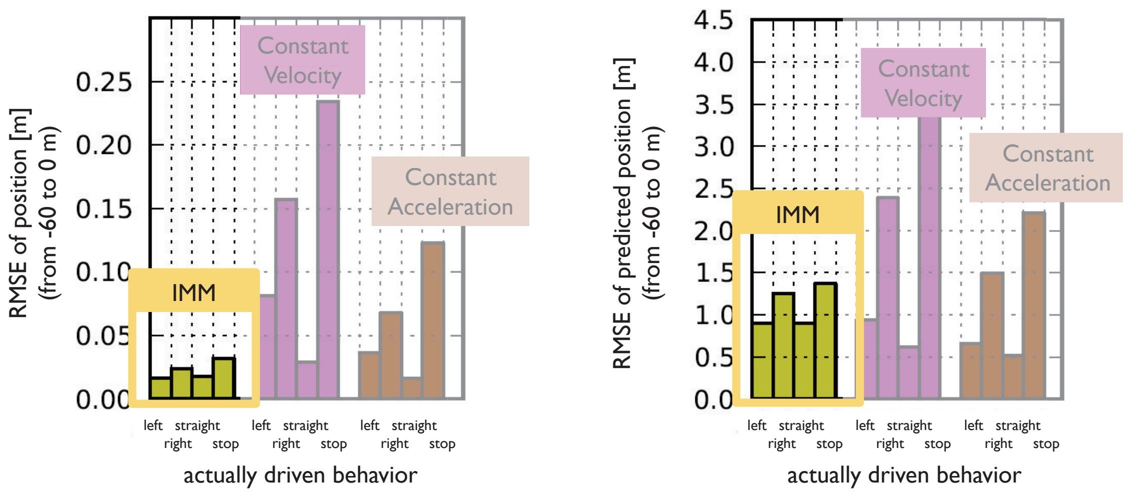


Figure 9.30: Results of the localization (left) and state prediction 2 seconds ahead (right). The approach using the proposed IMM filter is compared to two basic Kalman Filters with Constant Velocity Model and Constant Acceleration Model. [6]

9.6.3 Interaction-dependent State Prediction

The experiment setup is the same as in Sec. 9.6.2 with the test vehicle CoCar, the 4-way intersection, 20 tracks and the three instantiated behavior models.

For evaluating the state estimation and prediction the results are compared to two basic Kalman filters, one using a constant velocity model and one using a constant acceleration model. In all three filters the same measurement noise of 0.1 m is assumed. For every run, the root mean square error (RMSE) is calculated between the estimated value and the ground truth value from the high precision reference sensor. The average RMSE over all 5 runs of one maneuver can be compared to those of the two reference filters (Fig. 9.30).

In state estimation the new approach reduces the error from between 0.05 m and 0.25 m to a value that is lower than 0.03 m for all maneuvers. This already matches the precision of 0.02 m of the reference sensor. For evaluating the prediction results, the estimated velocity and acceleration (if available) is used to predict the state 2 seconds ahead. This value is then compared to the value of the reference sensor at that time. The improvement is not that significant but the maximum reduces from about 3.5 m (constant velocity) and 2.2 m (constant cceleration) to lower than 1.5 m.

This demonstrates that knowledge about environment-dependent behaviors in the filtering process improves state estimation and prediction, especially in the case of the advanced behaviors *stop* and *turn-right* based on the target velocity transition model.

9.7 Conclusion

This chapter showed how traffic participant prediction can profit from the proposed holistic modeling language.

9.7.1 Summary

The key points are:

- Compared to the related work the proposed traffic participant prediction methods based on the OOFGML introduce a new degree of generalization.
- Most of the DSL description can be shared among the two shown approaches including the basic relational idea, relative attributes and complexity-reducing gating.
- The single shot approach separates interaction estimation from object state estimation.

- Hard to parametrize behaviors are learned on class level.
- The single shot approach can be implemented as discrete FOPL.
- The temporal filter approach integrates high level estimation with object state estimation.
- Interaction-dependent transition models can represent different behaviors.
- The temporal filter approach is implemented as extended IMM filter.
- The evaluation shows promising results in route and behavior estimation as well as in state estimation and prediction.

9.7.2 Used Language Properties

The key goal of presenting this application within this thesis is to exemplarily show the usage of properties (Chapter 4) of the proposed language.

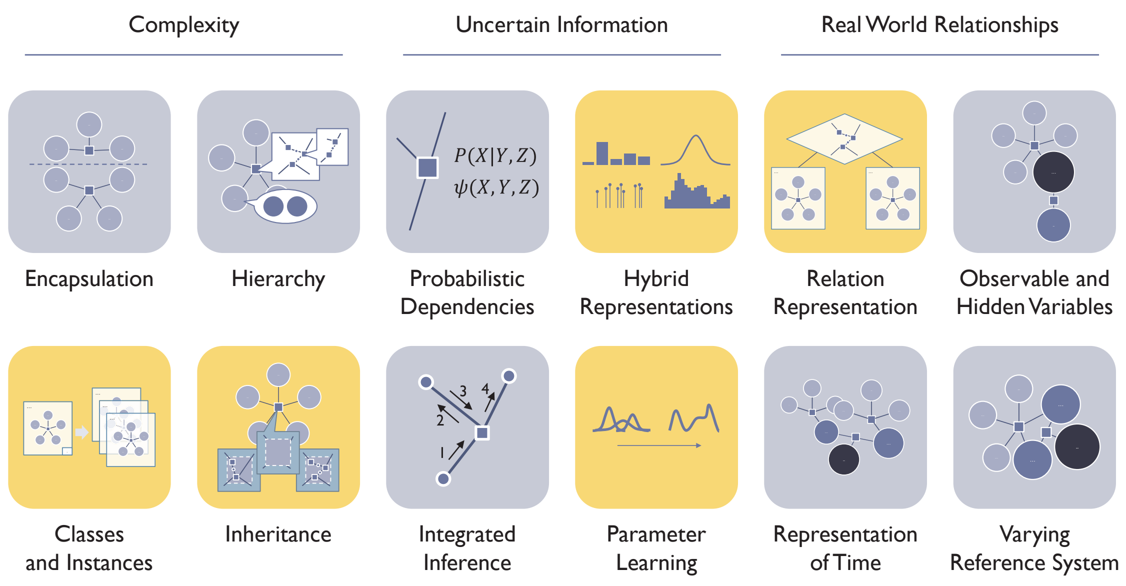


Figure 9.31: Some of the language properties that are beneficial to the route, behavior and trajectory estimation application are highlighted.

In this application the following properties (Fig. 9.31) are beneficial:

- **Classes and Instances** (Sec. 4.1.3):
  Just a little amount of classes is defined (Sec. 9.3) but instantiated multiple times to represent complex situations (Sec. 9.3.3). They are directly matched to OPRM classes in the single shot approach (Sec. 9.4.5).

- **Inheritance** (Sec. 4.1.4):
  Different sub classes of the behavior class (Sec. 9.4.2, Sec. 9.5.3) are used in the same application. They inherit from a parent class describing their common interface that allows an unspecialized view on them from other classes.
- **Hybrid Probability Distribution Representations** (Sec. 4.2.2):
  The discrete and continuous state representations are handled differently in the two approaches: In the single shot case the continuous states are quantized at the transition from temporal filtering to interaction estimation (Sec. 9.4.5). In the integrated temporal filter case (Sec. 9.5.4) discrete and continuous states are handled like in an IMM filter.
- **Parameter Learning** (Sec. 4.2.4):
  Behaviors of human traffic participants are hard to model by expert knowledge. The parameters on class level are learned from instantiations with varying number of traffic participants (Sec. 9.4.3).
- **Relation Representation** (Sec. 4.3.1):
  The relations between real world entities are modeled in the ontology (Sec. 9.3). During inference their existence is estimated in several estimation steps (Sec. 9.4.4 and Sec. 9.5.4) using the technique of gating (Sec. 9.3.3).

Note that besides the here highlighted properties also all other properties have been used in this application. They were already discussed in detail in the preceding chapters. Combining all these properties in one language makes the OOFGML a perfect foundation for solving the complex task of traffic participant prediction.

With its capability to also solve other estimation tasks (e.g. road layout estimation (Chapter 8) or localization (Chapter 7)) it is natural to use it also for a higher integration of all different estimation tasks in the field of autonomous driving.

10 Conclusion

This chapter concludes the findings of this work. A summary is given in Sec. 10.1 and an outlook to future research directions in Sec. 10.2.

10.1 Summary

This thesis focuses on a holistic approach to complete situation estimation in the traffic domain, especially for predicting future actions of traffic participants.

Traffic participant prediction is a challenging task that requires a complete and consistent understanding of the environment. Therefore, the estimation has to be seen as a holistic problem considering all processing steps from sensor data acquisition and basic estimation tasks to high level estimation of unobservable behaviors of traffic participants.

The goal of this thesis is to contribute to the development process of future autonomous driving systems by proposing an overall probabilistic modeling language for estimation problems and a domain specific specialization to the traffic domain. The approach has been applied to selected autonomous driving applications to show its feasibility and potential to advance towards human estimation capabilities.

A probabilistic modeling language called OOFGML has been introduced. It is the first modeling language that successfully implements the four ideas:

- A factor graph models probabilistic dependencies in a unified way.
- Scalability is achieved by object orientation including classes and instances and inheritence based on a class hierarchy.
- Message passing schedules and hybrid state space representations allow efficient inference.
- Relationships that are hard to define by expert knowledge are mastered by locally applied machine learning.

These four directions induce a lot of properties that are all available in the proposed modeling language.

It has been shown how the generic language can contribute to the development of applications: Generic estimation algorithms including inference methods can be described using the language. Depending on the domain, a domain specific language can be defined that includes entities and relations representing the real world objects and characteristics. For a specific application solving a task in the domain an application specific model can be derived from the domain specific language and the generic algorithm classes. Thus, the unified modeling language allows sharing and exchange of modules and inference methods between different applications.

This has been shown on exemplary applications in the traffic domain that cover a wide field of estimation tasks including ego vehicle localization, static environment perception and advanced estimation of interactions between dynamic objects. In ego vehicle localization it has been shown how advanced interpreted dynamic objects can be used to estimate the ego vehicle's pose relative to a road map more precisely than before. The static environment perception has focused on local dependencies to derive a coherent road layout from various environment measurements incorporating a hybrid state space. In interaction estimation object orientation has been utilized to achieve scalability and temporal dependencies have been modeled to estimate discrete vehicle behaviors as well as continuous vehicle states simultaneously.

The detailed description of these applications has shown that all estimation tasks are describable by the proposed language in a consistent way.

The language contributes to the development of autonomous driving systems bringing them closer to human performance. This includes capabilities like creating a coherent scene understanding from multiple environment measurements, estimating basic knowledge from scene understanding and deriving advanced interpretations from temporal context. These capabilities are covered by the exemplary applications that have been chosen to evaluate the proposed language.

While the applications show how the proposed language eases the development of future autonomous driving systems, the language can also be applied to other estimation tasks outside the traffic domain.

10.2 Outlook

The introduced modeling language opens a new field for subsequent research. Some of the possible directions are described in the following.

Deeper Integration of Estimation Tasks

Some of the estimation tasks in the autonomous driving domain have been considered in separate applications. A next step is to integrate them in one model. The complete model should reach from sensor data to estimated environment and driving decisions. Inference methods have to be chosen and integrated into the model. Finally, the model can be the concept for a whole system architecture for automated vehicles.

Hierarchical Learning

The class hierarchy offers new opportunities for parameter learning. In this work it has been shown that behavior models can be learned from real world observations but varying behaviors make it difficult to find precise parameters. The variation depends largely on the vehicles' physical capabilities but also on the drivers' mental constitution. While the second is hard to detect in advance, the physical capabilities could be derived from the vehicle class. An open question is how these insights can be stored efficiently in the class hierarchy. For example if a sports car is detected and a behavior is observed, should the new knowledge be stored in the sports car class, the car class, the vehicle class or the dynamic object class?

Deeper Learning

The language can be applied to other fields of learning. Recent deep learning methods can be integrated in factors. A first concept is given in [3]. It is an open question how learning algorithms can be adapted in a generic way to support the probabilistic graphical model. For example the deep learned model could provide additional information about the inner uncertainties to give a detailed probability distribution to the probabilistic model. Or the deep learned model supplies additional complementary information via an integrated additional learning task. The graphical model has the potential to integrate safety criterias based on expert knowledge into otherwise non-transparent approaches based on machine learning.

More Relational Database Features

The proposed definition of the language supports direct references via attributes. Often classes have to be referenced explicitly during instantiation although the information would be already available in the existing relations. An extension towards more SQL-like queries is conceivable, such as for example "reference the behavior class, that describes the reaction of object A to object B". Additionally if focusing on relational components and the inference of additional relations from observed relations new efficient data structures can be introduced. A first idea of an efficient tensor representation is presented in [10].

Language Specification, Tooling and Software Integration

The proposed language is a basis for describing all estimation tasks in a coherent way. The language can already help developers to model complex systems with deep integration of various estimation tasks. Nevertheless the expert still needs a not to be underestimated amount of knowledge to not step into pitfalls. It is desirable to support the whole development process by specialized tooling. A graphical editor can support the construction of the domain specific graphical model. Knowledge about possible inference methods could be presented to the user and replacement rules (like in the Kalman filter example) can be automatically applied.

All constraints and restrictions should be supported by the language definition. Existing formal languages and tooling as for example the UML and corresponding editors can be utilized in more detail than it has been only rudimentarily started in this work. Automatic consistency checks will be possible and code generators can directly produce efficient machine code from the graphical description.

The same base graphical model can be used for automated vehicles with different capabilities including sensor setup and computing performance. If the language is supported in all layers from graphical interface down to the online running system it is also imaginable that an online performance assessment (similar to the one presented in [13]) can utilize the class structure to dynamically switch to the currently best fitting technology. Increased robustness is one of the core features to make the dream of self driving cars come true.

List of Figures

List of Tables

Own Publications

This list includes all publications where the author of this thesis is either the main author or has contributed to the publication.

[1] Qi Chen, Ting Yuan, Axel Gern, Tobias Roth, Florian Kuhnt, Jakob Breu, Miro Bogdanovic, and Christian Weiss. DSRC and Radar Object Matching for Cooperative Driver Assistance Systems. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1348–1354, 2015.

[2] Tobias Fleck, Karam Daaboul, Michael Weber, Philip Schörner, Marek Wehmer, Jens Doll, Stefan Orf, Nico Sussmann, Christian Hubschneider, Marc René Zofka, Florian Kuhnt, Ralf Kohlhaas, Ingmar Baumgart, Raoul Zöllner, and J. Marius Zöllner. Towards Large Scale Urban Traffic Reference Data: Smart Infrastructure in the Test Area Autonomous Driving Baden-Württemberg. In *15th International Conference on Intelligent Autonomous Systems (IAS)*, pages 964–982. Springer International Publishing, 2018.

[3] Christian Hubschneider, Jens Doll, Michael Weber, Sebastian Klemm, Florian Kuhnt, and J. Marius Zöllner. Integrating End-to-End Learned Steering into Probabilistic Autonomous Driving. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017.

[4] Sebastian Klemm, Marc Essinger, Jan Oberländer, Marc René Zofka, Florian Kuhnt, Michael Weber, Ralf Kohlhaas, Alexander Kohs, Arne Roennau, Thomas Schamm, and J. Marius Zöllner. Autonomous Multi-Story Navigation for Valet Parking. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016.

[5] Florian Kuhnt, Ralf Kohlhaas, Rüdiger Jordan, Thomas Gußner, Thomas Gumpp, Thomas Schamm, and J. Marius Zöllner. Particle filter map matching and trajectory prediction using a spline based intersection model. In *2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1892–1893, 2014.

[6] Florian Kuhnt, Ralf Kohlhaas, Thomas Schamm, and J. Marius Zöllner. Towards a Unified Traffic Situation Estimation Model – Street-dependent Behaviour and Motion Models –. In *2015 IEEE 18th International Conference on Information Fusion (Fusion)*, pages 1223–1229, 2015.

[7] Florian Kuhnt, Stefan Orf, Sebastian Klemm, and J. Marius Zöllner. Lane-precise Localization of Intelligent Vehicles Using the Surrounding Object

Constellation. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016.

[8] Florian Kuhnt, Micha Pfeiffer, Peter Zimmer, David Zimmerer, Jan Markus Gomer, Vitali Kaiser, Ralf Kohlhaas, and J. Marius Zöllner. Robust environment perception for the audi autonomous driving cup. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1424–1431, 2016.

[9] Florian Kuhnt, Jens Schulz, Thomas Schamm, and J Marius Zöllner. Understanding Interactions between Traffic Participants based on Learned Behaviors. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[10] Dominik Petrich, Darius Azarfar, Florian Kuhnt, and J. Marius Zöllner. The Fingerprint of a Traffic Situation: A Semantic Relationship Tensor for Situation Description and Awareness. In *2018 IEEE 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 429–435. IEEE, 2018.

[11] Fabian Poggenhans, Jan-hendrik Pauls, Johannes Janosovits, Stefan Orf, Maximilian Naumann, Florian Kuhnt, and Matthias Mayr. Lanelet2 : A high-definition map framework for the future of automated driving. In *2018 IEEE 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[12] Jan Erik Stellet, Christian Heigele, Florian Kuhnt, and J. Marius Zöllner. Performance Evaluation and Statistical Analysis of Algorithms for Ego-Motion Estimation. In *2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[13] Ömer Sahin Tas, Stefan Hörmann, Bernd Schäufele, and Florian Kuhnt. Automated Vehicle System Architecture with Performance Assessment. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017.

[14] Ömer Sahin Tas, Florian Kuhnt, J. Marius Zöllner, and Christoph Stiller. Functional System Architectures towards Fully Automated Driving. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[15] Peter Wolf, Karl Kurzer, Tobias Wingert, Florian Kuhnt, and J. Marius Zöllner. Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 993–1000, 2018.

[16] Marc René Zofka, Sebastian Klemm, Florian Kuhnt, Thomas Schamm, and J. Marius Zöllner. Testing and Validating High Level Components for Automated Driving : Simulation Framework for Traffic Scenarios. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[17] Marc René Zofka, Florian Kuhnt, Ralf Kohlhaas, Christoph Rist, Thomas Schamm, and J. Marius Zöllner. Data-Driven Simulation and Parametrization of Traffic Scenarios for the Development of Advanced Driver Assistance

Systems. In *2015 IEEE 18th International Conference on Information Fusion (Fusion)*, pages 1422–1428, 2015.

[18] Marc René Zofka, Florian Kuhnt, Ralf Kohlhaas, and J. Marius Zöllner. Simulation framework for the development of autonomous small scale vehicles. *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 318–324, 2016.

Bibliography

[19] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[20] Deborah J Armstrong. The Quarks of Object-Oriented Development. *Communications of the ACM*, 49(2):123–129, 2006.

[21] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[22] Olav Bangsø. *Object Oriented Bayesian Networks*. PhD thesis, Aalborg University, 2004.

[23] Yaakov Bar-Shalom and Thomas E. Fortmann. *Tracking and data association*. Academic Press Professional, Inc., 1987.

[24] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*, volume 9. 2001.

[25] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[26] Alexander Barth and Uwe Franke. Where will the oncoming vehicle be the next second? In *2008 IEEE Intelligent Vehicles Symposium*, pages 1068–1073, jun 2008.

[27] Philipp Bender, Julius Ziegler, and Christoph Stiller. Lanelets: Efficient map representation for autonomous driving. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 420–425, 2014.

[28] Christopher M. Bishop. *Pattern Recognition and Machine Learning*, volume 4. 2006.

[29] Henk A. P. Blom and Yaakov Bar-Shalom. Interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783, 1988.

[30] Lukas Bolliger. *Digital estimation of continuous-time signals using factor graphs*. PhD thesis, 2012.

[31] Amol Borkar, Monson Hayes, and Mark T. Smith. Robust Lane Detection and Tracking with RANSAC and Kalman Filter. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3261–3264. IEEE, 2009.

[32] Ingemar J. Cox and Sunita L. Hingorani. An efficient implementation of Reid's multiple hypothesis tracker algorithm and its evaluation for the purpose of visual tracking. *Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[33] Jörg Dallmeyer. *Simulation des Straßenverkehrs in der Großstadt: das Mit-und Gegeneinander verschiedener Verkehrsteilnehmertypen*. Springer-Verlag, 2014.

[34] Radu Danescu and Sergiu Nedevschi. Probabilistic lane tracking in difficult road scenarios using stereovision. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):272–282, 2009.

[35] Justin Dauwels. *On graphical models for communications and machine learning: algorithms, bounds, and analog implementation*. PhD thesis, 2005.

[36] Justin Dauwels, Sascha Korl, and Hans-Andrea Loeliger. Expectation maximization as message passing. In *Proc. IEEE Int. Symp. Information Theory*, pages 583–586, 2005.

[37] Justin Dauwels, Sascha Korl, and Hans-Andrea Loeliger. Steepest descent on factor graphs. *Proc. IEEE Information Theory Workshop*, (4):42–46, 2005.

[38] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Artificial Intelligence*, 1989.

[39] Frank Dellaert. Factor graphs and GTSAM: A hands-on introduction. 2012.

[40] Frank Dellaert and Michael Kaess. Factor Graphs for Robot Perception. *Foundations and Trends in Robotics*, 6(1-2):1–139, 2017.

[41] Ernst D. Dickmanns and Birger D. Mysliwetz. Recursive 3-d road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):199–213, 1992.

[42] Ernst D. Dickmanns and Alfred Zapp. A curvature-based scheme for improving road vehicle guidance by computer vision. In *Mobile Robots I*, volume 727, pages 161–169. International Society for Optics and Photonics, 1987.

[43] Ernst D. Dickmanns and Alfred Zapp. Autonomous High Speed Road Vehicle Guidance by Computer Vision. *IFAC Proceedings Volumes*, 20(5):221–226, 1987.

[44] Arnaud Doucet, Nando De Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Statistics for engineering and information science. Springer, New York, 2001.

[45] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, pages 197–208, 2000.

[46] Christian Duchow. *Videobasierte Wahrnehmung markierter Kreuzungen mit lokalem Markierungstest und Bayes' scher Modellierung*, volume 16. KIT Scientific Publishing, 2011.

[47] Marius Dupuis and Han Grezlikowski. OpenDRIVE® - an open standard for the description of roads in driving simulations. In *Proceedings of the Driving Simulation Conference*, pages 25–36, 2006.

[48] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[49] G. David Forney Jr. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47(2):520–548, 2001.

[50] Thomas E. Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *IEEE Journal of Oceanic Engineering*, 8(3):173–184, 1983.

[51] Christian Frese. *Planung kooperativer Fahrmanöver für kognitive Automobile*. PhD thesis, 2012.

[52] Andreas Geiger. *Probabilistic models for 3D urban scene understanding from movable platforms*. PhD thesis, 2013.

[53] Andreas Geiger, Martin Lauer, Christian Wojek, Christoph Stiller, and Raquel Urtasun. 3D Traffic Scene Understanding from Movable Platforms. *IEEE transactions on pattern analysis and machine intelligence*, pages 1–14, sep 2013.

[54] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. 2007.

[55] Zoubin Ghahramani and Geoffrey E. Hinton. Variational Learning for Switching State-Space Models. *Neural Computation*, 2000.

[56] Tobias Gindele. Learning Behavior Models for Interpreting and Predicting Traffic Situations. page 195, 2014.

[57] Tobias Gindele, Sebastian Brechtel, and Rüdiger Dillmann. A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1625–1631, 2010.

[58] Tobias Gindele, Sebastian Brechtel, and Rüdiger Dillmann. Learning Context Sensitive Behavior Models from Observations for Predicting Traffic Situations. In *16th International Conference on Intelligent Transportation Systems*, pages 1764–1771, 2013.

[59] Regine Graf, Hendrik Deusch, Florian Seeliger, Martin Fritzsche, and Klaus Dietmayer. A learning concept for behavior prediction at intersections. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 939–945, 2014.

[60] E Grafarend. The optimal universal transverse Mercator projection. In *Geodetic Theory Today*, page 51. Springer, 1995.

[61] Benjamin H. Groh, Martin Friedl, Andre G. Linarth, and Elli Angelopoulou. Advanced Real-time Indoor Parking Localization based on Semi-Static Objects. *International Conference on Information Fusion*, pages 1–7, 2014.

[62] Toni Heidenreich, Jens Spehr, and Christoph Stiller. LaneSLAM - Simultaneous Pose and Lane Estimation Using Maps With Lane-Level Accuracy. *18th International Conference on Intelligent Transportation Systems*, pages 2512–2517, 2015.

[63] Brian Henderson-Sellers. *A book of object-oriented knowledge: an introduction to object-oriented software engineering*. Prentice-Hall, Inc., 1996.

[64] Catherine Howard. *Knowledge representation and reasoning for a model-based approach to higher level information fusion*. PhD thesis, 2010.

[65] Catherine Howard and Markus Stumptner. A Survey of Directed Entity-Relation–Based First-Order Probabilistic Languages. *ACM Computing Surveys*, 47(1):1–40, 2014.

[66] Marco Huber. *Nonlinear Gaussian Filtering : Theory, Algorithms, and Applications*. 2014.

[67] Junhwa Hur, Seung-Nam Kang, and Seung-Woo Seo. Multi-lane detection in urban driving environments using conditional random fields. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 1297–1302. IEEE, 2013.

[68] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[69] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *ASME, Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[70] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, 2008.

[71] Stefan Klingelschmitt, Matthias Platho, Volker Willert, Julian Eggert, H.-M. Gross, Volker Willert, and Julian Eggert. Combining behavior and situation information for reliably estimating multiple intentions. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 388–393, 2014.

[72] Jörn Knaup and Kai Homeier. RoadGraph - Graph based environmental modelling and function independent situation analysis for driver assistance systems. *13th International IEEE Conference on Intelligent Transportation Systems*, pages 428–432, 2010.

[73] Ralf Kohlhaas, Thomas Bittner, Thomas Schamm, and J. Marius Zöllner. Semantic state space for high-level maneuver planning in structured traffic scenes. In *17th International Conference on Intelligent Transportation Systems*, pages 1060–1065, 2014.

[74] Ralf Kohlhaas, Thomas Schamm, Dominik Lenk, and J. Marius Zöllner. Towards driving autonomously: Autonomous cruise control in urban environments. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 116–121, 2013.

[75] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[76] Daphne Koller and Avi Pfeffer. Object-Oriented Bayesian Networks. In *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, pages 302–313, 1997.

[77] Sascha Korl. *A factor graph approach to signal modelling, system identification and filtering*. PhD thesis, 2005.

[78] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5:128–138, 2012.

[79] Frank Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[80] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A General Framework for Graph Optimization. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[81] Henning Lategahn and Christoph Stiller. Vision-Only Localization. 15(3):1246–1257, 2014.

[82] S. Lefèvre, C. Laugier, and J. Ibañez-Guzmán. Risk assessment at road intersections: Comparing intention and expectation. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 165–171, 2012.

[83] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-Based Precision Vehicle Localization in Urban Environments. *Robotics: Science and Systems III*, pages 121–128, 2008.

[84] Jesse Levinson and Sebastian Thrun. Robust vehicle localization in urban environments using probabilistic maps. *2010 IEEE International Conference on Robotics and Automation*, pages 4372–4378, 2010.

[85] Hans-Andrea Loeliger. Least Squares and Kalman Filtering on Forney Graphs. *Codes, Graphs, and Systems*, pages 113–135, 2002.

[86] Hans-Andrea Loeliger. An Introduction to Factor Graphs. *IEEE Signal Processing Magazine*, 21(January):28–41, 2004.

[87] Hans Andrea Loeliger, Justin Dauwels, Junli Hu, Sascha Korl, Li Ping, and Frank R. Kschischang. The factor graph approach to model-based signal processing. *Proceedings of the IEEE*, 95(6):1295–1322, 2007.

[88] Hanspeter A. Mallot, Heinrich H. Bülthoff, J. J. Little, and Stefan Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, 64(3):177–185, 1991.

[89] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo Decoding as an Instance of Pearl ' s " Belief Propagation " Algorithm. 16(2):140–152, 1998.

[90] Florian Meyer, Thomas Kropfreiter, Jason L. Williams, Roslyn A. Lau, Franz Hlawatsch, Paolo Braca, and Moe Z. Win. Message Passing Algorithms for Scalable Multitarget Tracking. *Proceedings of the IEEE*, 106(2):221–259, 2018.

[91] Kazuma Minoura and Toyohide Watanabe. Driving Support by Estimating Vehicle Behavior. *International Conference on Pattern Recognition*, pages 1144–1147, 2012.

[92] M. Minsky. A framework for representing knowledge. *The Psychology of Computer Vision*, 1975.

[93] Joris M. Mooij. libDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models. *Journal of Machine Learning Research*, 11:2169–2173, 2010.

[94] R. Munoz-Salinas and S. Garrido-Jurado. Aruco library. *URL: http://sourceforge.net/projects/aruco*.

[95] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, 2002.

[96] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, Mass. [u.a.], 2012.

[97] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[98] Xenia Naidenova. *Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems*. IGI Global, 2012.

[99] Sergiu Nedevschi, Voichita Popescu, Radu Danescu, Tiberiu Marita, and Florin Oniga. Accurate ego-vehicle global localization at intersections through alignment of visual data with digital map. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):673–687, 2013.

[100] Jan Oberländer, Sebastian Klemm, Marc Essinger, Arne Roennau, Thomas Schamm, J. Marius Zöllner, and Rüdiger Dillmann. A semantic approach to sensor-independent vehicle localization. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 1351–1357. IEEE, 2014.

[101] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1 edition, sep 1988.

[102] A. U. Peker, O. Tosun, and T. Acarman. Particle filter vehicle localization and map-matching using map topology. *IEEE Intelligent Vehicles Symposium (IV)*, pages 248–253, 2011.

[103] Dominik Petrich, Thao Dang, Gabi Breuel, and Christoph Stiller. Assessing Map-Based Maneuver Hypotheses using Probabilistic Methods and Evidence Theory. In *17th International Conference on Intelligent Transportation Systems*, pages 995–1002, 2014.

[104] Avrom J. Pfeffer. Probabilistic Reasoning for Complex Systems. Technical report, 2000.

[105] Matthias Platho, Horst-Michael Gross, and Julian Eggert. Traffic situation assessment by recognizing interrelated road users. In *15th International Conference on Intelligent Transportation Systems*, pages 1339–1344, 2012.

[106] Matthias Platho, Horst-Michael Gross, and Julian Eggert. Predicting Velocity Profiles of Road Users at Intersections Using Configurations. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 945–951, 2013.

[107] Dean Pomerleau. RALPH: Rapidly adapting lateral position handler. In *Intelligent Vehicles' 95 Symposium., Proceedings of the*, pages 506–511. IEEE, 1995.

[108] James Power. Notes on Formal Language Theory and Parsing. *National University of Ireland, Maynooth, Kildare*, 2002.

[109] Stefano Crespi Reghizzi, Luca Breveglieri, and Angelo Morzenti. *Formal languages and compilation*. Springer, 2013.

[110] Christoph Reller. *State-space methods in statistical signal processing: New ideas and applications*. PhD thesis, 2013.

[111] Mary Beth Rosson and Sherman R. Alpert. The cognitive consequences of object-oriented design. *Human-Computer Interaction*, 5(4):345–379, 1990.

[112] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2010.

[113] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

[114] Thomas Schamm. *Modellbasierter Ansatz zur probabilistischen Interpretation von Fahrsituationen*. PhD thesis, 2014.

[115] Thomas Schamm and J. Marius Zöllner. A model-based approach to probabilistic situation assessment for driver assistance systems. In *14th International IEEE Conference on Intelligent Transportation Systems*, pages 1404–1409, 2011.

[116] Ullrich Scheunert, Heiko Cramer, and Gerd Wanielik. Precise vehicle localization using multiple sensors and natural landmarks. *Proceedings of the Seventh International Conference on Information Fusion*, pages 649–656, 2004.

[117] Andreas Schindler. Vehicle self-localization with high-precision digital maps. *IEEE Intelligent Vehicles Symposium (IV)*, pages 141–146, 2013.

[118] Matthias Schreier. *Bayesian Environment Representation , Prediction , and Criticality Assessment for Driver Assistance Systems*. PhD thesis, 2015.

[119] Michael Lee Scott. *Programming language pragmatics*. Morgan Kaufmann, 2000.

[120] Christophe Simon and Philippe Weber. Bayesian Networks Implementation of the Dempster-Shafer Theory to Model Reliability Uncertainty. *ARES'06, 1st International Conference on Availability, Reliability and Security*, pages 788–793, 2006.

[121] Jens Spehr. *On Hierarchical Models for Visual Recognition and Learning of Objects, Scenes, and Activities*, volume 11. 2015.

[122] John J. Sudano. Inverse pignistic probability transforms. *Proceedings of the 5th International Conference on Information Fusion, FUSION 2002*, 2:763–768, 2002.

[123] Erik B. Sudderth. *Graphical Models for Visual Object Recognition and Tracking*. PhD thesis, 2006.

[124] Daniel Töpfer. *On Compositional Hierarchical Models for holistic Lane and Road Perception in Intelligent Vehicles*. PhD thesis, 2014.

[125] Daniel Töpfer, Jens Spehr, Jan Effertz, and Christoph Stiller. Efficient scene understanding for intelligent vehicles using a part-based road representation. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, (Itsc):65–70, 2013.

[126] Daniel Töpfer, Jens Spehr, Jan Effertz, and Christoph Stiller. Efficient Road Scene Understanding for Intelligent Vehicles Using Compositional Hierarchical Models. 16(1):441–451, 2015.

[127] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E*, 62(2):1805–1824, aug 2000.

[128] M. Tsogas, Xun Dai, G. Thomaidis, P. Lytrivis, and A. Amditis. Detection of maneuvers using evidence theory. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 126–131, 2008.

[129] Lijun Wei. *Multi-sources fusion based vehicle localization in urban environments under a loosely coupled probabilistic framework*. PhD thesis, 2013.

[130] Niclas Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, 1996.

[131] W Wiegerinck and T Heskes. Fractional belief propagation. In *Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference*, volume 15, page 455. The MIT Press, 2003.

[132] Jürgen Wiest, Matthias Hoffken, Ulrich Kreßel, and Klaus Dietmayer. Probabilistic trajectory prediction with Gaussian mixture models. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 141–146, 2012.

[133] Jian Wu, Zhi-ming Cui, Peng-peng Zhao, and Jian-ming Chen. Traffic vehicle behavior prediction using hidden markov models. *Artificial Intelligence and Computational Intelligence*, pages 383–390, 2012.

[134] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing Free Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory 51*, 2005.

[135] Nour Zalmai. *A State Space World for Detecting and Estimating Events and Learning Sparse Signal Decompositions*. PhD thesis, 2017.

[136] Jianwei Zhang and Bernd Roessler. Situation analysis and adaptive risk assessment for intersection safety systems in advanced assisted driving. In *Autonome Mobile Systeme*, pages 249–258. Springer, 2009.

[137] Long Zhu, Yuanhao Chen, Antonio Torralba, William Freeman, and Alan Yuille. Part and appearance sharing: Recursive compositional models for multi-view multi-object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1919–1926, 2010.