Universität Passau
Fakultät für Informatik und Mathematik

FORWISS – Institut für Softwaresysteme
in technischen Anwendungen der Informatik

Masterarbeit

Modellierung des Fahrzeugumfelds mit Occupancy Grids

Christian Pieringer

Erstprüfer:	Prof. Dr. Tomas Sauer
Zweitprüfer:	Prof. Dr. Ilia Polian
Abgabedatum:	13. August 2013

Zusammenfassung

Erfassung und Modellierung des Fahrzeugumfelds bilden einen wesentlichen Bestandteil von Fahrerassistenzsystemen. Aktuell werden zur Umfeldmodellierung hauptsächlich objektbasierte Modelle eingesetzt – einer der Hauptgründe dafür, wieso viele aktuelle Fahrerassistenzsysteme auf den außerstädtischen Bereich beschränkt sind. Zukünftige Fahrerassistenzsysteme benötigen eine ganzheitlichere Beschreibung, die auch freien Raum explizit modelliert und auf den Daten mehrerer Sensoren basiert. Ein kartenbasiertes Umfeldmodell, insbesondere in Form eines so genannten *Occupancy Grids*, stellt eine vielversprechende Möglichkeit dafür dar.

Unter einem Occupancy Grid (OG) versteht man ein zweidimensionales, in der Bodenebene liegendes Gitter, das in jeder Zelle die Wahrscheinlichkeit dafür enthält, dass dieser Bereich des Umfelds durch ein statisches Hindernis belegt ist. In der Literatur beschriebene Ansätze zur Erstellung eines Occupancy Grids als kartenbasiertes Fahrzeugumfeldmodell sind typischerweise maßgeschneidert auf ein konkretes Einsatzszenario, insbesondere aus Gründen der Einfachheit und Effizienz. Dies beschränkt allerdings deren Wiederverwendbarkeit.

Diese Arbeit bietet mit der *OG-Bibliothek* eine solide und anpassbare Grundlage zur Erstellung automobiler Occupancy Grids. Die OG-Bibliothek kann in ganz verschiedenen Szenarien zum Einsatz kommen und ermöglicht insbesondere den gleichberechtigten Einsatz beliebiger, unterschiedlicher Sensoren. Dies wird durch die Realisierung einer Reihe neuer Ideen möglich; dazu gehören vor allem die anwendungsunabhängige und leicht anpassbare Architektur, das aus mehreren Ebenen bestehende Messwertgrid zur Integration heterogener Sensorik, die Speicherung eines Dynamikwerts je Zelle zur Filterung oder Visualisierung dynamischer Objekte, die Verwendung von Konfliktwerten zum Umgang mit Widersprüchen und die effiziente Positionsmodellierung bestehend aus Zell-Position und In-Zell-Position. Wie eine Anpassung der OG-Bibliothek konkret aussieht, wird anhand eines beispielhaften Anwendungsszenarios unter Verwendung eines Laser- und eines Radarsensors, der Dempster-Shafer-Theorie und des Frameworks *ADTF* demonstriert.

Abstract

Capturing and modeling of the vehicle's surroundings make an important contribution to driver assistance systems. Currently, mainly object-based representations are used for modeling the environment – this is one of the main reasons why many of the current driver assistance systems are used in extra-urban areas only. Future driver assistance systems will need a more holistic approach, explicitly modeling obstacle-free areas and based on data of multiple sensors. As a promising implementation way a map-based environmental model is considered, especially in the form of the so-called *occupancy grid*.

An occupancy grid (OG) is defined as a two-dimensional floor-level layered grid, each cell containing the posterior probability that this part of the environment is occupied by a static obstacle. Typically, approaches found in the literature to build occupancy grids as a map-based model of the vehicle environment are tailored to a specific operational scenario, mainly for the reasons of simplicity and efficiency. This, however, limits their reusability.

With the *OG library*, this paper provides a solid and adaptable basic method for applying occupancy grids to the automotive sector. The OG library is highly adjustable and allows the parallel and equal terms integration of arbitrary additional and different sensor types, and can, therefore, be easily used in quite different scenarios. This is made possible by introducing several new ideas, namely the application-independent and easily adaptable architecture, the multi-level measurement grid for integration of heterogeneous sensors, storage of one dynamic value per cell for filtering or visualization of dynamic objects, handling of discrepancies by means of conflict values, and an effective position model consisting of cell position and in-cell position. An exemplary application of the OG library is demonstrated in a scenario using a laser sensor and a radar sensor, applying the Dempster-Shafer theory and the *ADTF* framework.

Danksagung

An dieser Stelle möchte ich meinen beiden Prüfern Prof. Dr. Tomas Sauer und Prof. Dr. Ilia Polian für die Betreuung dieser Arbeit danken. Beide haben mir zudem wertvolle Anmerkungen im Rahmen der Präsentation dieser Arbeit geliefert, die auch in diese Ausarbeitung eingeflossen sind. Prof. Dr. Sauer stand darüber hinaus nicht nur für all meine Fragen zur Verfügung, sondern hat durch etliche Anregungen und konstruktive Kritik auch zur endgültigen Form dieser Arbeit beigetragen.

Mein Dank gilt auch allen Mitarbeitern des FORWISS-Instituts, die mir bei Fragen immer gerne weitergeholfen haben. Insbesondere gilt dies für Eva Lang und Florian Janda, die von Beginn an meine Arbeit begleitet haben, stets ein offenes Ohr für mich hatten und mir immer mit wertvollen Einschätzungen und Tipps zur Seite standen.

Nicht zuletzt möchte ich mich bei meiner Freundin Eva und meiner Familie bedanken, die mir meinen Rücken freihielten und kräftigten. Danke für eure Liebe, Motivation und Unterstützung – nicht nur in den letzten Monaten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel soll die in dieser Arbeit behandelte Thematik motiviert und eingeführt werden. Zudem wird ein Überblick über die Literatur gegeben und es werden Ausgangssituation, Zielstellung und Aufbau vorliegender Arbeit skizziert.

1.1 Motivation

„Fahren ohne Fahrer", überschreibt *DER SPIEGEL* einen Artikel [56] und fragt: „Schon bald werden Roboterautos auf den Straßen fahren, die Passagiere chauffieren. [...] Lassen sich so Millionen Verkehrstote verhindern?"

Die Automobilhersteller zeigen in den letzten Jahren jedenfalls, wozu ihre Autos fähig sind [56]. Fünf Volvos fuhren Kolonne, wobei nur der erste von einem Menschen gesteuert wurde. Ein autonom fahrender BMW fuhr von München nach Nürnberg, ein autonom fahrender Audi auf einigen amerikanischen Rennstrecken. Ein neues Modell von Mercedes kann im Stau bei Schrittgeschwindigkeit selbständig Gasgeben, Bremsen und Lenken. Am bekanntesten ist aber die autonome Fahrzeugflotte von Google, die immer wieder publikumswirksam in Szene gesetzt wird und bereits in einigen US-Bundesstaaten die Straßenzulassung besitzt.

Die Idee autonom fahrender Fahrzeuge ist alt, sie galt aber lange Zeit als Utopie und Science-Fiction. Obige Beispiele zeigen, dass sie nun jedoch bald Realität werden könnte – nur noch gut fünf bis zehn Jahre werden bis dahin vergehen, schätzen Fachleute [25][56]. Dabei erfolgt der Übergang zu autonomen Fahrzeugen nicht abrupt, sondern inkrementell [25], indem immer mehr und immer bessere Assistenzsysteme immer komplexere Aufgaben übernehmen.

Um solche fortschrittlichen Fahrerassistenzsysteme zu ermöglichen, ist eine maschinelle Wahrnehmung notwendig, die die Umgebung erfasst und modelliert. Die Erfassung erfolgt durch Sensorik, wobei es oftmals sinnvoll ist, die Daten unterschiedlicher Sensoren zu kombinieren. Dies betrifft insbesondere sicherheitskritische Assistenzsysteme, die hohe Anforderungen bezüglich Verlässlichkeit, Zuverlässigkeit und Robustheit erfüllen sollen – welche mit einem einzelnen Sensor typischerweise nicht erreichbar sind. Die anschließende Modellierung überführt die erfassten Daten in eine interne Repräsentation der Umgebung, das so genannte *Umfeldmodell*, wobei je nach Assistenzfunktion ein unterschiedlich komplexes Modell erforderlich ist.

Heutige, serienmäßige Assistenzsysteme verwenden meist ein objektbasiertes Umfeldmodell, das die Umgebung als Liste von Objekten mit gewissen Eigenschaften modelliert. Dieses Modell ist einfach und effizient, verhindert jedoch den innerstädtischen Einsatz, da die Umgebung dort so komplex ist, dass sie nicht mehr die Annahmen der simplen Objektmodelle erfüllt. Aus diesem Grund ist für fortschrittliche Fahrerassistenzsysteme ein verbessertes Umfeldmodell notwendig, das eine umfassendere Beschreibung der Umgebung liefert, die sich nicht auf einige einfache Objekte beschränkt und auch freien Raum explizit modelliert.

Genau hier liegt die Stärke kartenbasierter Umfeldmodelle, die deshalb für künftige Fahrerassistenzsysteme eine sehr interessante alternative oder zusätzliche Möglichkeit zur Modellierung darstellen. Einer der erfolgreichsten und bekanntesten Vertreter ist das so genannte *Occupancy Grid* (OG).

1.2 Einführung

Das klassische Occupancy Grid wurde für den Einsatz mit einem Roboter konzipiert, der sein Umfeld mit einem Sonar abtastet und sich in einer statischen Umgebung fortbewegt. *Statisch* heißt, dass die Umgebung sich über die Zeit nicht verändert, da darin – mit Ausnahme des eigenen Roboters – keine *dynamischen*, also sich fortbewegende Objekte vorhanden sind.

Das Occupancy Grid ist eine spezielle Karte, die dem Roboter als Umfeldmodell dient. Es modelliert den Bereich um den Roboter als zweidimensionale, in der Bodenebene der Umwelt liegende Gitterstruktur. Alle Zellen dieser Gitterstruktur besitzen dieselbe, feste Größe und werden als binäre Zufallsvariablen modelliert, die jeweils die Zustände *belegt* oder *frei* annehmen können. Eine Zelle gilt genau dann als *belegt*, wenn sich an dem von der Zelle abgedeckten Bereich ein (statisches[1]) Hindernis befindet, und andernfalls als *frei*. Je Zelle wird die Wahrscheinlichkeit für den Zustand *belegt* gespeichert; im Folgenden oftmals bezeichnet als *Belegtheitswert*. Diese Wahrscheinlichkeit legt zugleich die Wahrscheinlichkeit für den Zustand *frei* fest, da dieser das Gegenereignis darstellt und deshalb beide Werte zu eins summieren.

Die Modellierung des Umfelds findet statt, während sich der Roboter durch die Umgebung bewegt. Die aktuelle *Pose* des Roboters, das heißt seine Position und Orientierung, wird als *Eigenpose* bezeichnet und ist bekannt. Während sich der Roboter fortbewegt, liefert seine Sensorik kontinuierlich Messwerte, die Aufschluss über die Positionen von Hindernissen in der Umgebung liefern, aber mit Messunsicherheiten und -fehlern behaftet sind. Für jeden Messwert werden durch ein so genanntes *inverses Sensormodell* die betroffenen Zellen ermittelt und für jede dieser Zellen ein Belegtheitswert auf Basis des Messwerts kalkuliert und gespeichert. Dabei erfolgt eine Kombination mit dem bisherigen Wert, wodurch die Messunsicherheiten und -fehler reduziert werden, da dasselbe Hindernis typischerweise mehrfach erfasst und kartiert wird.

Das klassische *Occupancy Grid Mapping* (OGM) – der Algorithmus zur Erzeugung von Occupancy Grids – trifft also zwei Annahmen: die Welt ist statisch und die Eigenpose bekannt. Diese Arbeit behandelt die Erstellung von Occupancy Grids im automobilen Umfeld. Hier sind diese beiden Annahmen typischerweise (zunächst) nicht erfüllt, weshalb im

[1] Dynamische Hindernisse existieren nicht, da ein statisches Umfeld angenommen wird.

Vergleich zum klassischen Ansatz Erweiterungen notwendig sind. Diese dienen der bestmöglichen Einhaltung der Annahmen, indem sie eine Schätzung der Eigenpose durchführen und Messwerte, die durch dynamische Objekte verursacht wurden (z.B. andere Fahrzeuge, Fußgänger oder Radfahrer), geeignet behandeln. Bei der konkreten Umsetzung unterscheiden sich bestehende Ansätze, insbesondere bezüglich des Umgangs mit dynamischen Objekten.

1.3 Literatur

Das klassische Occupancy Grid Mapping stammt aus der Robotik der späten 1980er-Jahre und wurde ursprünglich mit Sonaren verwendet [37]. Es geht zurück auf die Ideen von Alberto Elfes [19], dessen Doktorarbeit das Gebiet begründete [18], und Hans Peter Moravec [38]. Eine wichtige darauf aufbauende Arbeit stammt von Konolige [29]. Dort werden die ursprünglichen Arbeiten kritisch gewürdigt und um zwei wesentliche Punkte ergänzt: Die Behandlung redundanter und durch Reflexionen entstandener Messwerte.

Damals blieb der Ansatz aufgrund seiner Zeit-, Berechnungs- und Speicheraufwendigkeit wenig genutzt. Mit der Verfügbarkeit leistungsfähigerer Rechner hat sich das Occupancy Grid Mapping jedoch in der Robotik etabliert und wird inzwischen auch im Automobilbereich eingesetzt. Zudem sind etliche Varianten entstanden, so dass der Begriff *Occupancy Grid* inzwischen eine ganze Familie verwandter Ansätze bezeichnet.

Die *klassische* Variante ist nach wie vor stark in der Literatur vertreten [54][52]. Ihre Attraktivität begründet sich in der einfachen rekursiven Form der Aktualisierungsgleichung [50][51]. Sie basiert auf der klassischen Wahrscheinlichkeitstheorie und verwendet den Satz von Bayes. Die zweite wichtige Variante basiert auf der Dempster-Shafer-Theorie, die mit Evidenzen anstelle klassischer Wahrscheinlichkeiten arbeitet und deshalb auch als *evidenztheoretisch* bezeichnet wird. Anders als in der klassischen Variante ist dadurch die Unterscheidung zwischen fehlendem und unsicherem Wissen möglich und ein Maß zur Quantifizierung von widersprüchlicher Information vorhanden [39][15].

Ferner gibt es auf Fuzzy-Logik basierende Ansätze (z.B. [40]), die in der aktuellen Literatur jedoch kaum noch vorhanden sind [28], und histogrammbasierte Ansätze (*Histogrammic in Motion Mapping*, HIMM), die eine Spezialisierung auf die für uns nicht relevante Sonarsensorik darstellen [39]. Diese beiden Varianten werden im Rahmen vorliegender Arbeit deshalb nicht betrachtet.

Die am häufigsten genutzten Einführungen in das Occupancy Grid Mapping stammen von Sebastian Thrun [50][51] und von Robin R. Murphy [39]. Thrun verwendet den binären Bayes-Filter alternativ in der so genannten *Odds-* und *Log-Odds-Darstellung*, während Murphy zum Einen den binären Bayes-Filter in seiner ursprünglichen Form verwendet und zum Anderen eine auf der Dempster-Shafer-Theorie basierende Variante vorstellt. Die Einführungen beider Autoren stammen aus der Robotik, bilden aber auch die Grundlage vieler automobiler Arbeiten. Die wichtigste Konsequenz ihrer Herkunft ist, dass sie von einem statischen Umfeld und einer gegebenen exakten Eigenpose ausgehen, was beides meist nicht der Situation im Automobilbereich entspricht; entsprechend sind dort Erweiterungen notwendig, die in unterschiedlichen Arbeiten ganz unterschiedlich aussehen können.

Viele relevante Arbeiten aus dem automobilen Umfeld stammen vom Forscherteam um Klaus Dietmayer. [54] überträgt den auf dem binären Bayes-Filter in Odds-Darstellung basierenden Ansatz von Thrun [50] auf den automobilen Einsatz. Wesentlich sind dabei die

Einführung eines zweistufigen und echtzeitfähigen inversen Sensormodells für einen Lasersensor sowie eines separaten Messwertgrids. Dynamische Objekte werden zunächst kartiert und erst nachträglich entfernt. [33] erweitert den Ansatz um ein komplexeres Sensormodell und verwendet eine objektbasierte Dynamikerkennung, wobei dynamische Objekte durch Segmentierung eines Differenzgrids (d.h. einem Grid bestehend aus den Unterschieden zwischen aktueller und vorheriger Belegung) erkannt und mithilfe eines Multi-Objekt-Kalman-Filters getrackt werden. Erkannte dynamische Objekte werden aus der Kartendarstellung entfernt. In [31] wird daneben mit dem „verzögerten Kartografieren" eine weitere Möglichkeit zum Umgang mit dynamischen Objekten beschrieben: Objekte werden erst kartiert, wenn sie mindestens zweimal erfasst wurden.

Mit [34] kommt die Möglichkeit hinzu, die aufgebaute Karte persistent zu speichern und später wieder zu laden. Dies erlaubt die Erstellung globaler Karten. Zudem wird das so genannte „Generic Grid Mapping" eingeführt, das das Speichern beliebiger Zellinhalte mit beliebigen Fusionsoperationen erlaubt (nicht mehr nur Belegtheitswahrscheinlichkeiten mittels binären Bayes-Filter). Dies ermöglicht beispielsweise den Einsatz der Dempster-Shafer-Theorie und die Erstellung anderer Karten als Occupancy Grids (z.B. Intensitätskarten auf Basis eines bildgebenden Radars). In [30] wird entsprechend die Dempster-Shafer-Theorie eingesetzt; in [32] werden weitere Beispiele gegeben.

[49] verwendet statt einem Lasersensor einen Radarsensor. Zudem wird digitales Kartenmaterial einbezogen, um den Straßenverlauf vor dem Auto zu bestimmen und dadurch die Genauigkeit des Occupancy Grids in hohen Reichweiten zu verbessern.

Eine Arbeit von BMW [26] übernimmt viele Punkte von [54]. Es werden ein Lasersensor, ein binärer Bayes-Filter in Odds-Darstellung und ein separates Messwertgrid verwendet. Neu ist, dass mit Objekt- und Bodenpunkten zwei Objektklassen unterschieden werden, deren Klassifizierung bereits der Lasersensor vornimmt und die in eigenen Grids kartiert werden. Das Boden-Occupancy-Grid enthält dabei auch detektierte Straßenmarkierungen, die in einem gewöhnlichen Occupancy Grid nicht enthalten sind.

Auch eine Forschergruppe von INRIA ist auf dem Gebiet automobiler Occupancy Grids sehr aktiv und hat mehrere Paper veröffentlicht [52][7][53][20][3], die alle dieselbe Architektur zusammen mit einem Lasersensor verwenden. Die Architektur ist zweiteilig und basiert auf einem karten- und einem objektbasierten Ansatz.

Im kartenbasierten Teil werden die Daten unter Verwendung des binären Bayes-Filter in Log-Odds-Darstellung fusioniert und kartiert. Dynamische Objekte werden basierend auf der Idee erkannt, dass sie zu abwechselnder Beobachtung einer Zelle als *frei* und *belegt* führen. Außerdem werden Bereiche, in denen schon mehrmals dynamische Objekte erkannt wurden, unabhängig von der tatsächlichen aktuellen Beobachtung als *dynamisch* klassifiziert. Für die Verwaltung bisher erkannter dynamischer Objekte wird ein zusätzliches Dynamikgrid eingeführt. In [7][53][20] werden ergänzend zum Lasersensor zwei Radarsensoren verwendet, um die dynamischen Objekte zu bestätigen und deren Geschwindigkeiten zu erhalten. Nur in der letztgenannten Arbeit erfolgt zusätzlich eine Kartierung der Radardaten.

Im objektbasierten Teil werden die erkannten Objekte verifiziert und verfolgt. Anders als in den übrigen hier vorgestellten Ansätzen und im klassischen Occupancy Grid Mapping vorgesehen, wird das bisher ermittelte Occupancy Grid zur Korrektur der mit Unsicherheiten behafteten, auf Odometriedaten basierenden Eigenpositionsschätzung verwendet.

In einer weiteren Arbeit von INRIA [57] werden vier auf Laufzeitmessung basierende

Lasersensoren eingesetzt. Es wird erstmalig die Parallelisierung des Occupancy Grid Mappings mittels Grafikprozessor (GPU) thematisiert. Die Grundlage bildet die übliche Annahme, dass die einzelnen Zellen voneinander unabhängig und damit parallel berechnet werden können. Die Kartierung findet zunächst in einem polaren Messwertgrid statt, da auf Laufzeitmessung basierende Sensoren die Messdaten typischerweise im Polarkoordinatensystem liefern und dadurch in einem polaren Grid leichter eingetragen werden können. Außerdem wird der nachteilige *Moiré-Effekt*[2] vermieden, der beim Standard-Occupancy-Grid-Mapping auftreten kann. Die Gitterstruktur des Messwertgrids im Polarkoordinatensystem muss dadurch jedoch abgebildet werden auf die Gitterstruktur der Karte im kartesischen Koordinatensystem, was einen sehr hohen Rechenaufwand bedeutet (sog. *„Map Overlay"*-Problem). Der vorgestellte, auf dem binären Bayes-Filter in Log-Odds-Darstellung basierende GPU-Ansatz erlaubt die schnelle Berechnung einer sehr guten Approximation.

Die Langversion dieser Arbeit [58] ergänzt viele Details, insbesondere zur richtigen Wahl von Sensormodellen und zur Implementierung. [27] von BMW übernimmt die prinzipielle Vorgehensweise, verwendet jedoch einen Laser- und einen Radarsensor sowie den binären Bayes-Filter in Odds-Darstellung.

[48] verwendet den binären Bayes-Filter in Log-Odds-Darstellung zusammen mit einem Rundum-Laserscanner auf dem Dach des Fahrzeugs. Dabei wird je Zelle die maximale ursprüngliche Höhe aller darin kartierten Messwerte bestimmt und Messungen unter einer Mindesthöhe werden verworfen. Die Erkennung dynamischer Objekte basiert auf der Idee, dass dynamische Objekte Messungen in bisher freien Regionen verursachen, wohingegen sich die Messungen statischer Objekte in bereits belegten Bereichen befinden. Es wird dazu die Differenz zwischen der aktuellen Belegung und der Belegung vor einigen Zeitschritten berechnet und mittels Schwellwerten eine Klassifizierung durchgeführt.

Ein hybrider Ansatz von Audi [4][5] verknüpft ein wahlweise zwei- oder dreidimensionales Occupancy Grid eng mit einem objektbasierten Tracking. Dazu wird eine Assoziation zwischen den Objekten und ihren zugehörigen Gridzellen eingeführt, indem das Objektmodell um die Liste der zugehörigen Zellen ergänzt wird. Das objektorientierte Tracking nutzt das Occupancy Grid zur Unterscheidung zwischen statischen und dynamischen Hindernissen; das Occupancy Grid nutzt die getrackten Objekte, um deren Bewegung in der Karte zu kompensieren. Es kommen ein Lidar und ein Fernbereichsradar zum Einsatz, die beide gleichberechtigt kartiert werden. Die Erkennung dynamischer Objekte erfolgt durch Differenzbildung (siehe oben), allerdings nicht mehr nur zwischen Zellpaaren, sondern über einen größeren Bereich, um auch Unsicherheiten einzubeziehen.

Der „Bayesian Occupancy Filter" von INRIA [10][11] möchte ein objektbasiertes Multi-Objekt-Tracking direkt auf die kartenbasierte Darstellung des Occupancy Grids übertragen. Dazu wird jede Zelle um einen Geschwindigkeitswert ergänzt, der Teil eines dynamischen Objektmodells ist. Damit wird die Position der Objekte auf Zellbasis verfolgt, ohne dass ein objektbasierter Ansatz verwendet wird. Allerdings scheitert der praktische Einsatz aufgrund des großen Berechnungsaufwands bereits bei kleinen Grids an den Echtzeitanforderungen und es wird zwingend ein Sensor benötigt, der Geschwindigkeiten messen kann.

In [42] werden ebenfalls die Geschwindigkeiten einzelner Zellen geschätzt und auf Zellbasis mit einem Bewegungsmodell verfolgt. Auch hier ist keine Echtzeitfähigkeit gegeben.

[2]Darunter versteht man Artefakte in der Kartendarstellung, die bei Verwendung von Raytracing (vgl. Abschnitt 3.1.2) entstehen können, da dabei viel weniger Aktualisierungen für eine ferne Zelle als für eine nahe Zelle durchgeführt werden.

Deshalb wird das so genannte „Pyramid Grid" eingeführt: ein Grid, das aus zwei Ebenen mit unterschiedlicher Auflösung besteht. Eine vollständige Berechnung findet nur auf dem niedrig aufgelösten Grid statt, mit dessen Hilfe dann „interessante" Bereiche erkannt werden, die anschließend zur näheren Betrachtung zusätzlich im hoch aufgelösten Grid berechnet werden.

Evidenztheoretische Occupancy Grids – die schon weiter oben kurz zur Sprache kamen – stellen neben dem klassischen Ansatz die wichtigste Variante dar. Sie verwenden die Dempster-Shafer-Theorie anstelle der klassischen Wahrscheinlichkeitstheorie. Der auf einem Mehrebenen-Lidar basierende, prototypische Ansatz von [36] verwendet diese Variante. Statt dem binären Bayes-Filter kommt Dempsters Kombinationsregel zum Einsatz. Es wird ein polares Messwertgrid eingesetzt, das unter Verwendung bilinearer Interpolation zeitlich in das eigentliche, kartesische Grid integriert wird. Die Erkennung dynamischer Objekte basiert auf dem entstehenden Konflikt, wenn sich der aktuelle Wert einer Zelle und der auf Basis einer Messung bestimmte neue Wert widersprechen.

Mit [41] wurde bereits früher ein ähnlicher Einsatz der Dempster-Shafer-Theorie für automobile Occupancy Grids beschrieben. Allerdings wird weder ein polares Messwertgrid eingesetzt noch der Konflikt genutzt; zudem kommt ein Sonar zum Einsatz, das in heutigen Assistenzsystemanwendungen kaum eine Rolle spielt (abgesehen von Parkassistenten).

1.4 Ausgangssituation & Ziel

Ausgangspunkt für diese Arbeit ist die Aufgabe, ein zweidimensionales Occupancy Grid zu erstellen, das das Umfeld des eigenen Fahrzeugs modelliert. Hierzu stehen Daten zur Verfügung, die bereits früher im Rahmen mehrerer Messfahrten mit einem mit Sensorik ausgestatteten Fahrzeug in unterschiedlichen Umgebungen aufgezeichnet wurden. Sie umfassen vorverarbeitete Messwerte eines jeweils in Fahrtrichtung orientierten Mittelbereichsradar- und Lasersensors sowie gefilterte Messwerte der Eigenzustandssensorik. Zusätzlich sind Sensor- und Fahrzeug-Konfigurationsdaten sowie ein Referenzvideo vorhanden, das im Rahmen dieser Arbeit der Visualisierung der Fahrzeugumgebung dient. Außerdem steht das Framework *ADTF* zur Verfügung, das unter anderem zum Abspielen der aufgezeichneten Daten (Playback) im Rahmen einer Echtzeit-Simulation genutzt wird.

In der Literatur beschriebene Ansätze sind typischerweise maßgeschneidert auf ein solches Einsatzszenario, insbesondere aus Gründen der Einfachheit und Effizienz. Dies beschränkt allerdings deren Wiederverwendbarkeit.

Diese Arbeit bietet mit der *OG-Bibliothek* eine solide und anpassbare Grundlage zur Erstellung automobiler Occupancy Grids. Die OG-Bibliothek kann in ganz verschiedenen Szenarien zum Einsatz kommen und ermöglicht insbesondere den gleichberechtigten Einsatz beliebiger, unterschiedlicher Sensoren. Wie eine Anpassung der OG-Bibliothek an ein Anwendungsszenario aussieht, wird anhand der Verwendung von ADTF und der bereitgestellten Daten demonstriert.

1.5 Aufbau

Die vorliegende schriftliche Ausarbeitung beschreibt zum Einen die theoretische Grundlage der OG-Bibliothek, zum Anderen die OG-Bibliothek selbst, mit dem Ziel, deren einfache

und korrekte Benutzung zu ermöglichen. Der weitere Aufbau der Arbeit ist dazu wie folgt.

Dieser Einleitung schließt sich mit Kapitel 2 eine kurze Einführung in die Fahrzeugumfeldwahrnehmung an. Im Rahmen eines Überblicks über die Umfelderfassung werden die dafür typischen Sensortechnologien gegenübergestellt und Multisensorsysteme inklusive der einhergehenden Datenfusion angesprochen. Anschließend werden der objekt- und kartenbasierte Ansatz zur Umfeldmodellierung sowie Anforderungen an künftige Umfeldmodelle im automobilen Umfeld besprochen.

Im darauffolgenden Kapitel 3 wird das klassische Occupancy Grid Mapping beschrieben, das auf der klassischen Wahrscheinlichkeitstheorie basiert. Danach wird mit den evidenztheoretischen Occupancy Grids auf Basis der Dempster-Shafer-Theorie die wichtigste Erweiterung vorgestellt. Abschließend wird speziell auf die Verwendung der Occupancy Grids als automobile Umfeldmodelle eingegangen.

In Kapitel 4 wird der praktische Teil der Arbeit vorgestellt. Zunächst werden die Rahmenbedingungen und Zielstellungen eingehender besprochen. Dann werden die beiden Implementierungen erläutert: Die OG-Bibliothek, die der Erstellung automobiler Occupancy Grids dient, und der OGFilter, der der Integration der OG-Bibliothek in das Framework ADTF dient. Im Rahmen dessen wird auch kurz ADTF vorgestellt.

Das Potential der OG-Bibliothek wird in Kapitel 5 besprochen, indem praktische Ergebnisse gezeigt werden. Dazu wird eine Auswahl der Occupancy Grids betrachtet, welche exemplarisch aus den verfügbaren Daten im Rahmen einer ADTF-Simulation erstellt wurden. Zudem erfolgt ein Vergleich mit anderen Arbeiten und es wird auf die Laufzeit eingegangen.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 6.

Kapitel 2

Erfassung & Modellierung des Fahrzeugumfelds

Bevor im nächsten Kapitel speziell auf die Modellierung des Fahrzeugumfelds mit Occupancy Grids eingegangen wird, werden zunächst in diesem Kapitel grob die allgemeinen Grundlagen der Umfelderfassung und -modellierung besprochen, die für Fahrerassistenzsysteme im Rahmen der Umfeldwahrnehmung eine wichtige Rolle spielen.

2.1 Umfeldwahrnehmung

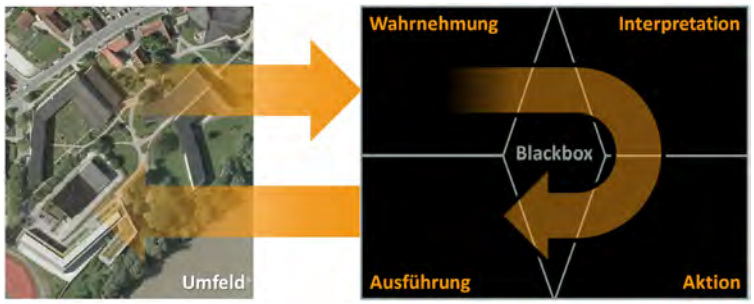

Abbildung 2.1: Die Bestandteile eines typischen Assistenzsystems [Luftbild: Google Maps].

Ein Fahrerassistenzsystem interagiert mit der Umwelt, indem es diese erfasst und darauf einwirkt. Dabei kann es der Erfüllung ganz unterschiedlicher Aufgaben dienen, weshalb es in Abbildung 2.1 als „Blackbox" dargestellt ist. Trotzdem lässt es sich im Allgemeinen in vier Bestandteile untergliedern [17]: Wahrnehmung, Interpretation, Aktion und Ausführung. In der *Wahrnehmungsebene* wird die eigene Position bestimmt und die Situation um das eigene Fahrzeug erfasst und modelliert. Anschließend wird das erstellte Modell in der *Interpretationsebene* gedeutet, das heißt, hinsichtlich der Auswirkungen auf das eigene Verhalten und der Erreichung der eigenen Ziele bewertet. In der nachfolgenden *Aktionsebene* wird auf Basis der Interpretation aus allen Alternativen eine Aktion gewählt, mit der das eigene Verhalten bezüglich der Zielstellung geeignet beeinflusst wird. Die *Ausführungsebene* setzt die gewählte Aktion schließlich um, wodurch eine Rückwirkung auf die Umwelt stattfindet.

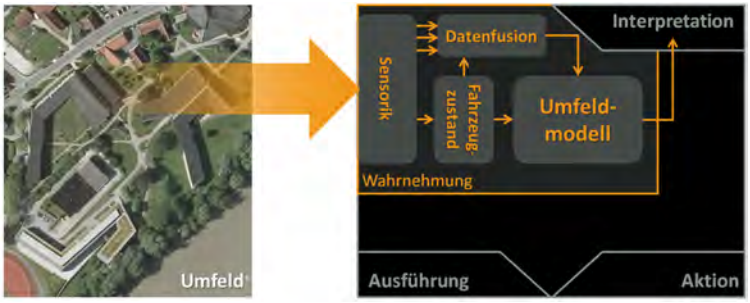

Abbildung 2.2: Die Bestandteile der Wahrnehmungsebene [Luftbild: Google Maps].

Im Rahmen dieser Arbeit wird nur die in Abbildung 2.2 skizzierte Wahrnehmungsebene behandelt, die eine Schlüsselrolle in künftigen Fahrerassistenzsystemen besitzt [22][23]. Ihre Aufgabe ist die Erfassung und Modellierung des Umfelds, wozu sie einzelne Beobachtungen in Wissen über das Umfeld überführt. Zur Erfassung steht dem System Sensorik für zweierlei Aufgaben zur Verfügung:

- Die Sensorik zur *Eigenzustandserfassung* dient dazu, den Zustand des eigenen Fahrzeugs zu bestimmen, wobei insbesondere die eigene Pose eine wichtige Rolle spielt.
- Die Sensorik zur *Umfelderfassung* tastet die Umgebung des eigenen Fahrzeugs ab. In vielen Fällen sind die Daten eines einzelnen Umfeldsensors nicht ausreichend, um die Umwelt geeignet zu beschreiben, weshalb oft mehrere, unterschiedliche Sensoren zum Einsatz kommen.

Die einzelnen Messwerte der Umfeldsensoren, kurz *Umfelddaten*, werden unter Verwendung des Eigenzustands im Rahmen einer *Datenfusion* zu einer Gesamtaussage zusammengeführt. Die erfassten und fusionierten Daten werden schließlich unter erneuter Verwendung des Eigenzustands in das *Umfeldmodell* eingebracht. Es gibt unterschiedliche Arten von Umfeldmodellen, viele führen jedoch eine zeitliche Integration der Daten durch, um dadurch Messunsicherheiten und -fehler zu kompensieren. Im Umfeldmodell soll das Wissen über den wahren Zustand der Umgebung über die Zeit durch die Kombination vieler Messungen gesteigert werden, wobei idealerweise ein Maß für die Unsicherheit verfügbar ist [24]. Das Umfeldmodell ist das Ergebnis der Wahrnehmungsebene und wird der nachfolgenden Interpretationsebene bereitgestellt.

2.2 Umfelderfassung

Im Rahmen der Umfelderfassung wird die Umgebung des eigenen Fahrzeugs mit Sensorik abgetastet. Die dafür im Automobilbereich üblichen Sensoren und deren Funktionsprinzipien werden als hinlänglich bekannt angenommen, so dass im Folgenden nur kurz deren unterschiedliche Charakteristiken gegenübergestellt werden. Für darüber hinausgehende Informationen wird auf die Literatur verwiesen (z.B. [55]). Da eine einzelne Sensortechnologie aufgrund ihrer jeweiligen Schwächen zur Umfeldwahrnehmung oft nicht ausreichend ist, wird anschließend auf Multisensorsysteme und die damit verbundene Datenfusion eingegangen.

2.2.1 Gegenüberstellung typischer Sensortechnologien

In Abbildung 2.3 ist ein so genanntes *Google Driverless Car* gezeigt, ein autonomes Fahrzeug aus der Flotte von Google. Es ist unter anderem mit Front- und Seitenradaren (1a bzw. 1b), 360-Grad-Laserscanner (2) und Kamera (3) ausgestattet. Radarsensoren, Lasersensoren (oft auch als *Lidar* bezeichnet für ***Li**ght **D**etection **A**nd **R**anging*) und Kameras sind auch die Sensortechnologien, die derzeit in der automobilen Umfelderfassung eine wesentliche Rolle spielen. Sie unterscheiden sich hinsichtlich des Messprinzips und damit der detektierten Merkmale.

Abbildung 2.3: Sensorik des autonomen *Google Driverless Car* [Foto: techyyouth.com].

In Tabelle 2.1 werden einige Eigenschaften dieser Sensortechnologien gegenübergestellt. Die Bewertung erfolgt auf Basis einer einzelnen Messung, eine möglicherweise eingesetzte Objektverfolgung wird nicht berücksichtigt. Alle Sensoren werden zudem in Fahrtrichtung ausgerichtet angenommen, Objektbreite und -länge sind entsprechend zu verstehen.

Sensor	Radarsensor	Kamera (mit Bildverarbeitung)	Lasersensor
Reichweite	++	+	++
Öffnungswinkel	-	+	++
Objektabstand	+ (Nahbereich 0)	0	++
Objektgeschwindigkeit	++	nicht messbar	nicht messbar
Objektbreite	nicht messbar	++	++
Objektlänge	nicht messbar	nicht messbar	+ (abhängig v. Aspektwinkel)
Objekthöhe	nicht messbar	+	2D-Lidar: nicht messbar; 3D-Lidar: +

Tabelle 2.1: Gegenüberstellung automobiler Umfeldsensorik nach [47].
(Reichweite und Objekthöhe des Lasersensors wurden aktualisiert (Originalwert *0* bzw. *nicht messbar*).)

2.2.2 Multisensorsysteme & Datenfusion

Fortschrittliche Fahrerassistenzsysteme – insbesondere solche, die sicherheitskritische Aufgaben übernehmen sollen – benötigen eine vollständige und zuverlässige Beschreibung des

Fahrzeugumfelds. Anhand Tabelle 2.1 im vorherigen Abschnitt ist erkennbar, dass dies keine einzelne Sensortechnologie leisten kann. Bereits heute kommen deshalb oftmals mehrere Sensoren gleichzeitig zum Einsatz.

Eine gleichzeitige Verwendung mehrerer Technologien kann die Stärken der unterschiedlichen Messprinzipien gewinnbringend kombinieren und/oder die einzelnen Schwächen reduzieren [55][13]. Aber nicht nur der Einsatz mehrerer Technologien ist lohnend – auch mehrere Sensoren einer Technologie können sinnvoll sein.

Grundsätzlich gilt: Redundante Sensorik beschreibt dieselben Objekte und kann dadurch zu einer erhöhten Genauigkeit und Fehlertoleranz beitragen. Komplementäre Sensorik liefert sich ergänzende Daten; diese können die Auflösung von Mehrdeutigkeiten erlauben und zu mehr Information führen, zum Beispiel aufgrund eines größeren Gesamtsichtbereichs der Sensorik oder aufgrund der Detektion unterschiedlicher Merkmale für dasselbe Objekt. Ferner ist es möglich, dass sich durch die Kombination mehrerer Sensoren die Akquisitionsgeschwindigkeit erhöht und die Gesamtkosten gegenüber einem einzelnen Sensor sinken. Außerdem ergibt sich in manchen Fällen eine gesuchte Eigenschaft erst durch die Kombination unterschiedlicher Sensoren [47] (z.B. benötigt eine Triangulation zur Positionsbestimmung eines Objekts die Entfernungsmessungen zweier Sensoren).

Wenngleich die Fusion der Daten unterschiedlicher Sensoren oft vorteilhaft ist, ist sie auch mit Herausforderungen verbunden. Die Sensorik kann sich hinsichtlich wesentlicher Charakteristiken unterscheiden, was im Rahmen des Fusionsprozesses berücksichtigt werden muss, um das erwünschte Resultat zu erhalten. So können unter anderem Unterschiede bestehen bezüglich der Messfrequenz, der Auflösung, der Darstellung der Messwerte, den Messunsicherheiten und den Messfehlerquellen. Außerdem können die Sensoren widersprüchliche oder übereinstimmende Ergebnisse liefern. In letzterem Fall ist klar, wie das Gesamtergebnis auszusehen hat. Hat man jedoch gleich zuverlässige Sensoren, die sich widersprechen, ist die Entscheidung schwieriger. Gleiches gilt, wenn man an einer allgemeingültigen Vorgehensweise zur Fusion interessiert ist, die nicht von konkreter Sensorik abhängt (die also beispielsweise nicht speziell die Fusion von Kamera- und Radardaten behandelt).

Wie eine gewinnbringende Datenfusion trotz dieser Herausforderungen durchgeführt werden kann, ist zu einem großen Teil abhängig vom verwendeten Umfeldmodell (vgl. nächster Abschnitt). Eine grundlegende Einführung in die Fusion von Umfelddaten bietet [55, S. 237ff]. Die kartenbasierte Datenfusion auf Basis von Occupancy Grids wird ausführlicher in Abschnitt 3.3.2 behandelt.

2.3 Umfeldmodellierung

Ein Umfeldmodell ist eine vereinfachende, interne Repräsentation der Fahrzeugumgebung. Da es die Entscheidungsgrundlage für nachfolgende Verarbeitungsschritte darstellt, kommt ihm eine große Bedeutung zu.

2.3.1 Aktuelle Umfeldmodelle

Die derzeit in der Literatur beschriebenen Umfeldmodelle lassen sich im Wesentlichen in zwei Klassen aufteilen: objektbasierte und kartenbasierte Darstellungen. Diese werden nachfolgend basierend auf [22][24][17][4][50] eingeführt.

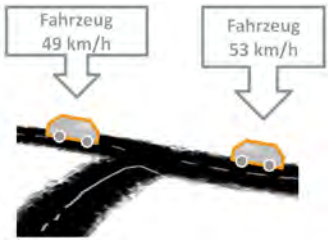

Abbildung 2.4: Objektbasiertes Umfeldmodell: Zwei Objekte der Klasse *Fahrzeug* mit der Eigenschaft *Geschwindigkeit*.

Ein *objektbasiertes Umfeldmodell*, manchmal auch als *merkmalsbasiert* (engl. feature-based) bezeichnet, ist derzeit im Automobilbereich üblich [22]. Es reduziert die Umgebung auf einige wenige Objekte, denen festgelegte Klassen zugeordnet werden. Durch die Klasse wird entschieden, mit welchen Eigenschaften das Objekt beschrieben wird. Für eine Instanz der Fahrzeugklasse kann dies beispielsweise die aktuelle Geschwindigkeit sein (vgl. Abbildung 2.4). Die Eigenschaften werden typischerweise mithilfe üblicher Schätzverfahren getrackt und mit neuen Daten aktualisiert, was eine fortlaufende Assoziierung zwischen Messwerten und Objekten notwendig macht, bei der die Messwerte bestehenden oder neuen Objekten zugeordnet werden.

Außerdem legt die Objektklasse fest, welche geometrische Form für die zugehörigen Instanzen angenommen wird. Aufgrund der Beschreibung durch einzelne Eigenschaften eignet sich das objektbasierte Modell vor allem für Objekte, die durch einfache geometrische Formen approximiert werden können, was hauptsächlich auf dynamische Objekte wie andere Fahrzeuge und Fußgänger zutrifft. Zur Darstellung beliebiger Formen, wie sie beispielsweise in unstrukturierten oder städtischen Gebieten vorkommen, ist es dagegen eher ungeeignet und auch die freien Bereiche zwischen den Objekten werden nicht explizit modelliert. Die Reduktion auf wenige Objekte mit wenigen Eigenschaften erlaubt jedoch eine effiziente Speicherung und eine echtzeitfähige Verwendung.

Abbildung 2.5: Kartenbasiertes Umfeldmodell: Das Occupancy Grid (links) kartiert den statischen Teil der Szenerie (Mitte). Rechts: Überlagerung von Szenerie und Grid.

Ein *kartenbasiertes Umfeldmodell* bietet dagegen eine gleichmäßige und flächige Beschreibung der Umgebung in 2D oder 3D. Messwerte werden entsprechend ihrer Position kartiert, eine Objektbildung und -zuordnung ist nicht notwendig – was eine einfache Modellierung beliebig komplexer Formen ermöglicht. Es werden auch hindernisfreie Bereiche modelliert, man spricht von der so genannten *Freiraummodellierung* oder *Freibereichsmodellierung*. Die Umfeldsensorik liefert im Allgemeinen Messwerte statischer und dynamischer Objekte in der Umgebung. Allerdings sollen in kartenbasierten Umfeldmodellen typischer-

weise nur die Messwerte statischer Objekte enthalten sein, weshalb die Messwerte dynamischer Objekte erkannt und verworfen werden müssen (*Dynamikerkennung* und *-filterung*).

Einige kartenbasierte Umfeldmodelle basieren auf einer diskreten Gitterstruktur und werden deshalb als *gridbasiert* bezeichnet. Jedes Element des Grids, eine so genannte *Zelle*, speichert einen Wert, der sich auf den dadurch repräsentierten Bereich bezieht. Der dadurch relativ hohe Speicher- und Berechnungsaufwand macht eine Abwägung zwischen Genauigkeit (= Zellgröße) und Aufwand notwendig.

Das Occupancy Grid ist ein Beispiel für ein gridbasiertes Umfeldmodell. Hier wird in jeder Zelle die Wahrscheinlichkeit dafür gespeichert, dass der von ihr repräsentierte Bereich durch ein statisches Objekt belegt ist. Ein Beispiel ist in Abbildung 2.5 gezeigt. Das mittlere Bild zeigt die gleiche Szene wie in Abbildung 2.4, dieses Mal aber senkrecht von oben betrachtet. Links daneben sieht man das zugehörige Occupancy Grid, rechts daneben eine Überlagerung der beiden Darstellungen. Zu erkennen ist, dass der verhältnismäßig komplexe Verlauf der Straße gut approximiert wird, die Karte eine flächige Beschreibung des Umfelds inklusive Freiraum liefert und die beiden Autos als dynamische Objekte von der Kartierung ausgenommen wurden.

Manchmal wird auch eine Kombination aus einem objekt- und einem kartenbasierten Umfeldmodell eingesetzt, da sich die beiden gut ergänzen. Dies wird als *hybrides Umfeldmodell* bezeichnet [22][17]. Der objektbasierte Ansatz erlaubt eine effiziente Darstellung dynamischer Hindernisse, solange diese eine einfache Struktur aufweisen; der kartenbasierte Ansatz modelliert gut beliebig komplexe Strukturen, ist jedoch auf statische Hindernisse beschränkt. Eine hybrides Umfeldmodell kombiniert die jeweiligen Stärken und ermöglicht dadurch eine objektbasierte Beschreibung der Dynamik zusammen mit einer kartenbasierten Beschreibung beliebig komplexer Strukturen. Damit die Vorteile beider Ansätze vollständig genutzt werden können, muss zwischen ihnen eine geeignete Schnittstelle zum Austausch von Informationen definiert werden [22].

2.3.2 Künftige Umfeldmodelle

Die Betrachtung der Umfeldmodelle soll mit einem Ausblick auf künftige, fortschrittliche Fahrerassistenzsysteme abgeschlossen werden. Die steigende Bedeutung multisensorieller Ansätze wurde bereits in Abschnitt 2.2.2 hervorgehoben. Darüber hinaus werden für künftige Umfeldmodelle zwei weitere wesentliche Veränderungen erwartet.

Die erste Erwartung ist, dass künftige Fahrerassistenzsysteme ein *hybrides Umfeldmodell* verwenden, also eine Ergänzung des objektbasierten Ansatzes um einen kartenbasierten (vgl. vorheriger Abschnitt). Der Hauptgrund dafür ist, dass die derzeit üblichen objektbasierten Modelle nicht ausreichen, um die komplexe städtische Szenerie abzubilden [17][22], und der Einsatz aktueller Systeme dadurch auf den außerstädtischen Bereich beschränkt ist. Insbesondere der unregelmäßige und uneinheitlich begrenzte Straßenverlauf (z.B. fehlende Markierungen, parkende Autos, komplizierte Kreuzungen, abrupte Veränderungen der Umgebung, ...) ist nicht mehr mit einfachen Objektmodellen darstellbar, aber auch viele Objekte der Straßeninfrastruktur und allgemein Umgebung besitzen unterschiedliche Erscheinungsformen; die Divergenz zwischen Realität und Modellannahmen wird zu groß, um noch eine vernünftige Verwendung zu erlauben. Zudem wird in objektbasierten Umfeldmodellen kein Freiraum modelliert, welcher für etliche Anwendungen (z.B. Wegplanung

für Ausweichmanöver) jedoch eine wichtige Zusatzinformation darstellt. Das zusätzliche kartenbasierte Umfeldmodell eines hybriden Ansatzes schafft in diesen Punkten Abhilfe.

Abbildung 2.6: Ein virtueller Umfeldsensor entkoppelt Sensorik und Aufgaben.

Die zweite Erwartung ist, dass in der Wahrnehmungsebene eine *Entkopplung* zwischen Sensorik und zu erfüllender Assistenzaufgabe stattfinden wird [13][22][2][17]. Bisher besitzt im Allgemeinen jedes Assistenzsystem seine eigene Sensorik. Allerdings halten immer mehr Assistenzsysteme Einzug in die Fahrzeuge [25][13], so dass nicht mehr in gleichem Maße weitere Sensorik eingebaut werden kann. Stattdessen müssen die Assistenzsysteme die vorhandene Sensorik gemeinsam nutzen, was zu einer hohen Vernetzung führt.

Die logische Konsequenz, um die Vernetzung zu reduzieren, ist die Einführung eines *virtuellen Umfeldsensors* [13] (vgl. Abbildung 2.6). Der virtuelle Umfeldsensor fusioniert die Daten aller vorhandenen Umfeldsensoren und integriert das Ergebnis in ein zentrales und allgemeingültiges Umfeldmodell, das möglichst genau ist, um keine Informationen zu verlieren. Auf dieses Umfeldmodell greifen alle Anwendungen zu; die Verarbeitung der Sensordaten ist von den Aufgaben entkoppelt. Eine Rückwirkungsmöglichkeit der Aufgaben auf die Sensoren gewährleistet, dass weiterhin bestehende Abhängigkeiten berücksichtigt werden können; nur die konkrete Aufgabe selbst weiß beispielsweise, welcher Bereich der Umgebung für sie relevant ist, und sie kann deshalb den Erfassungsbereich der Sensorik entsprechend beeinflussen.

Kapitel 3

Occupancy Grids als Umfeldmodell

Im vorherigen Kapitel wurde deutlich, dass für künftige Umfeldmodelle eine kartenbasierte Erweiterung der derzeit üblichen, objektbasierten Repräsentation notwendig ist. Eine Möglichkeit dazu stellen die bekannten und beliebten Occupancy Grids dar. Seit ihrer Einführung sind etliche Varianten entstanden, so dass der Begriff *Occupancy Grid* inzwischen eine ganze Familie verwandter Ansätze bezeichnet [28][39][50].

Im Folgenden werden mit der klassischen und der evidenztheoretischen Variante die beiden wichtigsten Vertreter vorgestellt. Zudem wird besprochen, wie Occupancy Grids im Rahmen eines künftigen Umfeldmodells verwendet werden können.

3.1 Klassische Occupancy Grids

Die klassischen Occupancy Grids stammen aus den späten 1980er-Jahren. Sie gehen zurück auf Ideen von Alberto Elfes [19], dessen Doktorarbeit das Gebiet begründete [18], und Hans Peter Moravec [38]. Ziel des Occupancy Grid Mappings ist die Erstellung konsistenter Karten aus unsicheren und verrauschten Messdaten, wobei die genaue eigene Pose (= Position und Orientierung) als gegeben und das Umfeld als statisch vorausgesetzt wird.

Damals blieb der Ansatz aufgrund seiner Zeit-, Berechnungs- und Speicheraufwendigkeit wenig genutzt. Mit steigender Rechenleistung und der Entwicklung unterschiedlicher Methoden zum Umgang mit dynamischen Objekten (vgl. Abschnitt 3.3.1) stieg jedoch die Verwendung der Occupancy Grids und insbesondere seit den letzten Jahren erfreuen sie sich großer Beliebtheit. Dabei werden sie nicht mehr nur wie ursprünglich in der Robotik mit Sonaren verwendet [37][50], sondern kommen heute auch im Automobilbereich mit den dort üblichen Umfeldsensoren erfolgreich zum Einsatz [5][26][12][16].

Der relativ hohe Speicher- und Berechnungsaufwand stellt jedoch nach wie vor ein Hemmnis für den Serieneinsatz dar und macht eine Abwägung zwischen Genauigkeit (= Zellgröße) und Aufwand notwendig. Insbesondere einige interessante Erweiterungen des klassischen Ansatzes konnten sich deshalb nicht durchsetzen. Dazu gehören die alternative Herangehensweise auf Basis einer Maximum-A-Posteriori-Berechnung mit Vorwärtsmodellen [51][50], die die Nachbarschaftsbeziehungen zwischen einzelnen Zellen berücksichtigt, und die Ansätze [10][11][42], die den Zustandsraum einer einzelnen Zelle zur Handhabung dynamischer Objekte um eine Geschwindigkeit erweitern.

3.1.1 Idee und bayestheoretische Grundlage

Ausgangspunkt des Occupancy Grid Mappings ist eine Umgebung, in der keines der darin vorhandenen Objekte seine Position über die Zeit verändert – außer das eigene Fahrzeug. Dieses bewegt sich in der statischen Umwelt fort und tastet sie dabei mithilfe eines Sensors ab. Auf diese Weise sammelt es Messwerte von unterschiedlichen Orten, die mit Messunsicherheiten und Messfehlern behaftet sind. Das Occupancy Grid Mapping erhält diese unsicheren und verrauschten Messdaten als Eingabe und möchte daraus ein konsistentes Occupancy Grid erzeugen.

Die dem Occupancy Grid Mapping zugrundeliegende Idee ist, eine zeitliche Integration der Messwerte durchzuführen, um die Qualität des Ergebnisses zu erhöhen. In jedem Zeitschritt wird das Occupancy Grid mit dem aktuellen Messwert des Umfeldsensors aktualisiert, wobei eine Kombination des aktuellen mit dem bisherigen Wert der Zelle erfolgt. Dies führt zu einer Reduktion der Messunsicherheiten und -fehler, da dasselbe Hindernis typischerweise mehrfach erfasst und kartiert wird. Betrachtet man das Occupancy Grid über mehrere Zeitschritte hinweg, so wird durch die Eigenbewegung des Fahrzeugs und durch die zeitliche Integration der Messungen mehrerer Zeitpunkte ein konsistentes Occupancy Grid aufgebaut (siehe Abbildung 3.1).

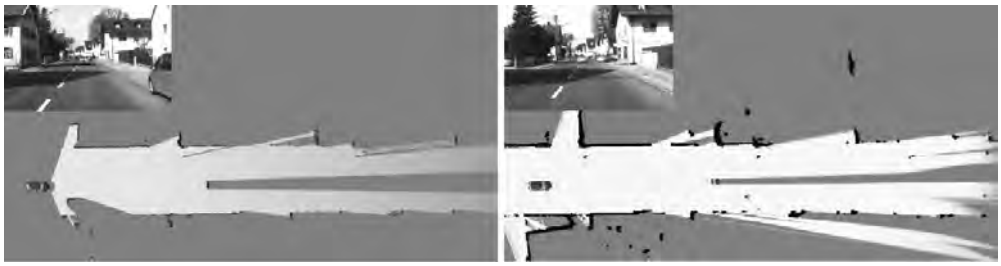

Abbildung 3.1: Ein Occupancy Grid nach einer (links) und nach 30 Aktualisierungen [27].

Ein (klassisches) Occupancy Grid m ist eine gridbasierte, zweidimensionale Datenstruktur, die sich in der Bodenebene des Fahrzeugumfelds befindet und dort einen Bereich M im $\mathbb{R}^2$ in $n \in \mathbb{N}$ Zellen gleicher Größe diskretisiert ($\mathbb{R}$: die Menge der reellen Zahlen; $\mathbb{N}$: die Menge der natürlichen Zahlen ohne Null). Jede dieser Zellen ist eine binäre Zufallsvariable m_i, die einen der beiden Zustände *frei* (F) und *belegt* (B) annehmen kann und dadurch die Belegung der jeweiligen Zelle durch ein statisches Hindernis beschreibt; der Index $i \in [1,n]$ gibt die Position der Zelle im Grid an. Im Allgemeinen wird eine Zelle als *belegt* betrachtet, sobald der von ihr repräsentierte Bereich teilweise belegt ist.

Je Zelle wird im Occupancy Grid die Wahrscheinlichkeit dafür gespeichert, dass ihr Zustand *belegt* lautet. Diese Wahrscheinlichkeit wird für eine Zelle mit Index i als $p(m_i = B)$ oder kürzer $p(m_i)$ notiert und als *Belegtheitswert* bezeichnet. Diese Wahrscheinlichkeit legt auch die Wahrscheinlichkeit für den Zustand *frei* fest, $p(m_i = F)$ oder kürzer $1 - p(m_i)$, da dieser das Gegenereignis darstellt und deshalb beide Werte zu eins summieren.

Die Gesamtheit aller Zellen bildet das Occupancy Grid $m := \{m_i\}_{i=1}^{n}$. Es stellt eine flächige Beschreibung des Umfelds dar, die – anders als ein objektbasiertes Umfeldmodell (vgl. Abschnitt 2.3.1) – keine Objektmodelle benötigt. Außerdem wird, zusätzlich zu den Hindernissen selbst, auch der freie Raum zwischen ihnen explizit modelliert, was als *Freiraummodellierung* oder *Freibereichsmodellierung* bezeichnet wird.

Als Eingabe für das Occupancy Grid Mapping sind die Messungen z_1,...,z_t eines Sensors zu den Zeitpunkten 1 bis t gegeben; dabei bezeichnet die Notation $z_{1:t}$ alle Messungen zwischen und einschließlich der Zeitpunkte 1 und t. Jede Messung besteht aus der Eigenpose, von der aus die Messung erfolgte, und dem eigentlichen Sensormesswert, der Informationen über das Vorhandensein eines statischen Hindernisses an der Messstelle liefert – im Falle vieler Radarsensoren ist das zum Beispiel die relative Position und Geschwindigkeit eines erfassten Objekts. Die Eigenpose wird benötigt, um die Zellen zu ermitteln, die aufgrund des aktuellen Sensormesswert z_t aktualisiert werden müssen.

Als Ausgabe des Occupancy Grid Mappings soll $p(m|z_{1:t})$, die A-Posteriori-Wahrscheinlichkeit des Occupancy Grids, berechnet werden – die Wahrscheinlichkeit des Occupancy Grids unter Kenntnis aller bisheriger Messungen. Diese A-Posteriori-Wahrscheinlichkeit kann aber aufgrund der typischerweise hohen Griddimensionalität im Rahmen einer praktischen Anwendung nicht berechnet werden. Das Problem wird deshalb zerlegt, indem die einzelnen Zellen als voneinander bedingt unabhängig angenommen werden:

$$p(m|z_{1:t}) = \prod_{i=1}^{n} p(m_i|z_{1:t}).$$

Diese Annahme der Zellunabhängigkeit bedeutet, dass man annimmt, dass der Belegtheitszustand einer Zelle nicht von dem ihres Nachbarn abhängt. Obwohl dies offensichtlich eine starke Vereinfachung darstellt, wird diese Annahme im Occupancy Grid Mapping getroffen. Dadurch ist das Problem für jede Zelle einzeln lösbar – für die Zelle mit Index i ist deren A-Posteriori-Wahrscheinlichkeit $p(m_i|z_{1:t})$ unabhängig von den übrigen Zellen berechenbar.

Bei der Durchführung dieser Berechnung zum Zeitpunkt t möchte man den neuen Messwert z_t einbringen, ohne dabei die gesamten bisherigen Messwerte $z_{1:t-1}$ berücksichtigen zu müssen. Dazu kann eine rekursive Aktualisierungsgleichung verwendet werden, die als *binärer Bayes-Filter* bezeichnet wird. Sie soll im Folgenden basierend auf den Darstellungen von Thrun [51][50] hergeleitet werden, die die Grundlage vieler Arbeiten im Automobilbereich bilden – trotz ihres thematischen Bezugs zur Robotik.

Wendet man auf die gesuchte A-Posteriori-Wahrscheinlichkeit einer einzelnen Zelle, $p(m_i|z_{1:t})$, den Satz von Bayes an, ergibt sich

$$p(m_i|z_{1:t}) = \frac{p(z_t|z_{1:t-1},m_i)\, p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}.$$

In der Kartierung ist die Annahme einer statischen Welt üblich, das heißt, alte Sensormesswerte werden für jeden Zeitpunkt t bei gegebenem Occupancy Grid m als bedingt unabhängig voneinander angenommen: $p(z_t|z_{1:t-1},m) = p(z_t|m)$. Das Occupancy Grid Mapping trifft eine stärkere Annahme: $p(z_t|z_{1:t-1},m_i) = p(z_t|m_i)$, das heißt, es wird bedingte Unabhängigkeit der Messwerte angenommen bei gegebener Belegtheit m_i der Zelle mit Index i, unabhängig von der Belegtheit benachbarter Zellen. Diese Annahme entspricht nicht der Realität, da in der Praxis[1] bei einer einzelnen Messung des Sensors mehrere Zellen gleichzeitig abgetastet werden [29][51]. Trotzdem wird diese starke Vereinfachung vom klassischen Ansatz getroffen und obige Gleichung wird zu

[1]Bei der praktischen Verwendung des Occupancy Grid Mappings muss man sich deshalb bewusst sein, dass möglicherweise mehr Zellen als *belegt* kartiert werden, als tatsächlich *belegt* sind.

$$p(m_i|z_{1:t}) = \frac{p(z_t|m_i)\, p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}.$$

Durch erneute Anwendung des Satzes von Bayes, dieses Mal auf $p(z_t|m_i)$, erhält man

$$p(m_i|z_{1:t}) = \frac{p(m_i|z_t)\, p(z_t)\, p(m_i|z_{1:t-1})}{p(m_i)\, p(z_t|z_{1:t-1})}.$$

Diese Gleichung berechnet die Wahrscheinlichkeit dafür, dass die Zelle mit Index i *belegt* ist. Analog kann man die Wahrscheinlichkeit dafür herleiten, dass diese Zelle *frei* ist:

$$1 - p(m_i|z_{1:t}) = \frac{(1 - p(m_i|z_t))\, p(z_t)\, (1 - p(m_i|z_{1:t-1}))}{(1 - p(m_i))\, p(z_t|z_{1:t-1})}.$$

Teilt man die Gleichung der *belegt*-Wahrscheinlichkeit durch den Term der *frei*-Wahrscheinlichkeit, ergibt sich mit dem binären Bayes-Filter die gesuchte Aktualisierungsgleichung:

$$odds(m_i|z_{1:t}) \quad := \quad \frac{p(m_i|z_{1:t})}{1 - p(m_i|z_{1:t})} = \frac{p(m_i|z_t)}{1 - p(m_i|z_t)} \frac{1 - p(m_i)}{p(m_i)} \frac{p(m_i|z_{1:t-1})}{1 - p(m_i|z_{1:t-1})}. \tag{3.1}$$

Sie befindet sich in der *Odds-Darstellung*, die beispielsweise von [54][33][26][27] verwendet wird. Darunter versteht man die Formulierung einer Wahrscheinlichkeit $p(A)$ für ein Ereignis A in der Form $odds(A) := \frac{p(A)}{1-p(A)}$. Der Wertebereich verändert sich von [0,1] zu [0,∞[. Diese Form bietet den Vorteil, dass sich einige schwer zu berechnende Terme kürzen. Nachteilig ist, dass es für Wahrscheinlichkeiten nahe 0 oder 1 zu numerischen Instabilitäten kommen kann [50], die geeignet vermieden werden müssen. Eine Möglichkeit dazu ist die Verwendung einer kleinen Zahl ϵ, mit der der Bereich verwendeter Wahrscheinlichkeiten auf $[\epsilon, 1-\epsilon]$ eingeschränkt wird [54]: Wahrscheinlichkeiten, die kleiner als ϵ sind, werden auf ϵ erhöht; Wahrscheinlichkeiten, die größer als $1-\epsilon$ sind, werden auf $1-\epsilon$ verringert. [54] hat in umfangreichen Experimenten auf diese Weise für $\epsilon := 0{,}00001$ qualitativ gleichwertige Ergebnisse zur Log-Odds-Variante erreicht, bei signifikant niedriger Laufzeit.

Die so genannte *Log-Odds-Darstellung* ist eine alternative Darstellungsmöglichkeit, die zum Beispiel in [52][48] genutzt wird. Diese Form ergibt sich, wenn man die Odds-Darstellung logarithmiert: $l(A) := \log odds(A)$. Der Wertebereich ist $]-\infty,\infty[= \mathbb{R}$. Die Rückgewinnung der eigentlichen Wahrscheinlichkeit ist möglich über den Zusammenhang $p(A) = 1 - \frac{1}{1+e^{l(A)}}$. Bezüglich der Kartenqualität gibt es keine Unterschiede zwischen Log-Odds- und Odds-Darstellung [54]; darüber hinaus sind beide Formen äquivalent zur Darstellung durch gewöhnliche Wahrscheinlichkeiten [39] (hier sind dann jedoch schwer zu berechnende Terme enthalten). Zu den Nachteilen der Log-Odds-Darstellung gehören, dass dadurch die relativ aufwendige Berechnung des Logarithmus' notwendig ist und der Wertebereich schwerer im Rahmen einer Implementierung zu handhaben ist [54]. Allerdings kommt es zu keinen numerischen Instabilitäten und es ergibt sich eine natürlichere, additive Darstellung der Aktualisierungsgleichung (Gleichung 3.1):

$$\log \frac{p(m_i|z_{1:t})}{1 - p(m_i|z_{1:t})} = \log \frac{p(m_i|z_t)}{1 - p(m_i|z_t)} + \log \frac{1 - p(m_i)}{p(m_i)} + \log \frac{p(m_i|z_{1:t-1})}{1 - p(m_i|z_{1:t-1})}.$$

Die in der Literatur übliche, rekursive Schreibweise des binären Bayes-Filter in Log-Odds-Form erhält man durch Substitution mit $l_{t,i} := \log \frac{p(m_i|z_{1:t})}{1-p(m_i|z_{1:t})}$, Einbeziehung des Rekursionsanfangs $l_{0,i} := \log \frac{p(m_i)}{1-p(m_i)}$ und Umstellen als

$$l_{t,i} = l_{t-1,i} + \log \frac{p(m_i|z_t)}{1 - p(m_i|z_t)} - l_{0,i}.$$

Der Term $l_{t,i} = \log \frac{p(m_i|z_{1:t})}{1-p(m_i|z_{1:t})}$ auf der linken Seite stellt den gesuchten, neuen Belegtheitswert der Zelle mit Index i zum aktuellen Zeitpunkt t dar, der als neuer Wert dieser Zelle im Occupancy Grid gespeichert werden soll. Es handelt sich dabei um die in Log-Odds-Form dargestellte Belegtheitswahrscheinlichkeit dieser Zelle unter Berücksichtigung aller Messungen $z_{1:t}$, inklusive der aus dem aktuellen Zeitschritt t.

Der erste Terme auf der rechten Seite, $l_{t-1,i} = \log \frac{p(m_i|z_{1:t-1})}{1-p(m_i|z_{1:t-1})}$, stellt den bisherigen Belegtheitswert dieser Zelle mit Index i dar, der für diese Zelle derzeit noch im Occupancy Grid gespeichert ist; hier handelt es sich um die in Log-Odds-Form dargestellte Belegtheitswahrscheinlichkeit dieser Zelle unter Berücksichtigung aller vorheriger Messungen $z_{1:t-1}$, also exklusive der aus dem aktuellen Zeitschritt t. Da dieser Wert noch im Occupancy Grid gespeichert ist, kann die Berechnung des neuen Werts ohne eine vollständige Datenhistorie durchgeführt werden.

Um die Aktualisierungsgleichung verwenden zu können, sind zwei Wahrscheinlichkeiten zu spezifizieren, die über die beiden übrigen Terme auf der rechten Seite in die Berechnung eingehen: $p(m_i)$ und $p(m_i|z_t)$.

$p(m_i)$ wird als *Prior-Wahrscheinlichkeit* bezeichnet und geht durch den Term $l_{0,i} = \log \frac{p(m_i)}{1-p(m_i)}$ in Log-Odds-Form in die Berechnung ein. Über die Prior-Wahrscheinlichkeit kann Vorwissen über die Belegtheitswahrscheinlichkeit einer Zelle eingebracht werden, über das man unabhängig von irgendwelchen Messungen verfügt [19] (z.B. digitales Kartenmaterial). Typischerweise ist jedoch kein Vorwissen vorhanden und man nimmt Unwissenheit an: $p(m_i) = 1 - p(m_i) = 0{,}5$. In diesem Fall gilt $l_{0,i} = 0$, weshalb der letzte Term der Aktualisierungsgleichung entfällt.

Ebenfalls zu spezifizieren ist das so genannte *inverse Sensormodell* $p(m_i|z_t)$, über das die aktuelle Messung z_t eingebracht wird. Es beschreibt die Belegtheitswahrscheinlichkeit der Zelle i unter Berücksichtigung der aktuellen Messung z_t und geht durch den Term $\log \frac{p(m_i|z_t)}{1-p(m_i|z_t)}$ in Log-Odds-Form in die Berechnung ein. Aufgrund seiner herausragenden Bedeutung wird es im nächsten Abschnitt genauer besprochen.

3.1.2 Inverses Sensormodell

Das inverse Sensormodell[2] ist ein charakteristischer und wichtiger Bestandteil des Occupancy Grid Mappings. Es beschreibt die Wahrscheinlichkeit $p(m_i|z_t)$ – die Wahrscheinlichkeit, dass die Zelle mit Index i zum Zeitpunkt t *belegt* ist, bei gegebenem aktuellen Messwert z_t. Folglich schließt das inverse Sensormodell von der Wirkung (Messung) auf die Ursache (Belegtheit); es ist also „invers" zum Entstehungsprozess des Messwerts.

[2]Genau genommen müsste man vom *marginalisierten* inversen Sensormodell sprechen, da es nur eine einzelne Zelle betrachtet [50]. In der Literatur wird diese Unterscheidung aber meist nicht getroffen, so dass auch hier darauf verzichtet werden soll.

Die prinzipielle Vorgehensweise eines inversen Sensormodells lässt sich am besten anhand eines einfachen Beispiels verstehen. Betrachten wir deshalb im Folgenden zunächst den typischen Fall, in dem wir ein Lidar zur Erstellung eines Occupancy Grids benutzen wollen. Alle in der aktuellen, dem Autor bekannten Literatur beschriebenen inversen Lidarmodelle verwenden das so genannte *Raycasting*. Dabei werden die einzelnen physikalischen Laserstrahlen zwischen dem Lasersensor und einem Hindernis nachgebildet.

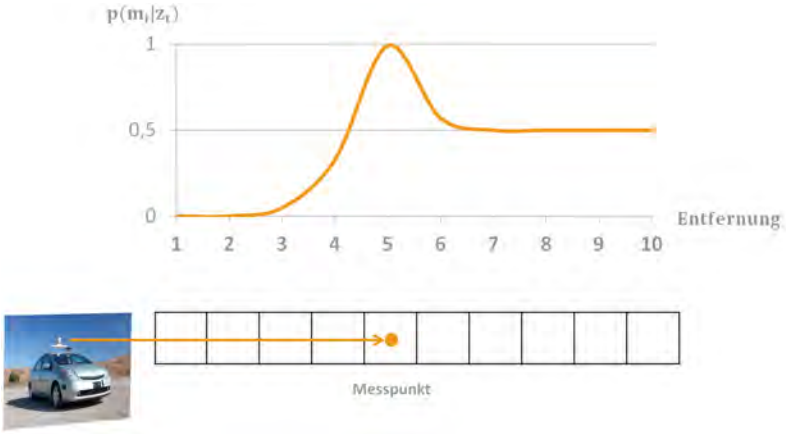

Abbildung 3.2: Schematische Darstellung eines inversen Sensormodells basierend auf Raycasting [Kleines Bild: techyyouth.com].

Im Abbildung 3.2 sind im unteren Teil ein einzelner Messwert, ein einzelner Laserstrahl und eine einzelne Zeile des Occupancy Grids gezeigt. Der Messwert gibt die Position eines detektierten Hindernisses an und wird als punktförmig angenommen, was im Falle eines typischen Lidars eine gerechtfertigte Annahme ist [58][51]. Zudem wird in dieser schematischen Darstellung vereinfachend angenommen, dass der Strahl und das Grid parallel zueinander ausgerichtet sind; im allgemeinen Fall muss der Strahl geeignet auf der Gitterstruktur approximiert werden, wozu oftmals der effiziente *Bresenham-Algorithmus* [6] eingesetzt wird: Übergibt man dem Algorithmus die Zellen am Anfang und Ende des Strahls, so erhält man eine Sequenz von Zellen zwischen Anfang und Ende, deren Abstand vom tatsächlichen Strahlenverlauf minimal ist.

Im oberen Teil der Abbildung ist die Belegtheitswahrscheinlichkeit der jeweiligen Zellen gezeigt unter Kenntnis des Messwerts. Dabei wird um den Messpunkt herum eine hohe Wahrscheinlichkeit für Belegtheit angenommen, da das Hindernis bei Verwendung eines üblichen Sensors tatsächlich etwa im Bereich der Messung liegen sollte. Durch die Höhe der Wahrscheinlichkeiten im Bereich um den Messpunkt herum lässt sich die Messunsicherheit des Sensors modellieren. Vor dem Hindernis werden Hindernisfreiheit und damit niedrige Belegtheitswahrscheinlichkeiten angenommen – das Hindernis kann nur mithilfe des Laserstrahls detektiert werden, wenn eine Sichtverbindung zwischen Hindernis und Lidar besteht. Hinter dem Hindernis wird typischerweise eine Wahrscheinlichkeit von 0,5 angenommen, das heißt, dass die Zelle gleich wahrscheinlich *belegt* oder *frei* ist. Das basiert auf der Überlegung, dass das Hindernis die Zellen hinter sich verschattet und dort dadurch keine Messungen möglich sind. Eine wichtige Ausnahme bilden moderne Mehrebenen-Lasersensoren, da diese im Schatten eines Hindernisses möglicherweise weitere Hindernisse detektieren können – je nach Höhe der Hindernisse. Im Fall eines solchen Sensors können an allen Hindernis-Messpunkten hohe Wahrscheinlichkeiten und vor dem ersten Hindernis Freiraum

angenommen werden; die restlichen Zellen sind entsprechend von einer Wahrscheinlichkeit von 0,5 umgeben [36].

Das typische Erscheinungsbild eines Occupancy Grids (vgl. Abbildung 3.1) ergibt sich, wenn man die Wahrscheinlichkeiten auf eine Graustufenskala abbildet, wobei hohe Belegtheitswahrscheinlichkeiten schwarz und niedrige weiß dargestellt werden. Für eine dreistufige Skala ist dies in Abbildung 3.3 gezeigt.

Welche Wahrscheinlichkeiten konkret in welchen Bereichen angenommen werden, ist abhängig vom eingesetzten Sensor – insbesondere von dessen Messprinzip, -unsicherheiten und -fehlern. Die entsprechenden Werte können durch maschinelles Lernen [50] oder aufgrund der Sensor-Spezifikationen des Herstellers ermittelt werden [39]. In der Praxis werden die Werte oft auch durch Experimente und aufgrund bisheriger Erfahrungen mit dem Sensor festgelegt [54][20][39].

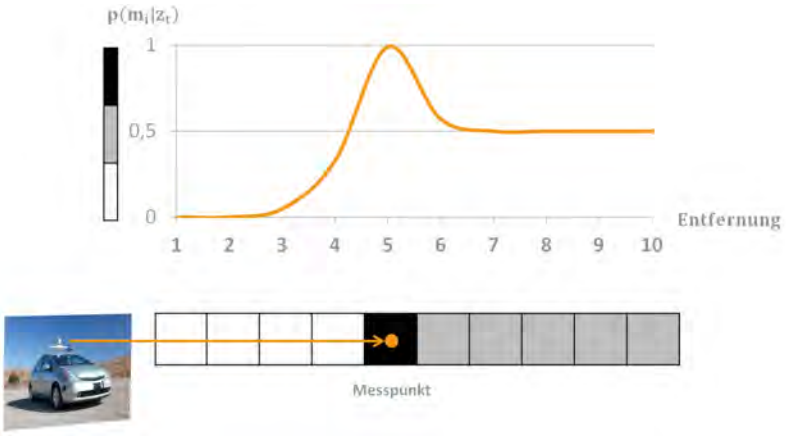

Abbildung 3.3: Zusammenhang zwischen inversem Sensormodell und Aussehen des Occupancy Grids [Kleines Bild: techyyouth.com].

Das inverse Sensormodell hat wesentlichen Einfluss auf die Genauigkeit des Grids, weshalb dessen Wahl eine entscheidende Bedeutung zukommt. Es kann stark vereinfachend sein bis hoch komplex. Ein genaues inverses Sensormodell berechnet die Belegtheitswahrscheinlichkeit einer Zelle basierend auf der Zellposition, der Sensorstrahlbreite und dem Abstand zum Strahlzentrum [27]. In [58] findet sich eine ausführliche theoretische Herleitung eines vollständigen inversen Sensormodells, welche auch der Rechtfertigung getroffener Vereinfachungen dienen kann. Außerdem wird gezeigt, dass eine sorgfältige Wahl notwendig ist – die Wahl eines falschen inversen Sensormodells kann zu einem Bias führen, durch den das Hindernis zwar im Grid angezeigt wird, jedoch um einige Zellen an der falschen Stelle.

Für die im Automobilbereich typischen Sensortechnologien existieren bereits einige Vorschläge für inverse Sensormodelle, die jeweils die spezifischen Charakteristiken des dort eingesetzten Sensors berücksichtigen. Da die Modelle meist auf andere Sensoren diesen Typs und damit möglicherweise den eigenen Anwendungsfall übertragbar sind, soll im Folgenden ein kurzer Überblick gegeben werden. Occupancy Grids auf Basis von Kameras [46][34][32] werden dabei ausgeklammert, da sie bislang nur eine untergeordnete Rolle spielen und wenig verbreitet sind.

Üblicherweise werden zur Erstellung automobiler Occupancy Grids Laser- und/oder Radarsensoren eingesetzt. Moderne Lasersensoren eignen sich hervorragend zur Kartierung,

denn sie bieten eine hohe Ortsauflösung und können dadurch auch Konturen erkennen. Radarsensoren scheinen aufgrund ihrer relativ schlechten Ortsauflösung auf den ersten Blick eher ungeeignet, allerdings werden sie bereits serienmäßig in einigen Autos eingesetzt und können auch Geschwindigkeiten messen, was für die Erkennung dynamischer Objekte wertvoll ist (vgl. Abschnitt 3.3.1). Prinzipiell können bei den inversen Sensormodellen beider Sensorarten zwei Bestandteile unterschieden werden [54]: Hinderniskartierung und Freiraummodellierung. Die Hinderniskartierung ist für beide Sensortypen recht ähnlich. Auf eine Freiraummodellierung wird bei Radarsensoren oft verzichtet, da sie nur punktförmige Hindernisse mit hoher Positionsunsicherheit liefern.

Im Rahmen der *Hinderniskartierung* werden die vom Sensor detektierten Hindernisse in das Occupancy Grid eingetragen. Dabei ist festzulegen, welche Form für das Hindernis angenommen wird. Im Falle eines relativ präzisen Sensors kann jeder Messwert direkt *punktförmig* kartiert werden, wie in obiger, schematischer Darstellung. Diese Methode wurde zum Beispiel für die Lasersensoren in [54][52] gewählt. Besitzt der eingesetzte Sensor eine nicht vernachlässigbare Strahlenbreite, so sollte der Messwert stattdessen *linienförmig* kartiert werden; bei der Freiraummodellierung ist in diesem Fall zu beachten, dass der Raum vor dem Messwert keine Linie, sondern ein Dreieck ist. Dieses Vorgehen verwenden beispielsweise [4][5], da der dortige Laserscanner eine Strahlenbreite von bis zu vier Grad besitzt. Den punkt- oder linienförmig kartierten Zellen kann ein konstanter oder entfernungsabhängiger Wert als Belegtheitswahrscheinlichkeit zugewiesen werden; letzteres ist dann empfehlenswert, wenn die Zuverlässigkeit des Sensors stark von der Distanz abhängt. In jedem Fall sollte die mit der Messung verbundene Positionsunsicherheit berücksichtigt werden. Für kleinere Unsicherheiten kann dies ebenfalls über die Höhe der kartierten Belegtheitswahrscheinlichkeit passieren.

Ist diese Positionsunsicherheit jedoch relativ groß – der Regelfall bei Radarsensorik –, sollte stattdessen eine andere Modellierungsform verwendet werden. Eine für Radarsensoren übliche [4][5][20], manchmal aber auch für Lasersensoren eingesetzte [33][31] Methode ist die Verwendung einer zweidimensionalen Gaußfunktion, die eine größere *kreis-* oder *ellipsenförmige* Region um den eigentlichen Messpunkt beschreibt. Die beiden Standardabweichungen als Parameter der Funktion modellieren die Messungenauigkeiten, optional gleichzeitig auch die Unsicherheiten der Eigenpositionsschätzung. Die Höhe der kartierten Belegtheitswahrscheinlichkeit hängt typischerweise vom Ergebnis der Funktion ab.

Arbeitet man mit vorverarbeiteten Sensordaten, sind oft nicht einzelne Messwerte, sondern mehrere, zu Objekten zusammengefasste Messwerte zu kartieren. Der in [20] verwendete Laserscanner liefert beispielsweise eine Liste dynamischer Objekte, wobei zu jedem Objekt dessen einzelne Messpunkte angegeben sind. In diesem Fall kann zunächst eine rechteckige Box berechnet werden, die möglichst eng alle Messwerte umfasst – eine so genannte *Bounding Box*. Oft ist diese Bounding Box auch bereits das Ergebnis der sensorinternen Vorverarbeitung. Nun erfolgt eine *boxförmige* Kartierung, indem obiger Raycasting-Ansatz auf den zweidimensionalen Fall erweitert wird: anstelle des punktförmigen Hindernisses betrachtet man die Box als Hindernis; im Dreieck vor der Box wird Freiraum, im Bereich hinter der Box Unwissenheit angenommen. Meist wird allen Zellen der Bounding Box dieselbe, konstante Belegtheitswahrscheinlichkeit zugewiesen; in [20] beispielsweise 0,8.

Manchmal kann es jedoch auch sinnvoll sein, bestimmte Messwerte *gar nicht* zu kartieren; hauptsächlich ist das der Fall, wenn die mit der Messung verbundene Unsicherheit zu groß ist oder die Messung auf ein dynamisches Objekt zurückzuführen ist (mehr dazu

später in Abschnitt 3.3.1). Zum Beispiel nimmt [49] Radarmessungen in Entfernungen über 50 Meter von der Kartierung aus, da dort die laterale Unsicherheit zu groß ist; für nähere Messungen wird diese hingegen durch die zeitliche Integration in das Occupancy Grid kompensiert.

Der zweite Bestandteil des inversen Sensormodells ist die *Freiraummodellierung*. Da ein Sensor typischerweise direkt nur Hindernisse und nur implizit Informationen über Freiraum liefert, kommt eine so genannte *Freiraumfunktion* zum Einsatz. Diese Funktion modelliert – unter Berücksichtigung der Sensorcharakteristiken – die Unsicherheiten, dass Bereiche ohne Messung belegt sind.

Eine einfache Möglichkeit zum Umgang mit dem freien Raum vor einem Hindernis ist, dass allen dortigen Zellen dieselbe *konstante, niedrige* Belegtheitswahrscheinlichkeit zugewiesen wird, wie zum Beispiel 0 in [50] oder 0,2 in [52][20]. Eine derartige Modellierung berücksichtigt jedoch nicht, dass mit zunehmender Entfernung vom Sensor der Abstand zwischen zwei benachbarten Laserstrahlen größer wird und damit die Wahrscheinlichkeit dafür steigt, dass sich im Zwischenraum ein undetektiertes Objekt befindet.

Besser ist deshalb im Allgemeinen eine Modellierung, die die Belegtheit einer Zelle in Abhängigkeit von ihrer *radialen Entfernung* vom Sensor bestimmt. Oft wird hierzu eine lineare Funktion verwendet, wie zum Beispiel in [54]. Dort liefert diese eine minimale Belegtheitswahrscheinlichkeit von 0,04 in 10 Zentimeter Entfernung und eine maximale von 0,4 in 50 Meter Entfernung. Da Messungen in größeren Entfernungen nicht mehr zuverlässig genug sind, wird dort Unwissenheit angenommen (Belegtheitswahrscheinlichkeit von 0,5) – obwohl der Laser Messwerte in Entfernungen bis zu 200 Meter liefert.

Komplexere Modellierungen können sinnvoll sein, wenn der Sensor gewisse Charakteristiken aufweist, die berücksichtigt werden sollen. Die Freiraumfunktion in [33] modelliert beispielsweise, dass der dort verwendete Sensor aufgrund seiner erhöhten Einbauposition nahe Objekte nicht detektieren kann (die Laserstrahlen gehen über das Objekt hinweg) und dass seine Winkelauflösung an den Rändern des Sichtfelds niedriger ist (dadurch ist zwischen den Laserstrahlen mehr Raum für undetektierte Objekte). Prinzipiell sind solch realitätsnähere Modellierungen den einfacheren vorzuziehen, allerdings ist zu berücksichtigen, dass die Freiraumfunktion sehr oft ausgeführt werden muss und deshalb eine Echtzeitverwendung leicht unmöglich werden kann.

3.1.3 Beschränkungen

Das eben beschriebene, klassische Occupancy Grid Mapping wird mit Erfolg in der Praxis eingesetzt [50][39][54]. Dennoch gibt es Situationen, in denen es an seine Grenzen stößt.

Betrachten wir einen einzelnen Messwert eines Sensors, um dies einzusehen. Dessen Auswertung bezüglich eines Ereignisses kann im Allgemeinen zu drei Fällen führen [17]:

- Der Messwert spricht für das Ereignis. Im Rahmen des Occupancy Grid Mappings kann die Messung beispielsweise auf ein Hindernis hindeuten und dadurch zur Aussage „Diese Zelle ist ziemlich sicher belegt“ führen.

- Der Messwert spricht für die Negation des Ereignisses; beispielsweise „Die Zelle ist ziemlich sicher nicht belegt, sondern frei.“

- Der Messwert spricht weder für das Ereignis noch für seine Negation – es ist keine

oder keine zuverlässige Aussage möglich: „Die Zelle ist scheinbar nicht belegt, aber scheinbar auch nicht frei."

Letzterer Fall kann sich insbesondere dann ergeben, wenn man die Daten mehrerer, gleich zuverlässiger Sensoren auswertet und diese widersprüchliche Informationen liefern. Liefert beispielsweise zum Zeitpunkt t Sensor Nummer 1 für die Zelle mit Index i unter Berücksichtigung seiner aktuellen Messung $z_{1,t}$ die Belegtheitswahrscheinlichkeit $p(m_i|z_{1,t}) = 0{,}0$, die für sichere Freiheit dieser Zelle spricht, aber Sensor Nummer 2 für dieselbe Zelle zur gleichen Zeit unter Berücksichtigung seiner aktuellen Messung $z_{2,t}$ die Belegtheitswahrscheinlichkeit $p(m_i|z_{2,t}) = 1{,}0$, die für sichere Belegtheit dieser Zelle spricht, so ergibt sich – eine gleiche Zuverlässigkeit und deshalb gleich gewichtete Kombination angenommen – als kombinierte Belegtheitswahrscheinlichkeit $p(m_i|z_t) = 0{,}5$. Wir erinnern uns, dass auch als initialer Belegtheitswert typischerweise 0,5 gewählt wird. Fehlendes und unsicheres Wissen sind also nicht unterscheidbar.

Das lässt sich auf zwei Punkte zurückführen. Zum Einen ist der initiale Zustand gleichbedeutend mit „Die Zelle ist zu 50% belegt". Diese Annahme wird getroffen, ohne dass irgendeine Messung zugrundeliegt [41]. Zum Anderen gilt, betrachtet man ein Ereignis E bei gemachter Beobachtung O, stets $P(\overline{E}|O) = 1 - P(E|O)$, unabhängig von der tatsächlichen Information über das inverse Ereignis $\overline{E}$. Die Aussage „Die Zelle ist zu 80% belegt" ist dadurch gleichbedeutend mit „Die Zelle ist zu 20% frei". Möchte man beispielsweise die Unsicherheiten des zweiten Sensors modellieren, indem er nicht $p(m_i|z_{2,t}) = 1{,}0$, sondern nur $p(m_i|z_{2,t}) = 0{,}8$ liefert, ist das deshalb problematisch.

Es ist erkennbar, dass es unter Verwendung klassischer Wahrscheinlichkeiten Schwierigkeiten gibt, oben beschriebenen dritten Fall – der Messwert spricht weder für das Ereignis noch für seine Negation – auszudrücken [9][17].

Eine mögliche Lösung ist es, zusätzlich zu jeder Wahrscheinlichkeit ein Gewicht zu bestimmen, das diese Unsicherheit wie gewünscht modelliert. Anschließend werden die Wahrscheinlichkeiten entsprechend gewichtet kombiniert. Dieser Ansatz wird als *Linear Opinion Pools* bezeichnet. Er wird beispielsweise in [1] im Rahmen automobiler Occupancy Grids beschrieben. Die Autoren berichten von besseren Ergebnissen als mit dem klassischen Ansatz im Falle widersprüchlicher oder fehlerhafter Informationen.

Eine üblichere Möglichkeit zur Lösung bietet allerdings die im folgenden Abschnitt vorgestellte *Dempster-Shafer-Theorie*, die unter anderem in [36][41][22] zur Erstellung automobiler Occupancy Grids eingesetzt wird.

3.2 Erweiterung durch die Dempster-Shafer-Theorie

Der vorherige Abschnitt hat gezeigt, dass das klassische Occupancy Grid Mapping bei konfliktbehafteten, unsicheren oder unvollständigen Messdaten an seine Grenzen gelangt. In diesem Fall kann die Verwendung der Dempster-Shafer-Theorie von Vorteil sein, die die klassische Wahrscheinlichkeitstheorie in gewisser Weise erweitert. Im Folgenden wird sie zunächst eingeführt; anschließend werden evidenztheoretische Occupancy Grids vorgestellt und mit dem klassischen Ansatz verglichen.

3.2.1 Überblick über die Dempster-Shafer-Theorie

Die Dempster-Shafer-Theorie, die auch als *Evidenztheorie* bezeichnet wird, bietet eine Alternative für die mathematische Behandlung von Unsicherheiten zur klassischen Wahrscheinlichkeitstheorie. Sie geht zurück auf Arbeiten von Arthur Dempster Ende der 1960er-Jahre, in denen er unter anderem den Satz von Bayes verallgemeinerte [14]. In den folgenden Jahren kamen Erweiterungen durch Glenn Shafer hinzu [45]. Eine Herleitung der Dempster-Shafer-Theorie übersteigt den Rahmen dieser Arbeit, allerdings sollen nachfolgend die wichtigsten Konzepte eingeführt werden, basierend auf [15][39][28][17]. Ein Beispiel für die Anwendung der Theorie findet sich im nächsten Abschnitt, wo sie im Rahmen des Occupancy Grid Mappings verwendet wird.

Die Evidenztheorie kann als Erweiterung der Bayes-Theorie angesehen werden, die durch explizite Modellierung von Unwissenheit auch Unentschiedenheit erlaubt. Das ermöglicht die Differenzierung zwischen den Messwerten selbst und deren Glaubwürdigkeit; anders als in der Bayes-Theorie, wo die Glaubwürdigkeit implizit an den Messwert gebunden ist [15]. Anstatt lediglich Elementarereignissen Wahrscheinlichkeiten zuzuweisen, werden im Rahmen der Dempster-Shafer-Theorie beliebigen Teilmengen des Ereignisraums so genannte *Evidenzen* zugeordnet. Die unterschiedliche Begrifflichkeit soll ausdrücken, dass Evidenzen und Wahrscheinlichkeiten sich zwar ähnlich, aber nicht dasselbe sind [15][28]. Evidenzen sind ein Indiz oder eine Beobachtung für oder gegen eine beliebige Teilmenge des Ereignisraums [17]; dies ist insbesondere nützlich im Falle unsicherer oder widersprüchlicher Messungen. Der Grad an Widersprüchlichkeit kann mithilfe eines Konfliktmaßes quantifiziert werden, wie wir im Folgenden sehen werden.

Der *Wahrnehmungsrahmen* Θ (engl. frame of discernment) bildet die Grundlage. Es handelt sich dabei um eine endliche Menge, die alle möglichen und sich gegenseitig ausschließenden Klassen enthält.

Evidenzen können aber nicht nur diesen Klassen zugewiesen werden, sondern auch Kombinationen davon. Die Menge aller möglichen Aussagen entspricht der *Potenzmenge* des Wahrnehmungsrahmens 2^Θ. Sie stellt eine Hierarchie aller möglichen Aussagen dar, da sie feinere Aussagen zu gröberen gruppiert. Darin ist auch der Wahrnehmungsrahmen Θ als Aussage enthalten, der alle anderen Aussagen zusammenfasst und damit für „unbekannt" oder „keine Aussage möglich" steht.

Einer Aussage wird – anstelle einer Wahrscheinlichkeit der klassischen Wahrscheinlichkeitstheorie – eine Evidenz zugewiesen, in Form einer so genannten *Masse* mithilfe einer *Massefunktion*. Eine Massefunktion ist eine Funktion $m : 2^\Theta \to [0,1]$, für die $m(\emptyset) = 0$ und $\sum_{X \epsilon 2^\Theta} m(X) = 1$ gilt. Für eine Aussage $X \in 2^\Theta$ repräsentiert $m(X)$ die zugeordnete Masse, die ein nicht-negatives Vertrauensmaß darstellt. $m(\emptyset) = 0$ bedeutet dabei, dass der leeren Menge keine Masse ungleich 0 zugewiesen werden darf – der Sensor hat immer eine Beobachtung gemacht, selbst wenn sie mehrdeutig ist. $\sum_{X \epsilon 2^\Theta} m(X) = 1$ entspricht ähnlich zur klassischen Wahrscheinlichkeitstheorie der Anforderung, dass alle Massen zu 1 summieren müssen.

Jeder Sensor erzeugt eine individuelle Belegung für alle Aussagen $X \in 2^\Theta$, abhängig vom Messwert und der Messunsicherheit, indem er eine Gesamtmasse von 1 auf alle nichtleeren Aussagen verteilt. Massenzuweisungen zu mehrelementigen Aussagen erlauben dabei die Berücksichtigung der Unsicherheit über das konkrete Element dieser Menge; es wird ausgesagt, dass zwar eines der Elemente dieser Menge das gesuchte Element ist, aber nicht

klar ist, welches genau. Vollständiges Unwissen kann durch $m(\Theta) = 1$ modelliert werden, da die Aussage Θ alle Elemente enthält und damit Unwissenheit repräsentiert. Eine mögliche Differenz der Summe verteilter Massen auf 1 wird deshalb auch immer der Aussage Θ zugeordnet.

Über eine Kombinationsoperation können zwei Massen zu einer neuen Masse kombiniert werden. Die dafür bekannteste und im Folgenden vorgestellte Möglichkeit ist *Dempsters Kombinationsregel* (engl. Dempster's Rule of Combination). Einen Überblick über einige Alternativen inklusive deren Bewertung unter Verwendung einer Implementierung gibt [44].

Dempsters Kombinationsregel ist eine kommutative und assoziative Verallgemeinerung des Satzes von Bayes [14]. Für zwei Massefunktionen m_1, m_2 und die Aussagen X,A,$B \in 2^\Theta$ lautet sie:

$$m_1(X) \otimes m_2(X) := \begin{cases} \frac{\sum_{A \cap B = X} m_1(A) m_2(B)}{1-k} & f\ddot{u}r\, X \neq \emptyset \\ 0 & f\ddot{u}r\, X = \emptyset \end{cases}$$

mit

$$k := \sum_{A \cap B = \emptyset} m_1(A) m_2(B).$$

Der Zähler summiert die Massen aller Aussagen, in denen X enthalten ist; er misst damit die Unterstützung für die Aussage X. Der Nenner repräsentiert dagegen die gesamte gültige Unterstützung, das heißt die gesamte Unterstützung, die nicht der leeren Menge zugeordnet würde; er stellt damit einen Normierungsterm dar. Insgesamt ist das Ergebnis von Dempsters Kombinationsregel die normierte Unterstützung für die Aussage X, das heißt die für X sprechende *Evidenz*, die sich aus den beiden Massen $m_1(X)$ und $m_2(X)$ ergibt.

Die Normalisierung ist notwendig, da der leeren Menge keine Masse ungleich 0 zugewiesen werden darf (vgl. Definition der Massefunktion oben); was aber durch den Zähler alleine im Falle widersprüchlicher Massen passieren würde. Die Normalisierung führt dazu, dass die leere Menge wieder die Masse 0 besitzt und die Summe der Massen aller möglichen Aussagen wieder 1 ergibt.

Durch die Normalisierung erhält man ein geeignetes Maß für Evidenz, aber in etlichen praktischen Anwendungen möchte man trotzdem über aufgetretene Widersprüche Bescheid wissen. Der Wert von k im Nenner stellt die Masse dar, die der leeren Menge zugeordnet werden würde. Da der leeren Menge jeglicher Widerspruch zugerechnet wird, handelt es sich bei $k \in [0,1]$ um ein Maß für die Widersprüchlichkeit der beiden betrachteten Massen; k wird deshalb als *Konfliktmasse* bezeichnet [21]. Shafer hat zudem eine Konfliktmetrik eingeführt, die auf k basiert, die so genannte *„weight of conflict"-Metrik* Con:

$$Con := \log(\frac{1}{1-k}) = -\log(1-k)$$

Es gilt $k \to 0 \Rightarrow Con \to 0$ und $k \to 1 \Rightarrow Con \to \infty$; der Wertebereich von Con ist also $[0,\infty[$. Da Con additiv ist, kann damit auch der Konflikt zwischen mehr als zwei Massen gemessen werden.

Bei hohen Widersprüchen, also bei $k \to 1$, ergibt die Kombinationsregel keine intuitiv sinnvollen Ergebnisse mehr [15]; für $k = 1$ ist sie nicht definiert, weshalb $m_1(X)$ und $m_2(X)$ in diesem Fall als *nicht kombinierbar* bezeichnet werden.

Mit der *Vertrauensfunktion* und der *Plausibilitätsfunktion* existiert ein Konzept, das es in der klassischen Wahrscheinlichkeitstheorie nicht gibt. In gewisser Weise ist dieses Konzept als Lösung für das Problem zu verstehen, dass die kombinierten Massen bei hohen Widersprüchen oftmals keine intuitiven Ergebnisse mehr liefern. Hier kann die Verwendung dieser beiden Funktionen hilfreich sein. Seien im Folgenden m eine Massefunktion und $X \in 2^\Theta$ eine Aussage.

Die *Vertrauensfunktion* (engl. belief function) ist dann die Funktion $Bel: 2^\Theta \to [0,1]$, für die $Bel(X) = \sum_{A \subseteq X} m(A)$ gilt. Sie quantifiziert das Vertrauen in X, das heißt die gesamte Unterstützung für die Aussage X, indem sie die Massen aller Teilmengen von X akkumuliert. $Bel(X)$ kann als eine Art untere Grenze der für X sprechenden Evidenz aufgefasst werden.

Die *Plausibilitätsfunktion* (engl. plausibility function) ist die Funktion $Pls: 2^\Theta \to [0,1]$, für die $Pls(X) = \sum_{A \cap X \neq \emptyset} m(A)$ gilt. Sie quantifiziert die Plausibilität von X, das heißt die insgesamt mögliche Unterstützung für die Aussage X, indem sie die Massen aller Aussagen akkumuliert, die eine nicht-leere Schnittmenge mit X besitzen. $Pls(X)$ kann als eine Art obere Grenze der für X sprechenden Evidenz aufgefasst werden.

Die tatsächliche Evidenz für X befindet sich im Bereich zwischen Vertrauen $Bel(X)$ und Plausibilität $Pls(X)$. Die *Intervallbreite* $Pls(X) - Bel(X)$ repräsentiert die Unsicherheit in Bezug auf die Aussage X. $Bel(X) = Pls(X)$ gilt im Falle vollständiger Information, in dem jede mehrelementige Aussage eine Masse von 0 besitzt. Dies entspricht dem in der klassischen Wahrscheinlichkeitstheorie angenommenen Szenario, weshalb die evidenztheoretische Darstellung als deren Erweiterung aufgefasst werden kann.

3.2.2 Evidenztheoretische Occupancy Grids

In diesem Abschnitt soll die Verwendung der eben eingeführten Dempster-Shafer-Theorie im Rahmen des Occupancy Grid Mappings besprochen werden, basierend auf [39][15][17].

Auch in der evidenztheoretischen Variante werden ein statisches Umfeld und voneinander unabhängige Zellen angenommen. Eine Zelle besitzt wie gehabt die beiden möglichen Zustände *belegt* und *frei* (abgekürzt mit B bzw. F), so dass sich als Wahrnehmungsrahmen

$$\Theta := \{B,F\}$$

ergibt. Die Potenzmenge, die alle möglichen Aussagen enthält, ist dadurch

$$2^\Theta := \{\emptyset,\{B\},\{F\},\{B,F\}\} = \{\emptyset,\{B\},\{F\},\Theta\}.$$

Der Wahrnehmungsrahmen Θ als Aussage fasst die Aussagen $\{B\}$ und $\{F\}$ zusammen und steht damit für *unbekannt* oder *keine Aussage möglich*.

Im klassischen Occupancy Grid Mapping besitzt eine Zelle mit Index i als Wert $p(m_i|z_{1:t})$, also die bis zum aktuellen Zeitpunkt t akkumulierte Wahrscheinlichkeit für ihre Belegtheit; in der evidenztheoretischen Variante werden stattdessen die Massen $m_{i,1:t}(\{B\})$, $m_{i,1:t}(\{F\})$ und $m_{i,1:t}(\Theta)$ für die möglichen Aussagen *belegt*, *frei* bzw. *unbekannt* gespeichert. Die Masse der leeren Menge $m_{i,1:t}(\emptyset)$ ist immer 0 und braucht deshalb nicht gespeichert werden. Prinzipiell ist je Zelle sogar das Speichern von zwei dieser Massen ausreichend, da $m_{i,1:t}(\{B\}) + m_{i,1:t}(\{F\}) + m_{i,1:t}(\Theta) = 1$ gilt; in diesem Fall benötigt man zur Rekonstruktion der dritten Masse allerdings einen zusätzlichen Berechnungsschritt, weshalb es sich hier um eine Abwägung zwischen Speicher- und Berechnungsaufwand handelt.

Analog zur klassischen Variante wird initial (also zum Zeitpunkt $t = 0$) – sofern kein Vorwissen über die Belegtheit der Zelle mit Index i zur Verfügung steht – vollständige Unwissenheit angenommen: $m_{i,0}(\{B\}) = 0$, $m_{i,0}(\{F\}) = 0$ und $m_{i,0}(\Theta) = 1$.

Wie beim klassischen Occupancy Grid Mapping werden Messwerte durch ein inverses Sensormodell in eine Zellbelegung überführt (vgl. Abschnitt 3.1.2). Hier überführt es für eine Zelle mit Index i den zum Zeitpunkt t aktuellen Messwert z_t in die Massen $m_{i,t}(\{B\})$, $m_{i,t}(\{F\})$ und $m_{i,t}(\Theta)$, wobei $m_{i,t}(\{B\})$ und $m_{i,t}(\{F\})$ die Evidenz für Belegtheit beziehungsweise Freiheit der Zelle aufgrund der Messung quantifizieren und $m_{i,t}(\Theta)$ die dabei vorhandene Unsicherheit.

Die Kartierung eines Hindernisses führt typischerweise zu einer Massenverteilung der Form $m_{i,t}(\{B\}) = b$, $m_{i,t}(\{F\}) = 0$, $m_{i,t}(\Theta) = 1 - b$, wobei die Höhe des Werts $b \in [0,1]$ die Sicherheit für Belegtheit widerspiegelt, basierend auf dem zugrundeliegenden Messwert und der Messunsicherheit. Analog führt die Freiraummodellierung zu einer Massenverteilung der Art $m_{i,t}(\{B\}) = 0$, $m_{i,t}(\{F\}) = f$, $m_{i,t}(\Theta) = 1 - f$, wobei die Höhe des Werts $f \in [0,1]$ die Sicherheit für Freiheit widerspiegelt.

Anschließend werden $m_{i,t}(\{B\})$ und $m_{i,t}(\{F\})$ durch eine Aktualisierungsgleichung mit den bisher im Occupancy Grid gespeicherten Werten $m_{i,1:t-1}(\{B\})$ und $m_{i,1:t-1}(\{F\})$ zu den neuen Werten $m_{i,1:t}(\{B\})$ und $m_{i,1:t}(\{F\})$ kombiniert. Hierfür kann zum Beispiel Dempsters Kombinationsregel zum Einsatz kommen; für die *belegt*-Masse $m_{i,t}(\{B\})$ lautet sie:

$$\begin{aligned} m_{i,1:t}(\{B\}) &= m_{i,1:t-1}(\{B\}) \otimes m_{i,t}(\{B\}) = \\ &= \frac{m_{i,1:t-1}(\{B\})m_{i,t}(\{B\}) + m_{i,1:t-1}(\{B\})m_{i,t}(\Theta) + m_{i,1:t-1}(\Theta)m_{i,t}(\{B\})}{1-k} \end{aligned}$$

mit

$$k = m_{i,1:t-1}(\{B\})m_{i,t}(\{F\}) - m_{i,1:t-1}(\{F\})m_{i,t}(\{B\}).$$

Analog erfolgt die Berechnung der *frei*-Masse $m_{i,1:t}(\{F\})$. Die *unbekannt*-Masse $m_{i,1:t}(\Theta)$ erhält man aufgrund der Summation der drei Massen zu 1 über

$$m_{i,1:t}(\Theta) = 1 - m_{i,1:t}(\{B\}) - m_{i,1:t}(\{F\}).$$

Die Konfliktmasse k misst den bei der Kombination aufgetretenen Widerspruch – zwischen dem bisher im Occupancy Grid gespeicherten und dem auf der aktuellen Messung basierenden Belegtheitswert. Widerspruch entsteht, wenn ein Hindernis in einer bisher *freien* Zelle kartiert wird oder wenn eine bisher durch ein Hindernis *belegte* Zelle wieder *frei* wird. Solch ein Widerspruch kann ein Hinweis auf ein dynamisches Objekt sein, da seine Bewegung zu einem Wechsel der Zellbelegung führt (vgl. Abschnitt 3.3.1): Es kann sich auf eine bis dahin *freie* Zelle bewegen, so dass diese nun *belegt* ist; oder es verlässt eine von ihm bis dahin *belegte* Zelle, so dass diese nun wieder *frei* ist.

Der zu beachtende Fall $k = 1$, in dem Dempsters Kombinationsregel nicht definiert ist, tritt im Rahmen des Occupancy Grid Mappings nur auf, wenn sehr widersprüchliche und gleichzeitig sehr zuverlässige Aussagen kombiniert werden sollen. Dies ist bei korrekter Modellierung jedoch ausgeschlossen, da maximal zuverlässige Sensoren sich nie widersprechen.

Außerdem ist zu beachten, dass – ähnlich zum klassischen Occupancy Grid Mapping – nur voneinander unabhängige Beobachtungen und damit Massen kombiniert werden dürfen. Das heißt, dass zwei eine Zelle betreffende Massen nur kombiniert werden dürfen, wenn sie von unterschiedlichen Zeitpunkten oder Sensoren stammen [29][39].

Abschließend sei angemerkt, dass das in der Dempster-Shafer-Theorie bestehende Konzept aus Vertrauens- und Plausibilitätsfunktion im Falle des Occupancy Grid Mappings von nachrangiger Bedeutung ist. Die Funktionen bieten keinen echten Mehrwert, da der Wahrnehmungsrahmen nur aus den zwei Klassen B und F besteht. Um dies einzusehen, sei m eine Massefunktion. Für die Plausibilitätsfunktion gilt $Pls(\{F\}) = m(\{F\}) + m(\Theta)$, $Pls(\{B\}) = m(\{B\}) + m(\Theta)$ und $Pls(\Theta) = 1$; es ergibt sich also nur ein leichter Informationsgewinn gegenüber der direkten Verwendung der Massen. Gar keinen Informationsgewinn bringt die Berechnung von Vertrauensfunktion und Intervallbreite: für das Vertrauen gilt $Bel(\{F\}) = m(\{F\})$, $Bel(\{B\}) = m(\{B\})$ und $Bel(\Theta) = 1$ – und damit für die Intervallbreite $Pls(\{F\}) - Bel(\{F\}) = m(\Theta)$ bzw. $Pls(\{B\}) - Bel(\{B\}) = m(\Theta)$.

3.2.3 Vergleich & Fazit

In den vorangehenden Abschnitten wurden mit der klassischen und der evidenztheoretischen Variante des Occupancy Grid Mappings die beiden wichtigsten Vertreter eingeführt. Eine zusammenfassende Gegenüberstellung findet sich in Tabelle 3.1.

	klassisches OGM	evidenztheoretisches OGM
Theorie	klassische Wahrscheinlichkeitstheorie	Dempster-Shafer-Theorie
Vertrauensmaß	klassische Wahrscheinlichkeit	Evidenzen in Form von Massen; Vertrauen/Plausibilität
Berechnungskomplexität	niedriger	höher
Aktualisierungsgleichung	binärer, statischer Bayes-Filter	Dempsters Kombinationsregel; ...
Prior-Wissen	Prior-Wahrscheinlichkeiten nötig	optional einbringbar
Fehlendes und unsicheres Wissen	nicht unterscheidbar	extra Zustand; mehrelementige Aussagen
Qualitätsmaß	nicht vorhanden	Intervallbreite; Konflikt
Konfliktmaß	nicht vorhanden	Konfliktmasse k; Con-Metrik; ...

Tabelle 3.1: Gegenüberstellung beider Varianten des Occupancy Grid Mappings (OGM).

Abschließend sollen die Unterschiede zusätzlich im Rahmen eines einfachen Beispiels illustriert und ein Fazit gezogen werden. Das Beispiel basiert auf [9]; Abbildung 3.4 gibt einen Überblick. Wir betrachten den Belegtheitswert einer einzelnen Zelle zu drei aufeinanderfolgenden Zeitpunkten, sowohl für das klassische als auch für das evidenztheoretische Occupancy Grid Mapping.

Zum Zeitpunkt $t = 0$ wird das Occupancy Grid initialisiert; es steht kein Vorwissen über das Umfeld (z.B. in Form digitaler Karten) zur Verfügung. Im Falle des klassischen Occupancy Grid Mappings wird der Wert 0,5 gewählt, da die Zelle gleich wahrscheinlich *belegt* oder *frei* ist. Dies entspricht der Wahrscheinlichkeit, dass die Zelle zu 50% *belegt* ist – obwohl dafür noch keine Messung einen Hinweis geliefert hat. Im Falle des evidenztheoretischen Occupancy Grid Mappings wird für die Massen *frei* und *belegt* der Wert 0 gewählt, für *unbekannt* der Wert 1. Dies repräsentiert perfekt unser Wissen: Wir haben keinen Hinweis dafür erhalten, dass die Zelle *frei* oder *belegt* ist, jedoch ist sie mit hundertprozentiger Sicherheit entweder *frei* oder *belegt*.

Zum Zeitpunkt $t = 1$ geht die erste Messung ein, die zu 84% darauf schließen lässt, dass die betrachtete Zelle *belegt* ist. Im Falle des klassischen Occupancy Grid Mappings wird deshalb der Wert 0,84 in das Occupancy Grid integriert. Der Wert des Occupancy Grids ergibt sich aus der Kombination des bisherigen Werts von 0,5 mit dem neuen Wert 0,84 mittels binärem Bayes-Filter; es ergibt sich ein Wert von 0,84 nach Integration der ersten Messung. Man beachte, dass dieser Wert impliziert, dass die Zelle zu 16% frei ist.

Im Falle des evidenztheoretischen Occupancy Grid Mappings wird für die *belegt*-Masse der Wert 0,84 gewählt, die verbleibende Masse von $1 - 0{,}84 = 0{,}16$ erhält die Masse für *unbekannt*. Dies repräsentiert wieder perfekt unser Wissen: Die bisher erhaltene Evidenz für Belegtheit ist 0,84 und zu 16% wissen wir nichts über den Zustand der Zelle, sie kann also entweder *frei* oder *belegt* sein. Nach Integration der ersten Messung mithilfe Dempsters Kombinationsregel ergeben sich diese Massen auch als Wert des Occupancy Grids.

Zum Zeitpunkt $t = 2$ trifft eine weitere Messung ein, die der ersten Messung genau widerspricht: Sie deutet zu 84% auf eine *freie* Zelle hin. Im Falle des klassischen Occupancy Grid Mappings wird deshalb der Wert $1-0{,}84 = 0{,}16$ als Belegtheitswert gewählt; es ergibt sich ein Wert von 0,5 im Occupancy Grid nach Integration dieser zweiten Messung.

Im Falle des evidenztheoretischen Occupancy Grid Mappings wird analog zum vorherigen Zeitpunkt vorgegangen: Für die *frei*-Masse wird der Wert 0,84 gewählt, die verbleibende Masse von $1 - 0{,}84 = 0{,}16$ erhält die *unbekannt*-Masse. Insgesamt ergibt sich nach Integration in das Occupancy Grid ein Wert von jeweils 0,46 für Belegtheit und Freiheit und von 0,08 für Unbekanntheit.

Vergleicht man das jeweilige Endergebnis mit dem zugehörigen initialen Wert, so kann man zwischen klassischem und evidenztheoretischen Occupancy Grid Mapping einen deutlichen Unterschied feststellen: Im klassischen Fall entspricht der Ergebniswert dem Ausgangswert, das heißt, es hat sich kein Informationsgewinn ergeben, obwohl zwei Messungen verarbeitet wurden. Im evidenztheoretischen Fall unterscheidet sich das Ergebnis deutlich vom Anfangswert: Es ist erkennbar, dass Informationen verarbeitet wurden, und es ist erkennbar, dass diese widersprüchlich waren. Möchte man das genaue Ausmaß des Widerspruchs quantifizieren, so kann man beispielsweise die Konfliktmasse k oder die Con-Metrik berechnen; in unserem Fall ergeben sich $k \approx 0{,}71$ und $Con \approx 0{,}53$.

Es mag der Eindruck entstanden sein, dass der evidenztheoretische Ansatz „besser" ist als der klassische. Die Dempster-Shafer-Theorie modelliert explizit Unwissenheit und erlaubt damit die Unterscheidung zwischen unsicherem und fehlendem Wissen. Zudem kann mithilfe eines Konfliktmaßes die Widersprüchlichkeit von Informationen gemessen werden; die resultierende Konfliktmasse kann darüber hinaus zur Erkennung von Fehlern im Occupancy Grid benutzt werden [9][8], die zum Beispiel aufgrund dynamischer Objekte entstanden sind [36] (vgl. Abschnitt 3.3.1). Tatsächlich ist sie damit theoretisch insbesondere besser zur Datenfusion geeignet und einige Arbeiten berichten von besseren Ergebnissen [41][36][39].

Allerdings ist zu berücksichtigen, dass in Anwendungen mit nur einem Sensor die Datenfusion eine weniger zentrale Rolle spielt[3]. Zudem ist die Berechnungskomplexität der Dempster-Shafer-Variante höher, was zu längeren Laufzeiten führt – dies kann für einige Einsatzszenarien ausschlaggebend sein. Es hängt also maßgeblich von den Rahmenbedingungen ab, was „besser" ist.

[3]Sie ist aber dennoch vorhanden, um die zeitliche Integration der Messwerte durchzuführen.

t	Vorgang	Klassisches OGM	Dempster-Shafer-OGM		
		belegt	belegt	frei	unbekannt
0	**Initialisierung** kein Vorwissen verfügbar	0,5	0	0	1
	Gesamtergebnis nach Integration in die Karte	0,5	0	0	1
1	**Verarbeitung der 1. Messung** die Zelle ist zu 84% belegt	0,84	0,84	0	0,16
	Gesamtergebnis nach Integration in die Karte	0,84	0,84	0	0,16
2	**Verarbeitung der 2. Messung** die Zelle ist zu 84% frei	0,16	0	0,84	0,16
	Gesamtergebnis nach Integration in die Karte	0,5	0,46	0,46	0,08

Abbildung 3.4: Beispielbasierter Vergleich der Varianten des Occupancy Grid Mappings (OGM).

3.3 Einsatz als automobiles Umfeldmodell

Dieser Abschnitt beschreibt abschließend, wie das Occupancy Grid Mapping (OGM) zur Erstellung eines zukunftsfähigen, automobilen Umfeldmodells verwendet werden kann.

3.3.1 Erfüllung der OGM-Voraussetzungen

In Abschnitt 3.1.1 klang an, dass für das Occupancy Grid Mapping zwei grundlegende Voraussetzungen erfüllt sein müssen: die genaue eigene Pose muss bekannt sein und das Umfeld muss statisch sein. Beide Voraussetzungen sind im automobilen Umfeld typischerweise nicht erfüllt, weshalb hier zusätzliche Schritte notwendig sind.

Das Occupancy Grid Mapping setzt die Kenntnis der exakten Pose des eigenen Fahrzeugs voraus – man spricht deswegen von Kartierung mit bekannten Posen (engl. *mapping with known poses*) [50] –, um die Messwerte aus dem im Allgemeinen mitbewegten Koordinatensystem des Sensors an der richtigen Stelle im üblicherweise erdfesten Occupancy Grid zu kartieren. Steht die Eigenpose nicht zur Verfügung, muss deshalb ein Verfahren zu deren Bestimmung eingesetzt werden. Ein genaues Studium der Eigenlokalisierung übersteigt den Rahmen dieser Arbeit, jedoch soll im Folgenden ein Überblick über die Methoden gegeben werden, die in der aktuellen Occupancy-Grid-Literatur dazu verwendet werden.

Die Positionsbestimmung über *GPS* ist eine seltener verfügbare, aber gegebenenfalls vorzuziehende Alternative zur Verwendung von Odometriedaten. Der Vorteil dieser Methode ist, dass es zu keiner Akkumulation des Lokalisierungsfehlers kommt. Wird allerdings ein günstiger GPS-Empfänger eingesetzt, kann bereits der Lokalisierungsfehler einer einzelnen Messung zu groß für eine ausreichend genaue Lokalisierung sein [49]. Stattdessen sollte deshalb ein hochpräzises Ortungssystem verwendet werden, wie beispielsweise ein so genanntes *GPS/INS-System* [17][48]: Ein günstiger GPS-Empfänger liefert regelmäßig abso-

lute Positionsangaben, während ein inertiales Navigationssystem (INS) die Zwischenwerte interpoliert. Leider stehen solche, sehr genauen Systeme in der Praxis oftmals nicht zur Verfügung.

Stehen *Odometriedaten* zur Verfügung, ist deshalb eine darauf basierende Schätzung der Eigenpose der übliche Lösungsansatz [54][33][42][52]. Oftmals werden die Daten vor der Schätzung mithilfe eines erweiterten Kalman-Filters gefiltert, um zuverlässigere Ergebnisse zu erhalten. In jedem Fall muss ein passendes Modell für die Bewegung des eigenen Fahrzeugs gewählt werden, das maßgeblich die Qualität der Schätzung bestimmt [43]. Ein großer Nachteil dieses Ansatzes ist, dass es – aufgrund der Ungenauigkeiten in den Odometriedaten und der Vereinfachungen des Bewegungsmodells – zu einem über die Zeit wachsenden Lokalisierungsfehler kommt [50]. Allerdings gibt es mehrere Möglichkeiten zur Lösung dieser Problematik.

Üblicherweise beschreibt ein automobiles Occupancy Grid einen Bereich konstanter Größe um das eigene, sich fortbewegende Fahrzeug. An einem Ende des Grids kommt dadurch immer der neu befahrene Teil der Umwelt hinzu, während am gegenüberliegenden Ende der ältere aufgrund der konstanten Gridgröße verworfen wird. Da sich der Lokalisierungsfehler nur auf den gerade kartierten Bereich auswirkt, lässt sich *über die Bereichsgröße* die Akkumulation des Lokalisierungsfehler so begrenzen, dass er sich in normalen Fahrsituationen nicht störend auswirkt. Zu beachten ist, dass es Situationen gegeben kann, in denen sich das Fahrzeug länger als üblich in einem Bereich aufhält und der Lokalisierungsfehler sich dadurch länger akkumuliert. Ein extremes Beispiel hierfür ist das permanente Kreisen in einem Kreisverkehr, der komplett in den Bereich um das Fahrzeug passt; da das Fahrzeug nie den Bereich wechselt, wird die Fehlerakkumulation überhaupt nicht beschränkt. Sind solche Fahrsituation jedoch selten oder das Ergebnis in diesem Fall ausreichend, ist dies eine gute und einfache Möglichkeit – vor allem auch, da sie sich aus der ohnehin notwendigen Begrenzung des repräsentierten Umfelds ergibt und dadurch keinen zusätzlichen Aufwand erfordert.

Eine alternative oder ergänzende Möglichkeit zur Beschränkung des Lokalisierungsfehlers ist die Einführung eines *Vergessens-Faktors* [12][17], durch den sich alle in einem Zeitschritt nicht aktualisierten Zellen mit einer vordefinierten Geschwindigkeit wieder ihrem initialen Zustand annähern. Da die Historie mithilfe des Vergessens-Faktors beschränkt wird, wird der sich akkumulierende Lokalisierungsfehler unter einem gewissen Wert gehalten. Mit diesem Verfahren lässt sich der Fehler feiner begrenzen als mit vorstehender Möglichkeit und es entfällt die eventuell unerwünschte Abhängigkeit von der Bereichsgröße, es erfordert aber die Integration des Faktors und die Wahl eines geeigneten Werts für ihn.

Die aufwendigste, aber im Allgemeinen auch genaueste Möglichkeit ist der Einsatz eines so genannten *Map-Matching-Algorithmus*'. Dabei wird die Umwelt zunächst erfasst und in einem temporären Occupancy Grid abgebildet. Durch Matching des temporären Grids mit dem eigentlichen Occupancy Grid lässt sich die Eigenlokalisierung verbessern und die Akkumulation des Lokalisierungsfehlers kompensieren. Ein Beispiel für solch ein Verfahren findet sich in [52]. Dieser Ansatz erfordert grundlegende Änderungen am Occupancy Grid Mapping und ist verhältnismäßig aufwendig; zudem ist er im Allgemeinen deutlich berechnungsintensiver. Allerdings ist das Ergebnis auch genauer als mit den beiden anderen Möglichkeiten.

Die zweite grundlegende Annahme des Occupancy Grid Mappings ist, dass alle Objekte über die Zeit keine Positionsveränderung erfahren und die Zellzustände damit zeitlich un-

abhängig sind. Dynamische Objekte werden als nicht existent angenommen, weshalb man von der Annahme einer statischen Welt spricht (engl. *static world assumption*) [50]. Diese Annahme ist notwendig, da dynamische Objekte zu widersprüchlichen Eingangsdaten und dadurch zu Fehlern im Occupancy Grid führen.

Wie diese aussehen können, zeigt Abbildung 3.5: Die Kartierung des vorausfahrenden Fahrzeugs führt im Occupancy Grid zu den gezeigten Artefakten. Deren charakteristisches Aussehen ist darauf zurückzuführen, dass zunächst an der aktuellen Position des Fahrzeugs eine hohe Belegtheitswahrscheinlichkeit kartiert wird, da es ein Hindernis darstellt. Hat sich das Fahrzeug weiterbewegt, wird seine ehemalige Position als frei erkannt, was schrittweise zu einem (teilweisen oder vollständigen) Rückgang der Belegtheitswahrscheinlichkeit führt. Für die meisten Anwendungen ist selbst das temporäre Vorhandensein solcher Artefakte nachteilig [31], da sie zumindest kurzzeitig ein statisches Hindernis unmittelbar vor dem eigenen Fahrzeug anzeigen. Dies könnte beispielsweise zu einem unnötigen und deswegen unerwünschten Ausweichmanöver führen.

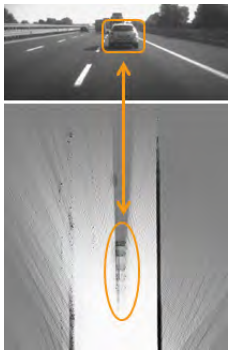

Abbildung 3.5: Typische Artefakte aufgrund eines vorausfahrenden Fahrzeugs.

Die automobile Welt ist meist sehr dynamisch – man ist umgeben von anderen Fahrzeugen, Radfahrern und Fußgängern. Um das Occupancy Grid Mapping trotzdem verwenden zu können, muss eine *Dynamikerkennung* und *-filterung* durchgeführt werden; das heißt, alle Messwerte der Umfeldsensorik, die durch dynamische Objekte verursacht wurden, müssen geeignet erkannt und gefiltert werden – so dass es zu keinen Artefakten kommt, da eine statische Welt simuliert wird. Dazu existieren mehrere Vorschläge, über die im Folgenden ein Überblick gegeben werden soll.

Die offensichtlichste Möglichkeit ist es, dynamische Objekte *bereits bei ihrer Detektion* von der Kartierung auszunehmen. Die Dynamikerkennung kann dabei auf unterschiedliche Arten erfolgen.

Sie ist insbesondere einfach, wenn ein Sensor zur Geschwindigkeitsmessung verfügbar ist, denn ein dynamisches Objekt kann leicht an seiner Geschwindigkeit erkannt werden. Dass diese Methode einfach und erfolgreich ist, zeigen die Ergebnisse vieler Arbeiten. Meist wird die in Radarmessungen enthaltene Doppler-Information genutzt, die bei Kenntnis der Eigengeschwindigkeit eine Bestimmung der Objektgeschwindigkeit erlaubt [49][27][7][53][20].

Eine weitere naheliegende Methode zur Dynamikerkennung basiert auf dem Umstand, dass dynamische Objekte aufgrund ihrer Bewegung zu einer abwechselnden Beobachtung einer Zelle als *frei* und *belegt* führen. Konkret erfolgt die Erkennung bei Eintreffen einer neuen Messung auf Basis des bisher erstellten Occupancy Grids. Liegt das für diese Messung

ursächliche Hindernis in einer bisher als *frei* angesehenen Zelle, so handelt es sich vermutlich um ein dynamisches Objekt. War die Zelle dagegen bisher schon *belegt*, so handelt es sich vermutlich um ein statisches Objekt. Diese Grundidee wurde von den nachfolgenden Arbeiten in Varianten umgesetzt.

In [4][5] wird nicht nur die Zelle berücksichtigt, die das Hindernis enthält, sondern ein größerer Bereich, um auch Unsicherheiten einzubeziehen. Außerdem speichert jede Zelle den Wert einer zweiten binären Zufallsvariable, die die Dynamik der Zelle klassifiziert (*statisch/dynamisch*) und zur Dynamikfilterung genutzt wird.

Auch in [52][7][53][20] werden Informationen über den Dynamikzustand der Zellen gespeichert. Hier wird dazu ein so genanntes *Dynamikgrid* verwaltet, das je Zelle die Anzahl dort bisher detektierter dynamischer Objekte speichert. Die Idee dahinter ist, dass Messungen in einem Bereich, in dem sich oft dynamische Objekte bewegen, mit hoher Wahrscheinlichkeit ebenfalls zu dynamischen Objekten gehören.

[36] nutzt mit dem Konfliktmaß der dort eingesetzten Dempster-Shafer-Theorie den gleichen Umstand aus, verwendet allerdings eine grundlegend andere Herangehensweise. Liegt der in einer Zelle auftretende Konflikt aufgrund der widersprüchlichen Beobachtungen über einem Schwellwert, wird dort ein dynamisches Objekt vermutet.

Die Dynamikerkennung basierend auf unterschiedlichen Beobachtungen, wie sie alle eben vorgestellte Arbeiten verwenden, ist eine gut geeignete und natürliche Vorgehensweise. Dabei ist insbesondere die Verwendung eines Konfliktmaßes im Rahmen der Evidenztheorie elegant, da dies eine natürliche, einfache und effiziente Realisierung ist und die Konfliktmasse k ohnehin für Dempsters Kombinationsregel berechnet werden muss. Auch die zellweise Speicherung von Dynamikinformationen, wie sie die übrigen Ansätze durchführen, erscheint sehr sinnvoll, da dies den Einsatz beliebig komplexer Algorithmen zur Dynamikerkennung und -filterung ermöglicht.

Die zweite Möglichkeit ist, dass alle Messungen unabhängig von der Dynamik *zunächst kartiert* und im Nachhinein gegebenenfalls wieder entfernt werden, indem die entsprechenden Zellen als *frei* eingetragen werden.

[54] sucht in einem Nachverarbeitungsschritt nach den charakteristischen Artefakten, die durch dynamische Objekte entstanden sind (siehe oben), und entfernt sie gegebenenfalls nachträglich. Da die Artefakte erst nach einigen Zeitschritten die charakteristische Form annehmen, erfolgt die Detektion allerdings verzögert – das Occupancy Grid ist bis dahin fehlerhaft.

In [48] bilden unterschiedliche Beobachtungen für eine einzelne Zelle die Grundlage der Dynamikerkennung, wie es eben bereits beschrieben wurde, allerdings wird mit *unbekannt* ein weiterer Zustand unterschieden. Nicht-*freie* Zellen werden zu potentiellen dynamischen Objekten zusammengefasst und mit einem erweiterten Kalman-Filter getrackt. Erst wenn ihre Dynamik auf diese Weise bestätigt wurde, werden sie aus dem Occupancy Grid entfernt – auch hier ist das Occupancy Grid bis dahin fehlerhaft. Da selbst eine nur vorübergehende fehlerhafte Darstellung in vielen Anwendungen störend ist, sind andere Methoden dieser und der vorangehenden im Allgemeinen vorzuziehen.

Auch für die Erstellung eines so genannten *Differenzgrids*, das die Unterschiede zwischen dem aktuellen Occupancy Grid und dem des vorherigen Zeitschritts enthält, ist zunächst eine Kartierung notwendig. Anders als in den beiden vorherigen Ansätzen kann die Dynamikfilterung aber noch im aktuellen Zeitschritt durchgeführt werden. Die zugrundeliegende Annahme dieser Methode ist, dass dynamische Objekte zu Unterschieden im Differenzgrid

führen und dadurch ihre Positionen erkennbar sind, wodurch sie aus dem Occupancy Grid entfernt werden können. Da allerdings auch Messfehler zu Unterschieden führen können und dynamische Objekte sich im Allgemeinen über mehrere Zellen erstrecken, ist eine Umsetzung deutlich komplexer. In [33][31] werden potentielle dynamische Objekte zum Beispiel durch Segmentierung ermittelt, mithilfe eines Tracking-Algorithmus' validiert und erst dann aus dem Grid entfernt; zudem können sie später ohne Informationsverlust wieder eingefügt werden, sollte sich die Annahme eines dynamischen Objekts als falsch herausstellen. Dieser Ansatz eignet sich gut zur Verwendung in einem hybriden Umfeldmodell, da dort ohnehin ein Objekttracking durchgeführt wird. Die Möglichkeit zur nachträglichen Korrektur schafft zudem Robustheit. Andererseits ist die Notwendigkeit zu einem solchen Tracking aufgrund des damit verbundenen Aufwands und der Objektbasiertheit auch als Nachteil für reine Occupancy-Grid-Ansätze zu sehen.

Alternativ zu den bisher vorgestellten Möglichkeiten kann die Dynamikfilterung *vorsorglich* erfolgen, wie beim *verzögerten Kartografieren* [31]. Es basiert auf der Annahme, dass dynamische Objekte im Gegensatz zu statischen Objekten nicht wiederholt an derselben Stelle detektiert werden. Objekte werden deshalb erst kartiert, wenn sie mindestens zweimal erfasst wurden. Zu beachten ist, dass in der Praxis dadurch auch langsame dynamische Objekte kartiert werden, da in diesem Fall entgegen der Annahme eine weitere Messung an ähnlicher Stelle erfolgt. Außerdem erfolgt die Kartierung – wie der Name schon sagt – um einen Zeitschritt verzögert.

Allgemeine Stärken und Schwächen der vorgestellten Ansätze zur Dynamikerkennung und -filterung wurden bereits genannt. Andererseits hängt die Beantwortung der Frage, welche Methode vorzuziehen ist, auch stark von der Ausgangssituation ab. Davon abgesehen wäre es sehr gut denkbar, mehrere der vorgestellten Methoden zu kombinieren, um ein robusteres Resultat zu erhalten. Im Rahmen der Dempster-Shafer-Theorie erscheint zum Beispiel eine gemeinsame Verwendung von einem Konfliktmaß und zellweise gespeicherten Dynamikinformationen eine vielversprechende Kombination.

3.3.2 Berücksichtigung mehrerer Sensoren

Das Occupancy Grid Mapping geht zunächst von einem einzelnen Sensor aus. Da multisensorielle Ansätze zur Erstellung automobiler Umfeldmodelle zunehmend an Bedeutung gewinnen (vgl. Abschnitt 2.2.2), wird im Folgenden besprochen, wie man es so auf mehrere Sensoren erweitern kann, dass die Daten beliebiger, unterschiedlicher Sensoren gleichberechtigt eingebracht werden können. Dass die dabei notwendige Datenfusion mit einigen Herausforderungen verbunden ist, klang bereits in Abschnitt 2.2.2 an; dies wird nun im Kontext des Occupancy Grid Mappings konkretisiert.

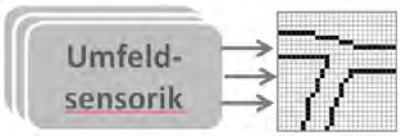

Abbildung 3.6: Naiver Ansatz zur Kartierung mehrerer Sensoren.

Der naive Ansatz [19] besteht darin, die Messwerte aller Sensoren direkt in einem einzigen Occupancy Grid zu kartieren (siehe Abbildung 3.6). Bisher, bei einem Sensor, erfolgte

die zeitliche Integration von Messungen in das Occupancy Grid über den binären Bayes-Filter bzw. Dempsters Kombinationsregel; bei nun mehreren Sensoren ist einfach eine mehrmalige, sequentielle Anwendung möglich [28]. Auf diese Art werden zunächst die Daten des ersten Sensors in das Occupancy Grid integriert, dann durch erneute Anwendung die Daten des zweiten Sensors und so weiter. Die Reihenfolge der Sensoren ist dabei ohne Belang, da beide Aktualisierungsgleichungen kommutativ sind [39][28]. Handelt es sich um gleichartige und gleichgetaktete Sensorik, liegt danach bereits das gewünschte, fusionierte Ergebnis vor [17]. Andernfalls sind jedoch weitere Schritte notwendig, da zwei Probleme auftreten [50]:

1. Unterschiedliche Sensoren besitzen im Allgemeinen eine unterschiedliche Messfrequenz. Bringt man die Messungen der Sensoren direkt in der jeweiligen Frequenz ein, ergibt sich eine Verzerrung des Gesamtergebnisses hin zu den hochfrequenten Sensoren, die die Darstellung dominieren. Dies ist in der Regel unerwünscht.

2. Unterschiedliche Sensoren detektieren möglicherweise unterschiedliche Hindernisse; beispielsweise aufgrund ihrer verschiedenen Messprinzipien. Erkennt ein Sensor ein Hindernis in einer bestimmten Zelle, zwei andere Sensoren jedoch nicht, wird der einzelne Sensor überstimmt; auch dies ist meist nicht das erwünschte Ergebnis.

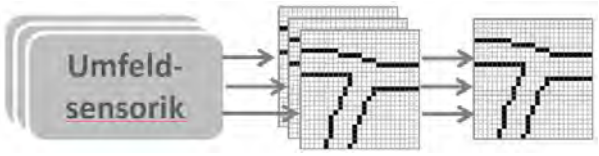

Abbildung 3.7: Üblicher Ansatz zur Kartierung mehrerer Sensoren.

Das erste Problem – die unterschiedliche Messfrequenz – wird typischerweise gelöst [17][50], indem für jeden Sensor zunächst ein eigenes Occupancy Grid erstellt wird (vgl. Abbildung 3.7). Darin erfolgt die zeitliche Integration der Messwerte zunächst für jeden Sensor allein, bis von jedem wenigstens ein Messwert integriert wurde. Erst dann werden diese sensorspezifischen Occupancy Grids mit einer geeigneten Funktion kombiniert, wodurch die unerwünschte Abhängigkeit von der Messfrequenz entfällt.

Diese Kombinationsfunktion kann gleichzeitig auch das zweite Problem – die Detektion verschiedener Hindernisse – beseitigen, indem sie die unterschiedliche Hindernissensitivität der Sensoren bei deren Kombination berücksichtigt. Bei klassischen Occupancy Grids kann dies beispielsweise über das Prinzip der *Bayes'schen Fusion* [28] oder über das *Gesetz von De Morgan* geschehen [50]; eine pessimistischere Möglichkeit ist es, für jede Zelle das Maximum aus allen Occupancy Grids zu verwenden [50]. Die bereits angesprochenen *Linear Opinion Pools* [1] sind eine weniger übliche, aber sehr interessante Alternative. Da sie zur Modellierung der Zuverlässigkeit eine Gewichtung verwenden, erlauben sie insbesondere eine genauere Differenzierung.

Vor allem bietet sich zur Lösung des zweiten Problems jedoch die Verwendung der Dempster-Shafer-Theorie [15] an, da dadurch je Sensor nicht nur der eigentliche Messwert, sondern auch die Zuverlässigkeit des Sensors und sein Vertrauen in die Messung sehr gut berücksichtigt werden können. Es ist dadurch ausreichend, Dempsters Kombinationsregel mehrmals, sequentiell anzuwenden (wie eben beschrieben) [28]. Dies ist einer der Gründe, warum evidenztheoretische Occupancy Grids in letzter Zeit so populär sind.

Abschließend sei angemerkt, dass ein großer Teil der aktuellen Occupancy-Grid-Literatur nach wie vor nur einen einzelnen Sensor verwendet. Kommen mehrere Sensoren zum Einsatz, werden deren Daten meist auf eine sensorspezifische Art und Weise fusioniert. In [46] werden zum Beispiel die Daten eines Lasersensors und einer Kamera in einem darauf spezialisierten Verfahren kombiniert. Beispiele für die von uns angestrebte gleichberechtigte Fusion der Daten beliebiger, unterschiedlicher Sensoren finden sich jedoch kaum; erwähnenswerte Ausnahmen sind [16][17][22]. Alle drei Arbeiten verwenden das gleiche Vorgehen, das dem eben vorgeschlagenem entspricht. Die erfassten Daten werden zunächst getrennt in sensorspezifischen Occupancy Grids akkumuliert und diese erst nach mehreren Zeitschritten fusioniert. Dazu verwenden sie die Dempster-Shafer-Theorie mit Dempsters Kombinationsregel.

3.3.3 Einbeziehung künftiger Anforderungen

Die erwarteten Anforderungen an ein künftiges Umfeldmodell wurden bereits in Abschnitt 2.3.2 besprochen: Es sollte auf den Daten mehrerer Sensoren basieren, einen hybriden Ansatz verwenden und eine Entkopplung zwischen Sensorik und Aufgaben schaffen. Beim Einsatz eines Occupancy Grids als Umfeldmodell sollte darauf geachtet werden, dass es diesen zukünftigen Anforderungen gerecht wird, um eine längerfristige Verwendungsperspektive zu besitzen. Wie dies möglich ist, soll nun abschließend kurz thematisiert werden.

Wie die Daten unterschiedlicher Sensoren in ein Occupancy Grid integriert werden können, wurde im vorherigen Abschnitt bereits gezeigt.

Die zweite Anforderung – ein hybrider Ansatz – kann mit einem Occupancy Grid allein nur teilweise realisiert werden: Es eignet sich zwar sehr gut für den kartenbasierten Teil [22], den objektbasierten kann es allerdings nicht darstellen[4]. Für ein hybrides Umfeldmodell ist also die Ergänzung durch ein separates objektbasiertes Modell notwendig. Die beiden Modellteile sollten über geeignete Schnittstellen Informationen untereinander austauschen, um voneinander zu profitieren, wie zum Beispiel in [4][5][16]. Für den objektbasierten Teil eignet sich als eine solche Schnittstelle gut die Objektliste, da sich diese bereits für rein objektbasierte Umfeldmodelle als Quasi-Standard durchgesetzt hat und deshalb von vielen Sensoren als Ausgabe erzeugt wird [22]. Für kartenbasierte Umfeldmodelle gibt es dagegen noch keine übliche Schnittstelle. Allerdings eignet sich die in einer einzelnen Occupancy-Grid-Zelle gespeicherte Evidenz gut zur direkten Verwendung, da sie – anders als unverarbeitete Messwerte oder aus dem Occupancy Grid extrahierte Merkmale – eine normalisierte und sensorunabhängige Ergebnisrepräsentation darstellt [22]. Zusammen mit einem objektbasierten Umfeldmodell und geeigneten Schnittstellen kann ein Occupancy Grid also auch die Anforderung eines hybriden Modells erfüllen. Hybride Ansätze, in denen die beiden Modellteile isoliert voneinander verwaltet werden, sind ebenfalls möglich (z.B. [52][7][3]) – sie scheinen aber weniger geeignet, da hier wertvolle Synergieeffekte ungenutzt bleiben.

Aufgrund der eben beschriebenen Sensorunabhängigkeit der Zellwerte kann auch die dritte und letzte Anforderung erfüllt werden: Sie schaffen die gewünschte Entkopplung zwischen Sensorik und Aufgaben, wodurch das Occupancy Grid als virtueller Umfeldsensor dienen kann [39].

[4]Es existieren Varianten des Occupancy Grid Mappings, die eine objektbasierte Modellierung integrieren [10][11][42]; diese scheitern in der Praxis aber an den Echtzeitanforderungen.

Kapitel 4

Praktische Umsetzung

Bisher wurden die theoretischen Grundlagen und etliche Ansätze aus der Literatur besprochen, in diesem Kapitel soll nun die praktische Umsetzung vorgestellt werden: die eigene Implementierung[1] eines automobilen Occupancy Grids.

4.1 Vorüberlegungen & Überblick

In diesem Abschnitt werden zunächst die Rahmenbedingungen der vorliegenden Arbeit besprochen, darauf aufbauend eine Zielstellung für die Implementierung formuliert und ein grober Überblick über deren Bestandteile gegeben.

4.1.1 Rahmenbedingungen

Ausgangspunkt für diese Arbeit ist die Aufgabe, ein automobiles Occupancy Grid zu erstellen. Dazu stehen exemplarische Daten und das Framework ADTF zur Verfügung. Ferner soll die Implementierung *C++* als Programmiersprache, *Qt* für Grafik und *CMake* als Buildsystem nutzen.

Das Framework ADTF wird später in Abschnitt 4.3.1 vorgestellt. Es dient in dieser Arbeit als eine Art „Testumgebung“: Im Rahmen einer Simulation liest es die nachfolgend beschriebenen Daten in C++-Strukturen und übergibt diese als Eingabe an unsere Implementierung, die daraus als Ausgabe ein Occupancy Grid erzeugt.

Die bereitgestellten Daten stammen aus einem FORWISS-Projekt. Sie wurden im Rahmen mehrerer Messfahrten in unterschiedlichen Umgebungen mit einem mit Sensorik ausgestatteten Fahrzeug zur späteren Verwendung aufgezeichnet. Sie umfassen:

Vorverarbeitete Messwerte eines Mittelbereichsradarsensors. Der Radarsensor besitzt einen Öffnungswinkel von 28 Grad und eine spezifizierte Reichweite von 20 Meter – tatsächlich sind aber auch Messwerte in Entfernungen deutlich über 100 Meter vorhanden. Er befindet sich im vorderen Teil des Fahrzeugs etwa mittig auf einer Höhe von knapp 70 Zentimeter und arbeitet mit einer Messfrequenz von 25 Hertz. Im Wesentlichen stellt er eine Liste mit maximal 20 Hindernissen, so genannten *Targets* bereit, wobei zu jedem Target die relative Geschwindigkeit und Position (gegeben durch radiale Entfernung und Winkel) angegeben werden.

[1] Der Quellcode und die zugehörige Dokumentation befinden sich auf der DVD im Anhang.

Vorverarbeitete Messwerte eines Lasersensors. Der Lasersensor besitzt einen Öffnungswinkel von 110 Grad und eine Reichweite von 200 Meter. Es handelt sich um einen Mehrebenenlaser mit vier Ebenen und einem vertikalen Gesamtöffnungswinkel von 3 Grad. Seine Messfrequenz beträgt 12,5 Hertz und er ist nahe der Fahrzeugspitze etwa mittig auf einer Höhe von zirka 60 Zentimeter angebracht. Er stellt eine Liste intern getrackter Objekte bereit. Im Wesentlichen liefert er zu jedem Objekt eine Typklassifikation (z.B. PKW, LKW, ...), die sensorrelative Position in Polarkoordinaten, Schätzungen für Größe und absolute Geschwindigkeit sowie die einzelnen Messpunkte. Teilweise existiert zudem eine *Bounding Box*, die alle Messpunkte umschließt und durch ihre Position, Größe und Orientierung gegeben ist.

Gefilterte Messwerte der Eigenzustandssensorik. Sie umfassen im Wesentlichen die longitudinale Geschwindigkeit, die longitudinale und laterale Beschleunigung, die Gierrate und einen Zeitstempel.

Video. Es stammt von einer Schwarz-Weiß-Kamera und zeigt den Bereich vor dem Fahrzeug. Im Rahmen dieser Arbeit wird es als *Referenzvideo* verwendet, das heißt, es dient ausschließlich der Visualisierung der Fahrzeugumgebung zu Referenzzwecken.

Konfigurationsdaten. Sie enthalten Informationen über das Fahrzeug selbst (insbesondere dessen Länge und Breite) und über dessen Sensorik (Art und jeweils Anzahl). Für den Radar- und Lasersensor sind insbesondere jeweils der Operationsmodus sowie die intrinsischen (z.B. Reichweite, Öffnungswinkel, ...) und extrinsischen (insbesondere Rotation und Translation) Parameter gegeben.

Bei der Verwendung dieser Daten ergeben sich eine Reihe von Herausforderungen, die erst im nächsten Kapitel besprochen werden (siehe Abschnitt 5.1). Sofern sie die nachfolgenden Ausführungen betreffen, werden sie auch hier thematisiert.

4.1.2 Zielformulierung & Abgrenzung

In der Literatur beschriebene Ansätze sind typischerweise maßgeschneidert auf ein einzelnes Einsatzszenario, wie beispielsweise das eben beschriebene. Sie unterscheiden sich dadurch in vielerlei Hinsicht, zum Beispiel bezüglich:

- Variante (z.B. klassisch, evidenztheoretisch, ...),
- Fusionsoperationen (z.B. binärer Bayes-Filter, Dempsters Kombinationsregel, ...),
- Sensorik (z.B. Mehrebenen-Laserscanner, Fernbereichsradar, ...),
- inversen Sensormodellen (z.B. Raycasting, zweidimensionale Gaußfunktion, ...),
- Dynamikerkennung und -filterung (z.B. Differenzgrid, basierend auf Konfliktmaß, ...),
- internen Grids (z.B. polares Messwertgrid, keine Hilfskarten, ...),
- Gridbezugssystemen (z.B. erdfest, fahrzeugfest, ...),
- Griddarstellungen (z.B. probabilistisch in Graustufen, diskret mit drei Werten, ...),
- Zellwerten (z.B. Belegtheitswahrscheinlichkeit, Massen und Geschwindigkeit, ...) und
- darauf aufbauenden Anwendungen (z.B. Straßenranderkennung, Routenplanung, ...).

Diese übliche Maßschneiderung auf einen einzelnen, konkreten Anwendungsfall erfolgt insbesondere aus Gründen der Einfachheit und Effizienz: Die Wahl der geeigneten Vorgehensweise ist von etlichen anwendungsspezifischen Faktoren abhängig, unter anderem von

den Rahmenbedingungen (z.B. zur Verfügung stehende Sensorik), der Zielstellung (z.B. darauf aufbauende Anwendung) und der Abwägung zwischen Geschwindigkeit und Genauigkeit. Eine Bewertung dessen, was der „bessere" Ansatz ist, ist also oft – und wenn überhaupt – nur im Kontext eines konkreten Anwendungsszenarios möglich. Eine derartige Spezialisierung beschränkt allerdings auch ganz erheblich die Wiederverwendbarkeit.

Diese Arbeit stellt mit der *OG-Bibliothek* (wobei *OG* für *Occupancy Grid* steht) eine Bibliothek bereit, die durch ihre Anpassbarkeit künftig zur Erzeugung automobiler Occupancy Grids in ganz unterschiedlichen Szenarien verwendet werden kann. Der Quellcode der OG-Bibliothek und die zugehörige Dokumentation befinden sich auf der DVD im Anhang.

Mit der OG-Bibliothek erstellte Occupancy Grids repräsentieren einen zweidimensionalen, quadratischen Ausschnitt um das eigene Fahrzeug und bestehen aus gleichgroßen und gleichartigen Zellen. Ansonsten ist die OG-Bibliothek so weit wie möglich anpassbar an einen konkreten Anwendungsfall; zum Beispiel an konkrete Sensorik mit deren Messdaten, bestimmte Zellinhalte und der Assistenzfunktion, die das Occupancy Grid nutzen möchte. Sie erlaubt die Verwendung möglichst vieler unterschiedlicher Vorgehensweisen, da es im allgemeinen, anwendungsunabhängigen Fall keine klar „beste Methode" gibt.

Liegt ein konkreter Anwendungsfall vor, in dessen Rahmen die OG-Bibliothek eingesetzt werden soll, kann sie daran angepasst werden und dabei die geeignetste Methodik verwendet werden. Die für diese Arbeit bereitgestellten Daten und die Verwendung von ADTF – wie sie im vorherigen Abschnitt beschrieben wurden – sind als Beispiel für einen solchen Anwendungsfall zu verstehen, an den die OG-Bibliothek angepasst wird.

Die OG-Bibliothek berücksichtigt zudem die Erfordernisse künftiger, automobiler Umfeldmodelle, wie sie in Abschnitt 3.3 besprochen wurden: sie erlaubt den gleichberechtigten Einsatz beliebiger, unterschiedlicher Sensoren, bietet die Möglichkeit zur Kombination mit einem objektbasierten Ansatz zu einem hybriden Modell und entkoppelt Assistenzaufgaben und Sensorik voneinander. Wenngleich Echtzeitfähigkeit kein primäres Ziel ist, ist deren Erreichen je nach Anwendungsszenario möglich.

4.1.3 Überblick

Wie eben bereits dargestellt wurde, wird die OG-Bibliothek im Rahmen dieser Arbeit beispielhaft zusammen mit dem Framework ADTF (vgl. späteren Abschnitt 4.3.1) verwendet. Die praktische Umsetzung besteht deshalb aus zwei eigenständigen Teilen, die miteinander interagieren, wie es Abbildung 4.1 zeigt.

Abbildung 4.1: Das Zusammenspiel zwischen ADTF und OG-Bibliothek.

Die OG-Bibliothek führt das Occupancy Grid Mapping durch und stellt damit den Kern der Funktionalität bereit. Mit der Klasse OGBuilder ist eine Schnittstelle verfügbar, über die jegliche Interaktion läuft. Um die OG-Bibliothek in ADTF verwenden zu können, wird mit dem OGFilter zusätzlich eine eigene ADTF-Komponente zur Verfügung gestellt, die der Integration der OG-Bibliothek in ADTF dient. Der OGFilter überreicht dem OGBuilder die aktuellen Messwerte der Sensorik, während er im Gegenzug das daraus erstellte, aktuelle Occupancy Grid und die aktuelle Schätzung des Eigenzustands erhält.

Die Abläufe innerhalb des OGBuilder und OGFilter sowie die Einbindung in ADTF werden in den nächsten Abschnitten genauer besprochen. An dieser Stelle soll ein grober Überblick über den Gesamtablauf gegeben werden:

1. Zu Beginn wird der OGFilter in die ADTF-Konfiguration eingebunden. Über dessen graphische Benutzeroberfläche werden durch den Anwender die Größe des Grids und einer einzelnen Zelle (die so genannte *Auflösung*) festgelegt und die ADTF-Simulation gestartet.

2. Während der Simulation empfängt der OGFilter asynchron die aufgezeichneten Messwerte der Eigenzustands- und Umfeldsensorik, die andere ADTF-Komponenten in Echtzeit aus einer Datei lesen, und leitet sie synchron an den OGBuilder weiter. Dieser steuert den Ablauf innerhalb der OG-Bibliothek, die aus den erhaltenen Daten ein aktualisiertes Occupancy Grid erstellt.

3. Bei Bedarf kann der OGFilter jederzeit das aktuelle Occupancy Grid oder die aktuelle Eigenzustandsschätzung anfordern, beispielsweise zur periodischen Aktualisierung der graphischen Anzeige in ADTF oder zur Weitergabe an andere ADTF-Komponenten, die dann eine Weiterverarbeitung durchführen (z.B. eine Straßenranderkennung). Optional kann das Occupancy Grid auch als Grafikdatei gespeichert werden.

4. Die Simulation endet, sobald sie durch den Anwender abgebrochen wird oder das Ende der Datei mit den aufgezeichneten Messdaten erreicht ist. Bis dahin wiederholt sich der Ablauf immer wieder ab Schritt 2.

4.2 Die OG-Bibliothek

Dieser Abschnitt beschäftigt sich ausführlich mit der OG-Bibliothek, die den Kern der praktischen Arbeit darstellt. Die nachfolgende Beschreibung der OG-Bibliothek verfolgt zwei Ziele:

1. Sie möchte die Architektur der OG-Bibliothek darstellen, indem die einzelnen Bestandteile, ihr jeweiliger Zweck sowie ihr Zusammenspiel besprochen werden. Dabei wird auch darauf eingegangen, welche Anpassungen konkret für die Verwendung im Rahmen eines eigenen Anwendungsfalls notwendig sind.

2. Sie möchte die Anpassungen anhand eines konkreten Anwendungsfalls zeigen. Dieser dient vor allem Demonstrations- und Evaluierungszwecken.

Insgesamt soll ein Eindruck vermittelt werden, wie eine Benutzung der OG-Bibliothek im eigenen Programm aussehen kann. Deshalb wird nicht klassenweise, sondern nach Themen geordnet vorgegangen, und nur in Ausnahmefällen auf die konkrete Verwendung oder Algorithmik einzelner Methoden eingegangen. Für derartige, weiterführende Informationen wird auf den Quellcode und die Dokumentation der OG-Bibliothek verwiesen, die sich auf der DVD im Anhang befinden.

4.2.1 Allgemeine Hinweise

Bevor im Folgenden die einzelnen Komponenten der OG-Bibliothek detailliert beschrieben werden, sollen hier zunächst grundlegende Hinweise zur Verwendung der OG-Bibliothek insgesamt gegeben werden, die auch zum Verständnis der nachfolgenden Beschreibung wichtig sind.

4.2.1.1 Anpassung an einen Anwendungsfall

Möchte man die OG-Bibliothek in seinem eigenen Programm verwenden, so muss man sie zuerst an seinen Anwendungsfall anpassen. Dies erfolgt prinzipiell in fünf Schritten:

1. Die von den Anpassungen betroffenen bibliotheksinternen Interfaces werden implementiert (vgl. nächster Abschnitt); je nach gewünschter Funktionalität sind dies unterschiedliche.

2. Falls zusätzlich eigene Komponenten benötigt werden, werden diese implementiert.

3. Die Implementierungen aus den beiden vorangegangen Punkten werden integriert. Dies erfolgt im Wesentlichen durch Anpassung der Typedefs in den Headern GridDefinitions und/oder DataDefinitions, der Klasse OGBuilder und/oder der Struktur SensorData.

4. Der Quellcode der OG-Bibliothek inklusive der eigenen Erweiterungen wird neu kompiliert.

5. Man erhält die angepasste OG-Bibliothek, die man in sein eigenes (externes) Programm einbinden und dort zur Erstellung von Occupancy Grids verwenden kann. Eine Instanz der Klasse OGBuilder stellt dabei die Schnittstelle zwischen dem eigenen Programm und der OG-Bibliothek dar.

Schritt 3 bedarf einer näheren Erläuterung. Die OG-Bibliothek arbeitet mit Konfigurationsdaten für das Fahrzeug und die Sensoren sowie mit Eigenzustands- und Umfelddaten. All diese Daten sind anwendungsspezifisch, das heißt von Anwendungsfall zu Anwendungsfall unterschiedlich. Trotzdem muss sich die Implementierung der OG-Bibliothek auf konkrete Strukturen festlegen, mit denen sie diese Daten repräsentiert.

Um diese Aufgabe zu lösen, setzt die OG-Bibliothek unter anderem so genannte *Typedefs* ein, mit deren Hilfe bestehenden Datentypen alternative Bezeichner zugewiesen werden können. In unserem Fall werden die anwendungsspezifischen Strukturen umbenannt in die in der OG-Bibliothek üblichen, so dass die Änderung an einer einzelnen Stelle ausreichend ist und viele restliche Teile der OG-Bibliothek unverändert bleiben können. Für die Strukturen der Sensor- und Konfigurationsdaten befinden sich alle Typedefs gesammelt im Header

DataDefinitions. Mit dem Header GridDefinitions existiert ein analoges Konzept für die zu verwendenden Grids und Zellen.

Durch die Verwendung der Typdefs sind Änderungen nur an denjenigen Teilen der Implementierung notwendig, die auf den anwendungsspezifischen Strukturen arbeiten müssen. Um auch davon soweit wie möglich zu abstrahieren, verwendet die OG-Bibliothek eine Reihe von „Interfaces" (vgl. nächster Abschnitt), die ein leichtes Austauschen der betroffenen Programmteile ermöglichen. Nur wenn neue Sensoren hinzugefügt oder entfernt werden, sind zusätzlich kleinere, offensichtliche Änderungen an der Klasse OGBuilder und der Struktur SensorData notwendig.

Mit den in Abschnitt 4.1.1 beschriebenen Rahmenbedingungen existiert nicht nur ein exemplarisches Anwendungsszenario, sondern es wurde bekanntermaßen auch eine dazu passende, beispielhafte Implementierung erstellt. Diese wird im Folgenden zu Zwecken der Demonstration immer wieder herangezogen. Zum Einen besteht sie aus einer entsprechend angepassten OG-Bibliothek; zum Anderen steht mit dem OGFilter auch ein konkretes Beispiel für ein externes Programm bereit, das die OG-Bibliothek zur Erstellung von Occupancy Grids innerhalb des Frameworks ADTF benutzt (vgl. Abschnitt 4.3.2).

Die an diesen Anwendungsfall angepasste OG-Bibliothek verwendet eine evidenztheoretische Modellierung (vgl. Abschnitt 3.2.2), da auch mit widersprüchlichen Messdaten zu rechnen ist und sich über den Konflikt dynamische Objekte erkennen und eliminieren lassen. Die Implementierungen der grundlegenden evidenztheoretischen Strukturen (z.B. Massen) und Funktionen (z.B. Dempsters Kombinationsregel) befinden sich in der Datei DempsterShaferBasics.cpp und dem zugehörigen, gleichnamigen Header.

Diese Datei ist ein Beispiel für die Implementierung eigener, zusätzlicher Komponenten im Rahmen der Anpassung (Schritt 2 obiger Anpassungsmaßnahmen). Beispiele für Interface-Implementierungen (Schritt 1) werden in den kommenden Abschnitten ausführlich beschrieben. Das Vorgehen zur Durchführung der Kompilierung und Einbindung (Schritte 4 und 5) bedarf keiner Erklärungen. Wenngleich auch das Vorgehen zur Integration der eigenen Anpassungen (Schritt 3) bei Begutachtung des im Anhang befindlichen Quellcodes schnell klar werden sollte, werden im weiteren Verlauf als Hilfestellung entsprechende Anmerkungen gemacht.

4.2.1.2 Anmerkungen zur Implementierung

Im Folgenden wird an etlichen Stellen von *Interfaces* (oder dt. *Schnittstellen*) gesprochen, obwohl es ein solches Konzept in C++ nicht gibt – zumindest nicht in dem Sinne, wie man es beispielsweise aus Java kennt. Allerdings kann man ein ähnliches Verhalten erreichen, indem man eine Klasse als „Interface" verwendet, die bis auf ihren Destruktor nur virtuelle Klassenfunktionen ohne zugehörige Definitionen (sog. *pure virtual member functions*) besitzt. Eine andere Klasse implementiert das „Interface", indem sie von der Interface-Klasse erbt und konkrete Definitionen für deren Funktionen hinzufügt.

Diese Art Interface – im Rahmen der OG-Bibliothek leicht erkennbar am führenden „I" im Bezeichner (z.B. IMainGrid) – ist immer gemeint, wenn im weiteren Verlauf von Schnittstellen gesprochen wird, die implementiert werden oder werden sollen. Dabei ist zu beachten, dass Interfaces in der OG-Bibliothek nur existieren, um eine Hilfestellung für die eigenen Anpassungen zu geben: So ist leicht ersichtlich, welche Funktionen benötigt und deshalb implementiert werden müssen. Ob das jeweilige Interface tatsächlich implementiert

wird oder nur die notwendigen Funktionen bereitgestellt werden, ist nicht relevant.

Wichtig ist zudem der Hinweis, dass die gesamte OG-Bibliothek nicht für Nebenläufigkeit ausgelegt ist. Es liegt in der Zuständigkeit des externen Verwender-Programms, für einen entsprechend synchronisierten Zugriff auf die OG-Bibliothek in asynchronen Umgebungen zu sorgen. Die Ausführung aller Operationen erfolgt im Aufrufer-Thread. Die OG-Bibliothek unterstützt keine Nebenläufigkeit, da dies nicht sinnvoll ist: Zum Einen müssen die Daten ohnehin sequentiell eingearbeitet werden, zum Anderen kann die Erstellung des Occupancy Grids und die Einarbeitung neuer Daten nicht gleichzeitig stattfinden.

Ferner sei angemerkt, dass sich alle zur OG-Bibliothek gehörenden Dateien im eigenen Namensraum (engl. namespace) OGLIBRARY befinden und dass die OG-Bibliothek explizite Instanziierung verwendet. Letztere wird immer in einer eigenen Datei durchgeführt, die die Bezeichnung *-EXPANDED.cpp erhält, wobei * für den Originalnamen steht (z.B. CLASS.cpp und CLASS-EXPANDED.cpp).

4.2.2 Grundlegender Überblick

Einleitend wird die grundlegende Modellierung der OG-Bibliothek vorgestellt. Dabei werden insbesondere deren wichtigste Bestandteile eingeführt: die Mapper und Grids, inklusive des verwendeten Koordinatensystems mit seiner speziellen Posenmodellierung. Außerdem wird ein Überblick über den prinzipiellen Ablauf innerhalb der OG-Bibliothek gegeben.

4.2.2.1 Mapper & Grids

Die Klasse OGBUILDER nimmt eine zentrale Rolle ein, da sie die Schnittstelle zum externen Programm darstellt und den Ablauf innerhalb der OG-Bibliothek steuert. Der OGBUILDER erzeugt im Rahmen der eigenen Instanziierung Instanzen von allen benötigten Grids und Mapper. Diese erfüllen ebenfalls wichtige Aufgaben innerhalb der OG-Bibliothek und sollen deshalb bereits zu Beginn kurz eingeführt werden; eine genauere Beschreibung folgt in späteren Abschnitten.

Die OG-Bibliothek verwendet zwei Arten so genannter *Mapper*:

- Die *Sensormapper* dienen der Kartierung der Umfelddaten. Es existiert dazu genau ein Sensormapper je Umfeldsensor. Jeder Sensormapper kartiert die Daten eines bestimmten, ihm zugeordneten Umfeldsensors.

- Mit dem *Fahrzeugmapper* existiert zusätzlich ein Mapper für das Fahrzeug selbst. Er dient vor allem der Schätzung des Eigenzustands auf Basis der Daten der Eigenzustandssensorik; insbesondere bestimmt er dabei die Pose des eigenen Fahrzeugs. Manche Fahrzeugmapper kartieren darüber hinaus die Fläche des Eigenfahrzeugs, da dies für einige Anwendungen lohnen kann.

Außerdem werden in der OG-Bibliothek drei unterschiedliche Grids verwendet. Jedes dieser Grids besitzt eine bestimmte Aufgabe, die es durch seine Architektur optimal unterstützt:

- Das *Messwertgrid* (MG) stellt eine effiziente Datenstruktur zur Kartierung neuer Sensordaten bereit. In dieses Grid tragen die Mapper die aktuellen Messwerte der Umfeldsensorik ein.

– Das *Hauptgrid* (HG) bietet eine zur zeitlichen Messwertintegration geeignete Datenstruktur. Es akkumuliert die Messwertgrids unterschiedlicher Zeitpunkte.

– Das *Occupancy Grid* (OG) erlaubt eine für die Ergebnisrepräsentation geeignetere Griddarstellung. Es wird aus dem Hauptgrid gewonnen und stellt das Ergebnis dar, das die OG-Bibliothek an das externe Verwender-Programm zurückliefert.

Die zeitliche Akkumulation des Messwertgrids in das Hauptgrid erfolgt zum Zeitpunkt t nach dem Schema $HG_t := HG_{t-1} \oplus MG_t$ (vgl. Abbildung 4.2): Das bisherige Hauptgrid HG_{t-1} wird mit dem aktuellen Messwertgrid MG_t zum aktuellen Hauptgrid HG_t fusioniert. Dazu wird eine bestimmte Fusionsoperation $\oplus$ verwendet (z.B. binärer Bayes-Filter oder Dempsters Kombinationsregel). Die Daten des Messwertgrids werden anschließend nicht mehr benötigt und deshalb verworfen.

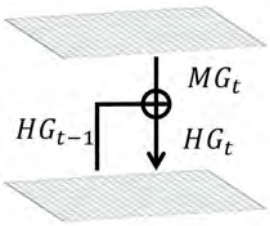

Abbildung 4.2: Integration des Messwertgrids in das Hauptgrid.

Das Hauptgrid HG_t besitzt im Allgemeinen noch nicht die für ein Occupancy Grid übliche Darstellung, sondern eine zur bibliotheksinternen Datenverwaltung geeignetere. Deshalb wird zur Ausgabe mit dem Occupancy Grid OG_t ein weiteres Grid verwendet, das aus dem Hauptgrid HG_t gewonnen wird und die übliche Ergebnisrepräsentation besitzt. Es stellt das Ergebnis der OG-Bibliothek dar, das das externe Verwender-Programm erhält.

Besagte Gewinnung des Occupancy Grids aus dem Hauptgrid ist aufgrund der in der Regel hohen Zellanzahl aufwendig. Deshalb wird das Occupancy Grid nur und erst erstellt, wenn es vom externen Programm über den OGBuilder angefordert wird.

Im Falle einer Anforderung überprüft der OGBuilder zuerst, ob bereits ein Occupancy Grid erzeugt wurde und dann gegebenenfalls, ob dieses noch aktuell ist. Falls bereits ein aktuelles Occupancy Grid vorliegt, wird dieses ohne weitere Bearbeitung sofort zurückgeliefert. Andernfalls wird versucht, es zu aktualisieren. Dies basiert auf der Überlegung, dass sich seit der letzten Anforderung eventuell nur ein kleiner Ausschnitt geändert hat und dessen Aktualisierung daher ausreichend ist.

Ist keine solche Aktualisierung möglich oder wurde noch gar kein Occupancy Grid erzeugt, wird aus dem Hauptgrid ein neues Occupancy Grid erstellt. Wie eine solche Aktualisierung konkret aussieht und unter welchen Rahmenbedingungen sie möglich ist, ist genauso von einer konkreten Implementierung abhängig wie die Art und Weise, auf die ein Occupancy Grid erzeugt wird, falls noch gar keines zur Aktualisierung vorliegt.

Abschließend ist auch noch ein genauerer Blick auf das Messwertgrid notwendig. Die Verwendung eines separaten Messwertgrids, das die gleiche Größe wie das Hauptgrid besitzt, wurde in [54] eingeführt für die Verwendung mit einem einzelnen Sensor. Die Idee wurde von vielen Arbeiten übernommen (z.B. [33][42][36][26]), da es in vielen Situationen eine deutlich einfachere und effizientere Kartierung erlaubt: Geschriebene Werte sind

im Messwertgrid erkennbar, auslesbar und veränderbar. Im Hauptgrid wird dagegen eine zeitliche Akkumulation durchgeführt, wodurch die Einzelwerte verloren gehen.

Das Messwertgrid, wie es bisher vorgestellt und in [54] eingeführt wurde, reicht im Rahmen der OG-Bibliothek nicht aus, da hier die Daten beliebiger, unterschiedlicher Sensoren gleichberechtigt eingebracht werden. *Gleichberechtigt* bedeutet dabei, dass keiner der Sensoren als Haupt- oder Nebensensor behandelt wird. Alle Sensoren werden als vollwertig und als zur Kartierung geeignet betrachtet[2]. *Gleichberechtigt* bedeutet jedoch ausdrücklich nicht, dass alle Sensoren als gleich zuverlässig betrachtet werden oder dass bei einem Sensor nicht auf die Kartierung verzichtet werden *kann*.

Wie bereits in Abschnitt 3.3.2 deutlich gemacht wurde, kommt es im Allgemeinen zu unerwünschten Ergebnissen, wenn die Daten unterschiedlicher Sensoren mit unterschiedlichen Messfrequenzen direkt in ein einzelnes Grid fusioniert werden. Um dies zu verhindern, besitzt das Messwertgrid mehrere Ebenen, wie in Abbildung 4.3 gezeigt.

Für jeden Mapper existiert eine eigene Ebene, in der dieser die Kartierung durchführt; diese stellt somit ein sensorlokales Messwertgrid dar. In diesen Ebenen werden getrennt die aktuellen Messungen der einzelnen Sensoren akkumuliert bis das Messwertgrid in das Hauptgrid integriert wird. Erst im Rahmen dieser Integration werden die einzelnen Ebenen zu einem gemeinsamen Messwertgrid kombiniert, das heißt, die Daten der unterschiedlichen Sensoren fusioniert. Da dies im Allgemeinen erst geschieht, wenn von jedem Sensor wenigstens ein Messwert kartiert wurde[3], kommt es dadurch zu keinen unerwünschten Effekten, obwohl die Sensordaten in ihrer jeweiligen Frequenz (in das sensorlokale Messwertgrid) eingetragen werden.

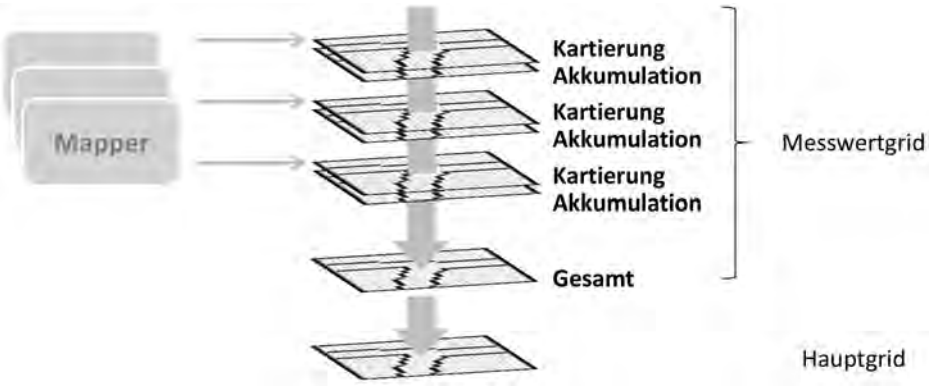

Abbildung 4.3: Architektur des Messwertgrids (gezeigt für drei Sensoren).

Die beschriebene Architektur ist allerdings noch nicht ausreichend, denn bis zur Integration des Messwertgrids in das Hauptgrid ist, wie eben erwähnt, eine sensorlokale Akkumulation notwendig; Kartierung und Akkumulation sollen aber getrennt stattfinden – dies ist schließlich der Hauptgrund für die Verwendung eines Messwertgrids. Die Ebene eines jeden Mappers wird deshalb wiederum unterteilt in zwei Ebenen, wie es ebenfalls in Abbildung 4.3 zu sehen ist. Die obere Kartierungsebene des Mappers ist diejenige, in der

[2]Dies steht in Kontrast zu einigen anderen Arbeiten; in [7][53] beispielsweise werden Radarsensoren nur zur Geschwindigkeitsbestimmung eingesetzt, deren Daten jedoch nicht kartiert.

[3]Dieses Verhalten sollte das im Regelfall erwünschte sein; es lässt sich aber durch Veränderung der entsprechenden Funktion im OGBuilder leicht auch an andere Bedürfnisse anpassen.

er die Messwerte eines Zeitschritts kartiert und noch verändern kann; die untere Akkumulationsebene ist diejenige, in der die zeitliche, sensorlokale Akkumulation stattfindet, bis das Messwertgrid in das Hauptgrid integriert wird, das heißt, bis von jedem Sensor wenigstens ein Messwert kartiert wurde. Ist dieser Fall eingetreten, werden die sensorspezifischen Messwertgrids zunächst zu einem Gesamt-Messwertgrid kombiniert, das alle aktuellen Messungen repräsentiert. Letzteres wird in das Hauptgrid integriert.

Das in der OG-Bibliothek verwendete Messwertgrid – mit der eben vorgestellten Ebenen-Architektur zur Unterstützung mehrerer, verschiedenartiger Sensoren und der später vorgestellten Unterscheidung zwischen logischer und realer Größe (vgl. Abschnitt 4.2.5.1) – kann als Weiterentwicklung und Erweiterung des ursprünglichen Ansatzes von [54] verstanden werden.

4.2.2.2 Koordinatensysteme & Posen

Zur Angabe der Lage von Objekten im Grid oder zur Kartierung der Sensormesswerte an der richtigen Stelle, ist die Verwendung von Koordinatensystemen notwendig. In unserem Kontext müssen drei Koordinatensysteme unterschieden werden [17]:

Sensorkoordinatensystem Das lokale Koordinatensystem eines einzelnen Sensors, in dem dieser seine Messdaten ermittelt. Unterschiedliche Sensoren besitzen im Allgemeinen auch unterschiedliche Sensorkoordinatensysteme.

Fahrzeugkoordinatensystem Ein ausgezeichnetes, fahrzeugfestes Koordinatensystem; es ist bezüglich eines bestimmten Referenzpunktes am Fahrzeug definiert und bewegt sich mit diesem mit.

Weltkoordinatensystem Ein ausgezeichnetes, erdfestes Koordinatensystem; es ist bezüglich eines bestimmten Referenzpunktes auf der Erde definiert.

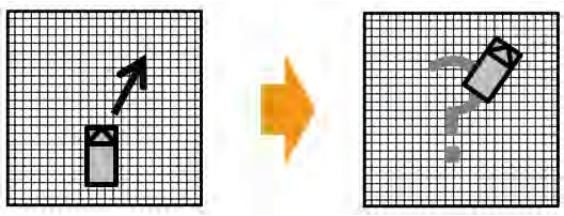

Abbildung 4.4: Grid und Fahrzeugbewegung.

Betrachtet man ein Grid und ein Fahrzeug, das sich initial irgendwo in diesem Grid befindet, so muss das Verhältnis zwischen Fahrzeug und Grid aktualisiert werden, wenn sich das Fahrzeug bewegt (vgl. Abbildung 4.4). Es gibt prinzipiell zwei Möglichkeiten für die Beziehung zwischen dem Koordinatensystem des Grids und dem des sich bewegenden Fahrzeugs [17]:

fahrzeugfeste Griddarstellung Das Grid bewegt sich mit dem Fahrzeug mit. Das führt dazu, dass die statischen Zellinhalte des Grids bei jeder Fahrzeugbewegung angepasst werden müssen, da diese nicht statisch gegenüber dem Fahrzeugkoordinatensystem, sondern gegenüber dem erdfesten Weltkoordinatensystem sind. Dies resultiert in einem sehr hohen Rechenaufwand, der sich bei einer Kurvenfahrt des Fahrzeugs durch

die erforderliche Drehung des Grids weiter erhöht. Insbesondere kommt es durch die Drehung auch zu Diskretisierungsfehlern [54] (siehe roter Bereich in Abbildung 4.5).

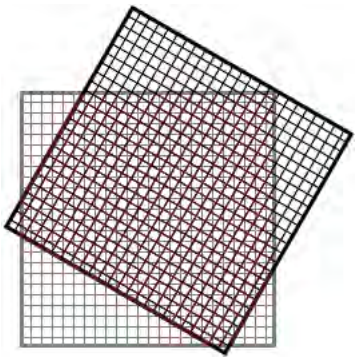

Abbildung 4.5: Fahrzeugfeste Griddarstellung.

erdfeste Griddarstellung Das Grid besitzt eine feste Pose im Weltkoordinatensystem und nur das Fahrzeug verändert durch seine Bewegung seine Pose im Weltkoordinatensystem und damit relativ zum Grid. Das Fahrzeug muss auf dem Grid folglich nur transliert und rotiert werden. Allerdings kommt es nun zu der Situation, dass das Fahrzeug durch seine Fahrt das Grid verlässt. Da das Grid eine feste Größe besitzt, müssen deshalb neue Zellen hinzugefügt und alte entfernt werden (grün bzw. rot in Abbildung 4.6). Anders als in der fahrzeugfesten Griddarstellung müssen die statischen Zellinhalte des Grids jedoch nicht verschoben werden.

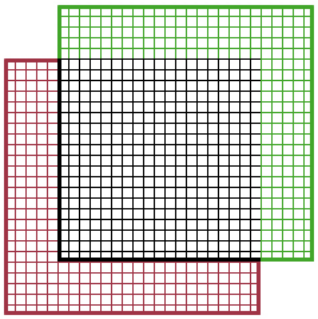

Abbildung 4.6: Erdfeste Griddarstellung.

Alle Grids der OG-Bibliothek verwenden einheitlich eine erdfeste Darstellung, da diese offensichtlich besser geeignet ist. Es kommt das zweidimensionale *Gridkoordinatensystem*[4]

[4]Trotz der vorhandenen Ähnlichkeit wird bewusst nicht von einem *Welt-*, sondern von einem *Gridkoordinatensystem* gesprochen. Zum Einen, weil das Gridkoordinatensystem aufgrund der Projektion in die Bodenebene auf den zweidimensionalen Fall beschränkt ist. Zum Anderen, weil kein absoluter Referenzpunkt auf der Erde definiert wird, wie ihn ein Weltkoordinatensystem jedoch benötigt. Dies liegt daran, dass absolute Positionen in vielen Anwendungsfällen nicht verfügbar sind – zum Beispiel, wenn wie üblich Odometriedaten zur Positionsbestimmung verwendet werden, ohne dass der Startpunkt absolut gegeben ist. Eine Verallgemeinerung auf absolute Positionen ist im Bedarfsfall (z.B. bei Positionsbestimmung mithilfe eines GPS-Empfängers) jedoch naheliegend und leicht möglich.

zum Einsatz. Die Ebene des Gridkoordinatensystems wird – wie alle Grids – in der Erdoberfläche liegend angenommen und an beiden Achsen entspricht der Maßstab der Größe einer einzelnen Zelle. Der Ursprung und die Richtung der positiven X-Achse des Gridkoordinatensystems entsprechen der initialen Position bzw. Orientierung des Fahrzeugs. Das Fahrzeug befindet sich damit im Gridkoordinatensystem initial an der Position (0,0) und besitzt eine Orientierung von 0 Grad.

Im Rahmen der Kartierung wird die Welt in die Ebene des Gridkoordinatensystems projiziert, das heißt, alle Objekte befinden sich in der Bodenebene, wo sich auch die Grids befinden. Die entsprechende Projektion der Sensormesswerte aus dem jeweiligen, im Allgemeinen dreidimensionalen Sensorkoordinatensystem in das zweidimensionale Gridkoordinatensystem ist Aufgabe der Sensormapper (vgl. Abschnitt 4.2.4.2).

Die Datei PositionOrientationBasics.cpp und der zugehörige, gleichnamige Header bieten die grundlegenden Strukturen und Operationen zur kontinuierlichen und diskreten Modellierung der Pose eines Objekts im Gridkoordinatensystem. Positionen und Posen werden innerhalb der OG-Bibliothek immer mit Hilfe der in dieser Datei befindlichen Strukturen angegeben, die nachfolgend beschrieben werden.

Die *Pose* eines Objekts (Struktur Pose) setzt sich zusammen aus

- seiner *kontinuierlichen Position* $(x,y) \in \mathbb{R} \times \mathbb{R}$ in der X-Y-Ebene des Gridkoordinatensystems (Struktur Position[5]) und
- seiner *Orientierung* $\theta \in [0,360[$ in der X-Y-Ebene des Gridkoordinatensystems (also dem so genannten *Gierwinkel*), gemessen in Grad.

Die kontinuierliche Position eines Objekts setzt sich wiederum zusammen aus (siehe auch Abbildung 4.7)

- einem diskreten Anteil, $(i,j) \in \mathbb{Z} \times \mathbb{Z}$, der die Position der linken unteren Ecke der Zelle im Gridkoordinatensystem angibt und als *diskrete Position* oder *Zell-Position* bezeichnet wird (Struktur CellPosition), und
- einem Nachkomma-Anteil $(k,l) \in [0,1[\times[0,1[$, der der Position innerhalb der Zelle entspricht und als *In-Zell-Position* bezeichnet wird (Struktur InCellPosition).

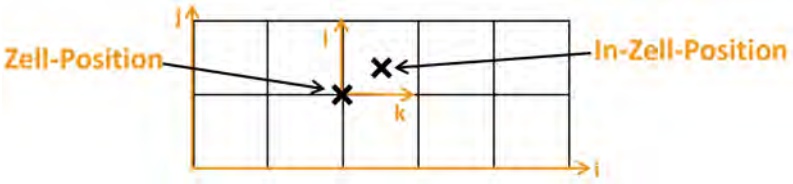

Abbildung 4.7: Position bestehend aus Zell- und In-Zell-Position.

Die kontinuierliche Position (x,y) wird nicht gespeichert, da sie, wenn sie benötigt wird, leicht aus der Zell-Position (i,j) und der In-Zell-Position (k,l) durch koordinatenweise Addition berechnet werden kann: $(x,y) := (i,j)+(k,l) = (i+k,j+l)$. Es stehen etliche Konstruktoren und Operatoren bereit, die eine komfortable Arbeit mit den genannten Strukturen ermöglichen und dabei, wo sinnvoll, auch den Wechsel zwischen den Darstellungen erlauben.

[5]In der Implementierung wird x als ik und y als jl bezeichnet.

Die zweiteilige Modellierung der kontinuierlichen Position bietet zwei entscheidende Vorteile. Erstens ermöglicht dies eine leichtere Adressierung der Zellen, da die sonst aufgrund der diskreten Gridstruktur bei jedem Zugriff notwendige Rundungsoperation vermieden wird. Stattdessen erfolgt diese Berechnung einmalig und ist alsdann nicht mehr notwendig. Zweitens können die Berechnungen auf zwei unterschiedlichen Genauigkeitsebenen durchgeführt werden – je nachdem, welche Genauigkeit für die zu erledigende Aufgabe vorteilhaft ist. Wird eine genaue Position benötigt, kann die Struktur POSITION verwendet werden; sollen dagegen beispielsweise nur Zellen adressiert werden, kann mittels der Struktur CELLPOSITION direkt der diskreten Gridstruktur entsprechend gerechnet werden.

4.2.2.3 Ablauf

Die Klasse OGBUILDER ist die Schnittstelle der OG-Bibliothek zu dem externen Programm, das die OG-Bibliothek zur Erstellung eines Occupancy Grids verwendet. Dieses Programm bindet dazu eine Instanz der Klasse OGBUILDER ein, über welche anschließend jegliche Interaktion mit der OG-Bibliothek erfolgt.

Bei der Instanziierung wird dem OGBUILDER die Größe des zu erstellenden, quadratischen[6] Occupancy Grids (Länge einer Seite in Meter) und die zu verwendende Auflösung mitgeteilt; die Auflösung beschreibt die Größe einer einzelnen, quadratischen Zelle (Länge einer Seite in Zentimeter) und bestimmt damit den Detailgrad des Ergebnisses. Außerdem werden dem OGBUILDER Konfigurationsdaten für das Fahrzeug und dessen Umfeldsensoren übergeben. Basierend auf den beiden Gridparametern und den Konfigurationsdaten bereitet der OGBUILDER die Erstellung eines Occupancy Grids vor:

- Die benötigten Grids werden mit den spezifizierten Parametern generiert und initialisiert.
- Für jeden Sensor wird ein zugehöriger *Sensormapper* erzeugt, der für die Kartierung dessen Daten zuständig ist.
- Es wird ein *Fahrzeugmapper* instanziiert, der für die Eigenzustandsschätzung und optional die Kartierung der Fläche des eigenen Fahrzeugs zuständig ist.

Anschließend ist der OGBUILDER bereit und wartet auf den Erhalt von Sensormesswerten, aus denen ein Occupancy Grid generiert werden soll. Er steuert für jeden Zeitschritt – das heißt für jedes Eintreffen neuer Daten – den Ablauf innerhalb der OG-Bibliothek. Dieser ist in Abbildung 4.8 skizziert und wird im Folgenden genauer beschrieben.

Zu Beginn eines Zeitschritts erhält der OGBUILDER vom externen Programm gleichzeitig[7] die aktuellen Eigenzustandsdaten und die zu diesem Zeitpunkt verfügbaren Umfelddaten. Die Eigenzustandsdaten sind dabei immer vorhanden (Typedef VEHICLERAWDATA). Zudem können Daten der Umfeldsensoren vorliegen, die dann in der Struktur SENSORDATA zusammengefasst sind, um eine leichtere Handhabung zu ermöglichen. Es können die Daten aller Umfeldsensoren vorliegen, es kann aber auch sein, dass nur einige von ihnen Messwerte liefern – oder keiner.

Die Eigenzustandsdaten reicht der OGBUILDER an den Fahrzeugmapper (Interface IVEHICLEMAPPER) weiter, welcher damit die Schätzung des Eigenzustands aktualisiert.

[6] Die quadratische Größe ist aufgrund der Fahrzeugbewegung intern notwendig (vgl. Abschnitt 4.2.5).
[7] Diese Synchronisierung ist Aufgabe des externen Programms (vgl. Abschnitt 4.2.1.2).

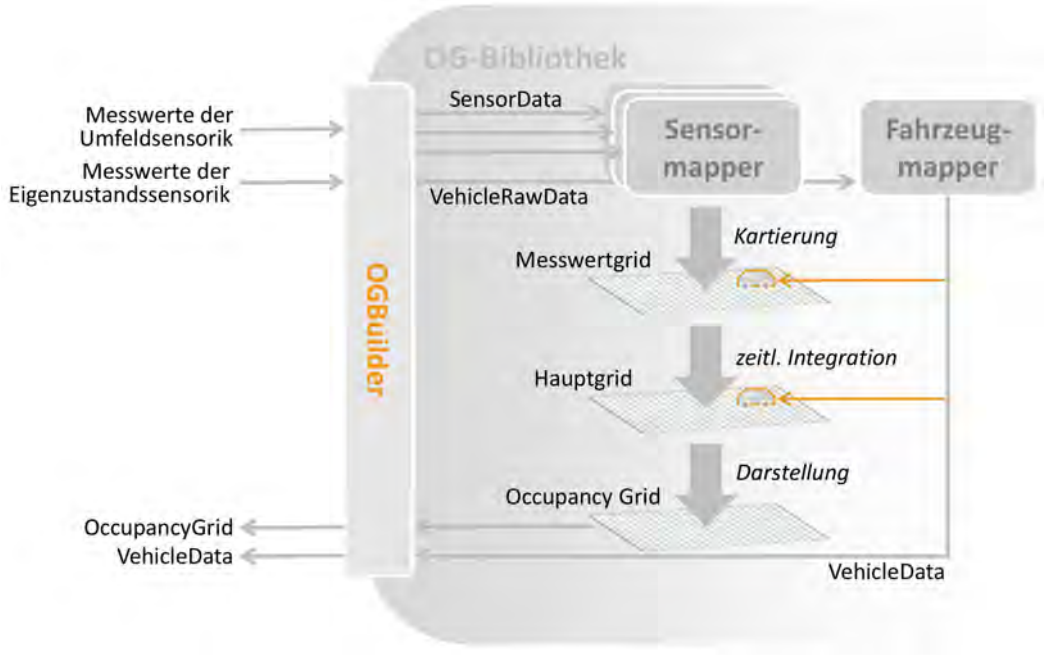

Abbildung 4.8: Schematischer Ablauf innerhalb der OG-Bibliothek.

Im Allgemeinen beinhaltet der Eigenzustand die Fahrzeugpose im Grid und die Fahrzeuggeschwindigkeit (Struktur VehicleData).

Falls sich die aktuelle Eigenposition gegenüber der letzten bekannten nicht verändert hat, werden die Messdaten verworfen[8]. Falls sie sich verändert hat, verändern Messwert- und Hauptgrid (Interface IMeasurementGrid bzw. IMainGrid) jeweils den aktuell von ihnen repräsentierten Bereich entsprechend der neuen Eigenposition.

Sind im aktuellen Zeitschritt neue Umfelddaten vorhanden, werden diese anschließend vom OGBuilder der Reihe nach an den jeweils dafür zuständigen Sensormapper (Interface ISensorMapper) weitergegeben, der diese in der ihm zugeteilten Ebene in das Messwertgrid einträgt. Falls in dieser Ebene bereits Daten vorangehender Zeitschritte vorhanden sind, erfolgt dort eine sensorlokale, temporäre Akkumulation.

Wurde bereits von jedem Umfeldsensor mindestens ein Messwert kartiert, werden die akkumulierten Ebenen zu einem einzigen Gesamt-Messwertgrid zusammengefasst und dieses in das Hauptgrid integriert (Interface IGridFusioner); danach werden alle Daten des Messwertgrids verworfen, da sie nun nicht mehr benötigt werden.

Anschließend ist der aktuelle Zeitschritt beendet und der nächste beginnt, sobald der OGBuilder wieder Daten erhält.

Betrachtet man das Hauptgrid und den geschilderten Ablauf über mehrere Zeitschritte, so wird darin – durch die Eigenbewegung des Fahrzeugs und die Messungen zu mehreren Zeitpunkten – eine Karte aufgebaut, die allerdings nicht die für Occupancy Grids übliche Darstellung besitzt. Da aber das externe Programm, das die OG-Bibliothek nutzt, an einem Occupancy Grid (Interface IOccupancyGrid) interessiert ist, wird die Karte des Hauptgrids in die entsprechende Darstellung überführt. Dieser relativ aufwendige Vorgang erfolgt

[8]In Rahmen des Occupancy-Grid-Mappings wird die Annahme getroffen, dass die Messungen aus voneinander unabhängigen Experimenten stammen. Wurden die Messungen jedoch vom gleichen Ort aus durchgeführt, ist diese Annahme nicht mehr haltbar [29][39].

aus Performanzgründen nur und erst, wenn das externe Programm das Occupancy Grid über den OGBuilder explizit anfordert. Ferner kann es auch die aktuelle Schätzung des Eigenzustands abrufen, die dann in der Struktur VehicleData zurückgegeben wird.

4.2.3 Die Zellen

In diesem Abschnitt wird der grundlegende Bestandteil eines Grids vorgestellt: die miteinander verbundenen Zellen. Die Zellen des Hauptgrids implementieren das Interface IMainCell, die Zellen des Messwertgrids das Interface IMeasurementCell. Konzept und Aufbau beider Zellarten sind dennoch gleich, weshalb sie in diesem Abschnitt gemeinsam behandelt werden. Die Zellen des Occupancy Grids unterscheiden sich deutlich, sie werden deshalb später gesondert behandelt (vgl. Abschnitt 4.2.5.3).

Zum besseren Verständnis der nachfolgenden Ausführungen wird ergänzend dazu das Studium der genannten Interfaces und ihrer Implementierungen empfohlen (siehe DVD im Anhang).

4.2.3.1 Aufbau & Dynamikerkennung

Betrachten wir zunächst die in Abbildung 4.9 gezeigte Vereinfachung des Zellaufbaus.

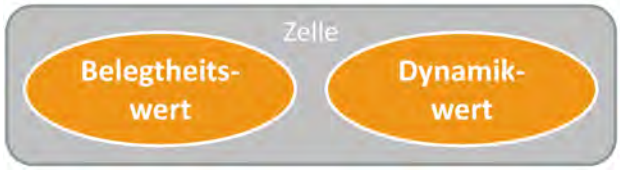

Abbildung 4.9: Vereinfachter Zellaufbau bestehend aus Belegtheit und Dynamik.

Jede Zelle enthält, wie für ein Occupancy Grid üblich, einen *Belegtheitswert* – die Evidenz dafür, dass der von der Zelle repräsentierte Bereich durch ein statisches Objekt belegt ist. Der Typ dieser Evidenz kann frei gewählt werden. Beispielsweise können klassische Wahrscheinlichkeiten oder Dempster-Shafer-Massen verwendet werden, wodurch das klassische bzw. evidenztheoretische Occupancy Grid Mapping leicht realisiert werden kann.

Zusätzlich enthält jede Zelle einen zweiten Wert, den so genannten *Dynamikwert*, der eine Evidenz oder Information irgendeiner Art (Wahrscheinlichkeit, Geschwindigkeit, Belegtheitsverhältnis, ...) für Dynamik im von dieser Zelle repräsentierten Bereich speichert. Die Art der Evidenz kann ebenfalls frei – auch unabhängig von der für den Belegtheitswert getroffenen Wahl – gewählt und damit den Bedürfnissen der eigenen Anwendung angepasst werden.

Die Idee für die zellweise Speicherung einer zusätzlichen Dynamikinformation geht zurück auf [52][4]. In [52] wird ein zusätzliches Dynamikgrid eingeführt, in dem jede Zelle die dortige Anzahl bisheriger Detektionen dynamischer Objekte speichert. [4] speichert in jeder Zelle eine binäre Dynamikklassifikation für diesen Bereich (*statisch*/*dynamisch*). Der Ansatz vorliegender Arbeit kann als Kombination und Erweiterung der beiden Ansätze aufgefasst werden. Es können dabei beliebige Arten von Informationen gespeichert werden. Dass die Wahl dazu auf die Erweiterung der Zellen fiel, anstatt ein separates Dynamikgrid zu verwenden, liegt an der höheren Effizienz dieser Möglichkeit: Es ist die Verwaltung eines

anstatt von zwei Grids ausreichend, wodurch insbesondere der Aufwand für die Navigation reduziert wird.

Jede Zelle besitzt also einen zweiteiligen Aufbau aus Belegtheit und Dynamik. Belegtheits- und Dynamikwert werden dauerhaft voneinander unabhängig verwaltet. Insbesondere werden beide auch separat kartiert und zeitlich integriert – so als *wären* sie jeweils in einem eigenen Messwert- und Hauptgrid.

Um diesen für eine einzelne Zelle gewählten Aufbau nachvollziehen zu können, muss etwas ausgeholt werden und insbesondere das automobile Umfeld einbezogen werden. Typischerweise wird in jeder Zelle eines Occupancy Grids die Wahrscheinlichkeit dafür gespeichert, dass der von ihr repräsentierte Bereich durch ein statisches Objekt belegt ist – dynamische Objekte dürfen nicht kartiert werden, da es sonst zu Inkonsistenzen kommt (vgl. Abschnitt 3.3.1). In der automobilen Realität ist die Beschränkung auf statische Objekte aber schwierig, da eine zuverlässige Unterscheidung zwischen dynamischen und statischen Objekten nicht immer leicht ist.

In Abschnitt 3.3.1 wurden etliche unterschiedliche Ansätze zur Dynamikerkennung und -filterung vorgestellt. Welche dieser Methoden zur Dynamikerkennung geeignet ist, hängt stark von der jeweiligen Situation ab.

Die OG-Bibliothek ermöglicht mit obigem Zellaufbau deshalb die Wahl der im Kontext eines konkreten Anwendungsfalls geeigneten Vorgehensweise. Dabei können auch unterschiedliche Verfahren für unterschiedliche Sensoren eingesetzt werden. Realisiert wird dies, indem die Dynamikerkennung prinzipiell für jeden Sensor unabhängig und auf diesen zugeschnitten lokal im zugehörigen Sensormapper erfolgt.

Die Sensormapper führen auf Basis der Messungen des ihnen zugeordneten Sensors lokal eine Dynamikerkennung aus. Die dabei ermittelte Evidenz für Dynamik tragen sie in die ihnen zugeordnete Kartierungsebene des virtuellen Dynamikgrids ein, zellweise als Dynamikwerte. Anschließend werden die Ergebnisse der sensorlokalen Dynamikerkennungen über eine Aktualisierungsgleichung (z.B. Dempsters Kombinationsregel) kombiniert; auch diese kann unabhängig von der Aktualisierungsgleichung für Belegtheit gewählt werden. Der Gesamt-Dynamikwert einer Zelle wird schließlich zur Dynamikfilterung verwendet; beispielsweise, indem Zellen mit hohen Dynamikwerten im Occupancy Grid als *frei* dargestellt werden, da sich in diesen Zellen mutmaßlich dynamische Objekte befinden und diese nicht kartiert werden sollen. Details zu derartigen Darstellungsmöglichkeiten folgen später in Abschnitt 4.2.5.3.

Abbildung 4.10: Realer Zellaufbau mit Containern und Konfliktwerten.

Abbildung 4.10 zeigt, dass der tatsächliche Aufbau einer Zelle komplexer ist, als er bisher beschrieben wurde. Dies ist darauf zurückzuführen, dass die OG-Bibliothek zusätzlich zur eben beschriebenen, sensorspezifischen Dynamikerkennung eine konfliktbasierte Dynamikerkennung unterstützt. Die Idee dazu haben wir in Abschnitt 3.3.1 bereits kennen gelernt:

Dynamische Objekte erzeugen aufgrund ihrer Bewegung bei der Kombination von bisherigem und neuem Zellwert Widersprüche, die man elegant mit einem Konfliktmaß quantifizieren und zur Dynamikerkennung nutzen kann. Diese Möglichkeit nutzt die OG-Bibliothek folgendermaßen.

Wie üblich wird eine Aktualisierungsgleichung verwendet, um zellweise den alten mit dem neuen Belegtheitswert zu kombinieren. Der aufgetretene Widerspruch wird dabei mit einem frei wählbaren Konfliktmaß quantifiziert und als so genannter *Konfliktwert* zusätzlich in der Zelle gespeichert. Später wird der Konfliktwert im Rahmen der Dynamikerkennung genutzt, um den Dynamikwert der Zelle entsprechend zu beeinflussen: Je höher der Konfliktwert einer Zelle ist, desto höher ist die Wahrscheinlichkeit für ein dynamisches Objekt an dieser Stelle.

Ferner wird auch ein zum Dynamikwert gehöriger Konfliktwert gespeichert; die Vorgehensweise ist analog. Dadurch können auch Widersprüche gemessen und berücksichtigt werden, die sich bei der Kombination der sensorlokalen Dynamikerkennungsergebnisse oder bei der zeitlichen Integration des Dynamikwerts ergeben.

Die eben vorgestellte konfliktbasierte Dynamikerkennung eignet sich insbesondere bei Verwendung evidenztheoretischer Occupancy Grids, da ein Konfliktmaß integraler Bestandteil der Dempster-Shafer-Theorie ist und dadurch leicht und effizient verwendet werden kann. Beispielsweise können Dempster-Shafer-Massen als Evidenzen und Dempsters Kombinationsregel als Aktualisierungsgleichung verwendet werden; dann kann die Konfliktmasse k berechnet werden, die dann direkt oder in Form der *Con*-Metrik als Konfliktwert verwendet werden kann (vgl. Abschnitt 3.2.2).

Die OG-Bibliothek trifft jedoch keine Annahmen über die verwendete Theorie und damit die Art der Evidenzen und der Fusionsoperation. Um in jedem Fall die Verwendung eines Konfliktmaßes zu unterstützen, werden Belegtheits- und Dynamikwert jeweils in einen so genannten *Container* gepackt (siehe Abbildung 4.10): Dieser erweitert den Belegtheits- bzw. Dynamikwert, der allgemein dann als *Basiswert* bezeichnet wird, um einen Konfliktwert. Der Container implementiert das Interface ICellDataFusionContainer, wobei der Typ des gewünschten Basiswerts als Template-Parameter übergeben wird. Durch die Verwendung von Containern können auch primitive Datentypen ohne Einschränkung als Basiswert verwenden werden, da sie dadurch entsprechend ergänzt werden.

Die Wahl von Belegtheits- und Dynamikcontainer ist voneinander unabhängig möglich, zudem kann eine unterschiedliche Wahl für Hauptgrid- und Messwertgridzellen getroffen werden. Welche Container die Zelle verwendet, wird bei der Implementierung ihres Interfaces IMainCell bzw. IMeasurementCell über dessen Template-Argumente festgelegt (vgl. Abschnitt 4.2.5.1); über den gewählten Container wird implizit auch eine Wahl für die von der Zelle genutzten Evidenzen getroffen. Mit der Klasse CellDataFusionContainer ist eine fertige Implementierung vorhanden, die eine beliebige Art von Evidenz als Typ des Basiswerts in Form eines Template-Arguments entgegennimmt.

Der eben vorgestellte Zellaufbau unterstützt die Dynamikerkennung und -filterung in erheblichem Maße und bietet eine Reihe von Vorteilen.

Ein offensichtlicher Vorteil dieses Ansatzes ist seine Anpassbarkeit an das jeweilige Anwendungsszenario. Zur Dynamikerkennung ist der Einsatz und die Kombination von denjenigen sensorspezifischen Methoden möglich, die sich aufgrund des oder der eingesetzten Sensoren anbieten (z.B. Nutzung von Geschwindigkeitsmessungen). Durch die konfliktbasierte Dynamikerkennung können dynamische Bereiche auch ohne Geschwindigkeitsmessun-

gen erkannt werden; sollte die Dempster-Shafer-Theorie zum Einsatz kommen, können auf diese Weise ihre diesbezüglichen Vorteile voll ausgenutzt werden. Zudem ermöglichen die Dynamikwerte den Einsatz beliebig komplexer Algorithmen zur Dynamikfilterung.

Von großer Bedeutung in künftigen Umfeldmodellen ist die Verwendung eines hybriden Ansatzes (vgl. Abschnitte 2.3.2 und 3.3.3). In unserem Fall heißt das, das Occupancy Grid soll mit der Objektliste eines objektbasierten Ansatzes kombiniert werden. Diese Kombination ist durch den Dynamikwert leicht möglich: An den Stellen, an denen die verfolgten Objekte vom objektbasierten Ansatz gerade vermutet werden, wird im Occupancy Grid einfach ein höherer Dynamikwert gesetzt[9]. Dies hat zur Folge, dass diese Zellen im Occupancy Grid aufgrund ihres hohen Dynamikwerts als *frei* dargestellt werden.

Ein weiterer Vorteil ist die verzögerte Dynamikklassifizierung oder deren nachträgliche Korrektur, die im klassischen Ansatz nicht gegeben ist. Ist initial die Unsicherheit noch zu groß für eine zuverlässige Entscheidung, ob es sich um ein dynamisches Objekt handelt, können zunächst entsprechende Evidenzen im Dynamikwert gesammelt werden und die zugehörige Zelle erst dann als *frei* kartiert werden, sobald die gesammelte Evidenz mit ausreichender Sicherheit für ein dynamisches Objekt spricht. Besonders interessant ist der gegenteilige Fall: Stellt sich heraus, dass ein Objekt, das ursprünglich für ein dynamisches Objekt gehalten und deshalb nicht kartiert wurde, doch ein statisches Objekt ist, so kann die Darstellung ab diesem Zeitpunkt korrigiert werden (durch eine Kartierung der vom Objekt belegten Zellen als *belegt*), da die Belegtheitsevidenzen unabhängig von der Annahme eines dynamischen Objekts weiterhin gesammelt wurden – anders als im üblichen Ansatz.

Ebenfalls gewinnbringend kann je nach Anwendungsfall die Visualisierung von Dynamik sein, die auf Basis des Dynamikwerts leicht möglich ist. Darauf wird später in Abschnitt 4.2.5.3 genauer eingegangen.

Der Dynamikwert stellt zudem eine einfache und elegante Möglichkeit dar, die Freiraummodellierung (vgl. Abschnitt 3.1.2) korrekt zu realisieren. Hier ist die Betrachtung von statischen als auch dynamischen Objekten notwendig: Obwohl die dynamischen Objekte nicht kartiert werden sollen, verschatten sie dennoch den Bereich hinter ihnen – so dass an ihrer Stelle und im Bereich davor Freiraum kartiert werden muss, im Bereich dahinter Unwissenheit. Die Zellen dynamischer Objekte sind also zu behandeln als wären sie *belegt*, obwohl sie als *frei* kartiert und deshalb von Freiraum nicht unterscheidbar sind. Hier bietet der Dynamikwert eine einfache Möglichkeit zur Lösung dieser Problematik: Die Freiraummodellierung nimmt alle Zellen als *belegt* an, die einen hohen Belegtheits- und/oder Dynamikwert besitzen.

4.2.3.2 Verknüpfung & Navigation

Im vorherigen Abschnitt wurde der Aufbau einer einzelnen Zelle beschrieben, der für die Zellen des Messwertgrids und des Hauptgrids gleich ist. Beide Grids besitzen typischerweise eine große Anzahl an Zellen, die sie zu einer Gitterstruktur verknüpfen. Wie diese Verknüpfung aussieht, ist abhängig von der jeweils verwendeten Zell-Implementierung.

Mit den Klassen MeasurementCell und MainCell stehen für beide Grids fertige Zell-Implementierungen bereit, die die entsprechenden Interfaces implementieren (Interface IMeasurementCell bzw. IMainCell) und von der Klasse FourNeighboursCell

[9] Man kann den objektbasierten Ansatz hierzu als virtuellen Sensor betrachten und über den zugehörigen Sensormapper einen hohen Dynamikwert fusionieren.

erben. Die Grids verknüpfen diese Zellen zu einer Gitterstruktur, indem für jede Zelle der obere, untere, linke und rechte Zellnachbar gespeichert wird. Der Vorteil dieser Modellierung als Vierernachbarschaft ist, dass diese einen schnellen Zugriff auf benachbarte Zellen und dadurch eine effiziente Iteration über das Grid ermöglicht. Die zugrundeliegende Klasse FourNeighboursCell implementiert das Interface IFourNeighboursCell und kann auch für eigene Zell-Implementierungen als Grundlage verwendet werden.

Das Messwertgrid besitzt mehrere Ebenen, wie einleitend in Abschnitt 4.2.2.1 dargestellt. Zu beachten ist in diesem Zusammenhang, dass diese Ebenen-Struktur auf Zellebene realisiert wird – das heißt, das Messwertgrid verknüpft Zellen, die jeweils mehrere Ebenen besitzen. Durch diese Modellierung ist eine einzelne paarweise Verknüpfung zwischen den Zellen ausreichend, was einen entscheidenden Vorteil bei der Navigation bringt. Andernfalls wäre statt einer Zelle nämlich eine Zelle je Ebene notwendig – die Anzahl der Zellen würde also linear mit der Zahl der Ebenen steigen, wodurch ebenfalls die Anzahl an Verknüpfungen zunähme. Da dadurch bei der Navigation zu einer Nachbarzelle den Verknüpfungen auf allen Ebenen gefolgt werden müsste, anstatt nur einer, stiege der Navigationsaufwand entsprechend.

Möchte man auf eine einzelne Haupt- oder Messwertzelle zugreifen, ist eine Navigation über die Gridstruktur zu der jeweiligen Zelle notwendig. Dazu werden die Möglichkeiten genutzt, die das Grid-Interface IMainGrid bzw. IMeasurementGrid bereitstellt: mittels unterschiedlicher Iteratoren, unter Verwendung des Besucher-Musters oder mit einem speziellen Getter. Nicht erwünscht – und deshalb auch nicht möglich – ist, dass der Benutzer direkt von einer Zelle zu einer benachbarten Zelle navigiert, indem er die Verknüpfung zwischen ihnen ausnutzt; auf diese Weise kann nämlich nicht sichergestellt werden, dass der Zugriff auf eine gültige Position im Grid erfolgt[10].

Die erste Methode zur Navigation über das Haupt- oder Messwertgrid sind verschiedene *Iteratoren*. Sie dienen der Iteration über das Grid in unterschiedlichem Kontext und mit unterschiedlichem Ziel:

- Der *Blockiterator* dient dem Durchlaufen eines Gridblocks (am Grid ausgerichtetes Viereck) zwischen einem Start- und einem diagonal gegenüberliegenden End-Eckpunkt.
- Der *sequentielle Iterator* ermöglicht die freie Navigation im Grid nach oben, unten, links und rechts ausgehend von einem Startpunkt.
- Der *geometrische Iterator* steht nur im Messwertgrid zur Verfügung, wo er zum Durchlaufen einer Linie oder eines Parallelogramms beliebiger Ausrichtung verwendet werden kann.
- Zusätzlich existiert von allen drei Iteratoren eine *const*-Version. Diese erlaubt keine Veränderung der Zellen über die iteriert wird, ist aber ansonsten analog definiert.

Die Iteratoren besitzen keine eigenen Interfaces, da diese stattdessen in die Grid-Interfaces IMainGrid bzw. IMeasurementGrid integriert sind, worüber man auch eine Instanz der Iteratoren erhält. Die OG-Bibliothek stellt mit den Klassen BaseBlockIterator, BaseSequentialIterator und BaseGeometricIterator fertige Iterator-Implementierungen

[10] Wir werden bei der Vorstellung von Haupt- und Messwertgrid in den Abschnitten 4.2.5.2 und 4.2.5.1 noch sehen, dass nicht jede vorhandene benachbarte Zelle eine gültige Position im Grid besitzt.

bereit. Diese befinden sich – anders als die Interfaces – aus mehreren Gründen in separaten Klassen:

- Beide Grids und auch eigene Grid-Implementierungen können die gleichen Iterator-Implementierungen verwenden[11].
- Die normale Version und die *const*-Version eines Iterators können die gleiche Iterator-Implementierung verwenden.
- Die vorhandenen Grid-Implementierungen können auch mit anderen Iteratoren verwendet werden.

Diese Punkte werden durch die Verwendung von Templates realisiert. Jeder Iterator nimmt als Template-Argument den Typ des Grids und den Typ der Zelle entgegen; übergibt man den Zelltyp mit dem vorgestellten Schlüsselwort *const*, so erhält man die entsprechende *const*-Version. Auch jedem Grid-Interface werden die zu verwendenden Iterator-Implementierungen als Template-Parameter übergeben[12].

Eine weitere Möglichkeit zur Navigation ist die Verwendung eines *geometrischen Besuchers* – einer Kombination des geometrischen Iterators (siehe oben) mit dem so genannten *Besucher-Muster* (engl. visitor pattern). Der geometrische Besucher durchläuft alle Zellen entlang einer Linie oder entlang eines (beliebig gedrehten) Rechtecks, wobei auf jeder „besuchten" Zelle eine Funktion ausgeführt wird – bis alle Zellen durchlaufen wurden oder eine optional spezifizierte Bedingung für vorzeitigen Abbruch erfüllt ist.

Mit der Klasse GeometricVisitor steht eine fertige und – aus den gleichen Gründen wie oben bei den Iteratoren – separate Implementierung bereit. Die auf die Zellen anzuwendende Funktion ist frei wählbar; sie implementiert das Interface IVisitor und ist für alle Zellen gleich. Zugriff auf eine Instanz erhält man über das Messwertgrid-Interface IMeasurementGrid, wobei die Funktion und das Messwertgrid über Template-Parameter festgelegt werden. Für das Hauptgrid steht diese Methode mangels Bedarf nicht zur Verfügung.

Eine weitere Methode zur Navigation ist ein spezielles *Getter-Konzept*. Ein Grid ist eine Datenstruktur aus verknüpften Zellen. Im Rahmen der Navigation kann deshalb nicht wahlfrei auf die Zelle an einer bestimmten Zielposition zugegriffen werden; die Navigation erfolgt immer sequentiell, ausgehend von einer bekannten Referenzposition. Zudem kann nicht direkt von Zelle zu Zelle navigiert werden (siehe oben). Möchte man auf mehrere benachbarte Zielpositionen zugreifen, die weit von der Referenzposition entfernt liegen, ist dies deshalb nur äußerst ineffizient möglich – trotz der Nachbarschaft muss die Navigation jedes Mal von der Referenzposition aus erfolgen.

Das hier vorgestellte Konzept erlaubt in diesem Fall eine effizientere Navigation, indem die Referenzposition je nach Bedarf gewählt werden kann. Dies geschieht mithilfe zweier Funktionen:

- Die *get*-Funktion navigiert ausgehend von der aktuellen Referenzposition zur gewünschten Zielposition und liefert die dortige Zelle als Ergebnis. Die Referenzposition bleibt unverändert; initial entspricht sie der Position des Gridmittelpunkts.

[11] Die Zellen des verwendeten Grids müssen dazu lediglich das Interface IFourNeighboursCell implementieren.

[12] Aus Gründen der Effizienz ist zusätzlich die Verwendung einer beidseitigen *friend*-Deklaration notwendig. Deren Verwendung scheint in diesem Fall gerechtfertigt, da sie hier zu einer *Erhöhung* der Datenkapselung beiträgt.

– Die *getAndSet*-Funktion gleicht prinzipiell der *get*-Funktion, allerdings wird die Zielposition ab sofort als neue Referenzposition verwendet.

Die beiden Funktionen stehen über das zugehörige Interface für Haupt- und Messwertgrid zur Verfügung (Interface IMainGrid bzw. IMeasurementGrid).

4.2.3.3 Zugriff & Fusion

Der Zugriff auf eine einzelne Messwert- bzw. Hauptzelle erfolgt über das zugehörige Zell-Interface IMeasurementCell bzw. IMainCell. Dieser Zugriff ist an vielen Stellen eng mit dem Fusionsprozess verwoben, weshalb dieser zunächst betrachtet werden soll.

Die Fusion von Daten ist ein sehr zentrales Thema der OG-Bibliothek, denn es müssen Messwerte aus unterschiedlichen Quellen und von unterschiedlichen Zeitpunkten miteinander kombiniert werden. Wie in Abschnitt 4.2.2.1 beschrieben, werden, um dies zu ermöglichen, zum Einen Messwert- und Hauptgrid unterschieden, zum Anderen besteht das Messwertgrid aus mehreren Ebenen. Fusion tritt dabei in drei unterschiedlichen Situationen auf (vgl. Abbildung 4.3):

– Die *sensorlokale Fusion* akkumuliert die Daten eines einzelnen Sensors zwischen zwei Sensorenfusionen. Dies erfolgt in der dazu vorhandenen, sensorspezifischen Akkumulationsebene des Messwertgrids.

– Die *Sensorenfusion* kombiniert die Daten aller sensorspezifischer Akkumulationsebenen zu einem Gesamt-Messwertgrid. Dieses enthält dadurch alle seit der letzten Gridfusion eingegangenen Daten in fusionierter Form.

– Die *Gridfusion* integriert das Gesamt-Messwertgrid in das Hauptgrid, das bereits alle vorherigen Gridfusionen in fusionierter Form enthält.

Das prinzipielle Konzept dieser Fusionen ist immer gleich: Die eigentlichen Fusionsoperationen sind in Funktionsobjekten gekapselt, welche innerhalb von Containern ausgeführt werden und die Fusion von Belegtheits-, Dynamik- oder Konfliktwerten erlauben. Die Verwendung von Funktionsobjekten dient der Kapselung der eigentlichen Fusionsoperation, da diese vom Typ des Belegtheits- bzw. Dynamikwerts und der Fusionssituation abhängt. Container (Interface ICellDataFusionContainer) wurden in Abschnitt 4.2.3.1 im Rahmen der Beschreibung des Zellaufbaus bereits eingeführt, da sie den Belegtheits- bzw. Dynamikwert (dann allgemein als Basiswert bezeichnet) einer Zelle um einen Konfliktwert ergänzen. Das Container-Konzept leistet aber mehr: Es kapselt die gesamte Fusion. Konkret ruft der Container bis zu zwei Fusionsoperationen auf seinen Werten auf, wozu ihm maximal zwei Funktionsobjekte und der zu fusionierende Wert zum Zeitpunkt der Fusion als Template-Parameter übergeben werden:

– Wird *kein* Funktionsobjekt übergeben, ersetzt der übergebene Wert den bisherigen Basiswert, ohne dass irgendeine Fusion stattfindet. Folglich tritt auch kein Konflikt auf.

– Wird *ein* Funktionsobjekt übergeben, dient dieses zur Fusion des bisherigen Basiswerts mit dem übergebenen Wert. Das Fusionsergebnis ersetzt den bisherigen Basiswert, der bei der Fusion entstandene Konfliktwert ersetzt den bisherigen Konfliktwert.

– Werden *zwei* Funktionsobjekte übergeben, wird das erste wie eben beschrieben verwendet. Das zweite fusioniert jedoch den dabei entstandenen Konfliktwert mit dem bisherigen Konfliktwert, das Fusionsergebnis wird als neuer Konfliktwert gespeichert.

Die exemplarische Implementierung verwendet beispielsweise Dempsters Kombinationsregel als Fusionsoperation zur Kombination der Basiswerte (vgl. Abschnitt 3.2.2). Dabei entsteht k als Konfliktwert, der ebenfalls fusioniert werden soll. Leider kann k dazu nicht einfach addiert werden, da dieses nicht additiv ist. Stattdessen kann man den Wert der auf k basierenden und additiven Con-Metrik als Konfliktwert verwenden und diesen durch Addition akkumulieren. Allerdings handelt es sich dabei um ein logarithmisches Maß mit dem Wertebereich $[0,\infty[$. Die Logarithmierung macht die Berechnung aufwendiger und der unendliche Wertebereich ist für die Implementierung und Weiterverarbeitung unpraktisch, weshalb wir lieber direkt mit k arbeiten wollen.

Aus der Additivität der Con-Metrik lässt sich eine Gleichung herleiten, die zum Zeitpunkt t die erwünschte direkte Kombination des aktuellen Konfliktwerts k_t mit dem bisherigen akkumulierten Konfliktwert $k_{1:t-1}$ zum neuen akkumulierten Konfliktwert $k_{1:t}$ erlaubt. Sind Con_t, $Con_{1:t-1}$ und $Con_{1:t}$ die auf k_t, $k_{1:t-1}$ bzw. $k_{1:t}$ basierenden Con-Metriken, gilt aufgrund deren Additivität:

$$Con_{1:t} = Con_{1:t-1} + Con_t.$$

Unter Verwendung der Definition der Con-Metrik $Con = -log(1-k)$ folgt daraus:

$$-log(1-k_{1:t}) = -log(1-k_{1:t-1}) - log(1-k_t).$$

Durch Vereinfachung ergibt sich:

$$k_{1:t} = k_{1:t-1} + k_t - k_{1:t-1} * k_t.$$

Diese Gleichung wird in der Implementierung als *Konfliktsumme* bezeichnet. Sie erlaubt die erwünschte direkte Kombination von k und kann deshalb als Fusionsoperation zur Kombination der Konfliktwerte verwendet werden.

Wir haben oben drei unterschiedliche Situationen gesehen, in denen Fusionen auftreten. Wie sich gleich zeigen wird, sind diese Fusionen eng verwoben mit dem Zugriff auf einzelne Mess- bzw. Hauptzellen, die über deren Interfaces IMeasurementCell bzw. IMainCell erfolgen können. Während das Fusionskonzept unabhängig vom Typ immer gleich ist (insbesondere ist der Ausführungsort jeder Fusion ein Container), ist die Zuständigkeit unterschiedlich, das heißt, wer die Fusion mit welchen Parametern (zu fusionierender Wert und bis zu zwei Funktionsobjekte) veranlasst.

Für die sensorlokale Fusion sind die Mapper zuständig. Jeder Mapper kartiert zellweise die Daten des im zugeordneten Sensors in der zugehörigen, sensorspezifischen Kartierungsebene des Messwertgrids (vgl. Abbildung 4.3). Dazu stehen ihm über das Zell-Interface IMeasurementCell eine Reihe von Funktionen zur Verfügung. Fusion spielt dabei keine Rolle, da in dieser Ebene keine Akkumulation stattfindet; der Mapper kann in jeder Zelle dieser Ebene die Basiswerte direkt lesen, schreiben oder zurücksetzen.

Sobald der Mapper die Bearbeitung der Kartierungsebene abgeschlossen hat, führt er zur zeitlichen Integration der kartierten Daten eine sensorlokale Fusion durch. Dazu ruft

er eine entsprechende Funktion des Interfaces IMeasurementCell mit bis zu zwei Funktionsobjekten auf, die die Fusionsoperation wie üblich kapseln (siehe oben). Diese Fusionsoperationen integrieren die Kartierungsebene zellweise in die ebenfalls sensorspezifische Akkumulationsebene; die Kartierungsebene wird anschließend zurückgesetzt.

Wurde von jedem Sensor über dessen Mapper wenigstens ein Messwert auf diese Weise kartiert, wird das Messwertgrid zeitlich in das Hauptgrid integriert und anschließend zurückgesetzt, also die Gridfusion durchgeführt. Diese fällt in die Zuständigkeit des OGBuilder. Zur Ausführung ruft der OGBuilder ein Funktionsobjekt auf und übergibt dabei das aktuelle Messwertgrid und das derzeitige Hauptgrid. Das Funktionsobjekt führt dann zellweise die gewünschte zeitliche Integration aus, indem es je Zelle und Wert die entsprechende Funktion des Hauptzellen-Interfaces IMainCell aufruft und dabei den entsprechenden Wert der korrespondierenden Messwertzelle und bis zu zwei Funktionsobjekte übergibt, die wie üblich die Fusionsoperationen kapseln (siehe oben).

Das Funktionsobjekt implementiert das Interface IGridFusioner und dient der Kapselung der Gridfusions-Implementierung. Durch diese Kapselung kann leicht – ohne Änderungen am OGBuilder, sondern durch bloße Anpassung der entsprechenden Typedef im Header GridDefinitions – eine Gridfusions-Implementierung gewählt werden, die zum Typ der Basiswerte passt, die gewünschten Fusionsoperationen verwendet und in geeigneter Weise den aufgetretenen, durch die Konfliktwerte angezeigten Konflikt nutzt (vgl. unten vorgestellte Beispiel-Implementierung in Form der Klasse GridDSFusioner).

Wie beschrieben greift das IGridFusioner-Funktionsobjekt während der Gridfusion lesend auf die Messwertzellen zu. Allerdings liegen die Zellinhalte intern nur getrennt vor – sensorlokal akkumuliert in den sensorspezifischen Akkumulationsebenen. Im Rahmen des Zellzugriffs wird deshalb die Sensorenfusion ausgeführt, wodurch ein einzelner, kombinierter Zellinhalt entsteht, auf den zugegriffen werden kann. Auch beim eigentlich nur lesenenden Zugriff auf eine Messwertgridzelle werden deshalb bis zu zwei Funktionsobjekte übergeben, die die Fusionsoperations-Implementierungen enthalten und den Umgang mit Basis- und Konfliktwerten festlegen (siehe oben).

Die beiden Interfaces IMeasurementCell und IMainCell bieten noch einige weitere Funktionen zum Zellzugriff, die teilweise weitere Funktionalität bereitstellen und teilweise in einigen Fällen eine effizientere Vorgehensweise erlauben. Für eine vollständige Darstellung wird auf die Implementierung und ihre Dokumentation verwiesen.

Für die in unserem Beispiel-Anwendungsszenario verwendete Dempster-Shafer-Theorie findet sich mit der Klasse GridDSFusioner eine exemplarische Implementierung eines IGridFusioner-Funktionsobjekts, das wie beschrieben die Grid- und dabei auch die Sensorenfusion durchführt. Für beide Fusionen gilt: Beide Basiswerte werden mittels Dempsters Kombinationsregel fusioniert, die Belegtheits-Konfliktwerte werden mittels Konfliktsumme fusioniert und die Dynamik-Konfliktwerte werden nicht fusioniert.

Der bei der Gridfusion entstehende Konfliktwert wird, da es sich bei dieser Fusion um eine zeitliche Integration derselben Sensorik handelt, als Indiz für das Vorhandensein eines dynamischen Objekts an dieser Stelle gewertet, weshalb die Dynamikmasse an der betrachteten Stelle entsprechend erhöht wird. Ein höherer Konfliktwert als Ergebnis der Sensorenfusion lässt dagegen keine direkten Rückschlüsse auf die tatsächliche Situation zu, da jeder Sensor als gleich zuverlässig angesehen wird[13]. Der Widerspruch führt deshalb dazu, dass

[13]Unterschiedliche Zuverlässigkeiten, wie zum Beispiel unterschiedliche Messgenauigkeiten, sind bereits im jeweiligen Sensormodell berücksichtigt.

die Masse der Evidenz, über die Uneinigkeit herrscht, in Abhängigkeit von der Höhe des Konflikts reduziert und die zugehörige *unbekannt*-Masse in gleichem Maße erhöht wird.

4.2.4 Die Mapper

Das Konzept der Mapper wurde einleitend bereits grundlegend eingeführt. Es ist daher bekannt, *dass* der Fahrzeugmapper in erster Linie der Bestimmung der Eigenpose dient und die Sensormapper die Umfelddaten kartieren. In diesem Abschnitt wird nun beschrieben, *wie* das geschieht.

Im Folgenden werden dazu die funktionellen Aufgaben sowie ihre möglichen Umsetzungen (insbesondere die für unseren Anwendungsfall gewählte) beschrieben. Vorweg soll jedoch mit der Kapselung aller Spezifika eine wichtige konzeptionelle Aufgabe besprochen werden, die für beide Mapper gleich ist.

Im Falle einer Fahrzeugmapper-Implementierung ist beispielsweise eine Wahl zu treffen, wie die Schätzung der Fahrzeugpose erfolgt und wie – falls überhaupt – die Fahrzeugeigenfläche kartiert wird. Ähnlich muss für eine Sensormapper-Implementierung eine Entscheidung getroffen werden, welches inverses Sensormodell im Rahmen der Kartierung verwendet wird. Zudem sind beide Mapper-Implementierungen auf bestimmte Strukturen für die zugrundeliegenden Sensor- und Konfigurationsdaten sowie auf eine bestimmte Messwertgrid-Implementierung mit einer bestimmten Art von Evidenzen festgelegt – und auch das wird gekapselt.

Die Kapselung der eingesetzten Mapper-Implementierung ist aufgrund ihrer damit verbundenen leichten Austauschbarkeit für den anwendungsunabhängigen Ansatz der OG-Bibliothek sehr wichtig. Um diese zu erleichtern, sind wie üblich Interfaces definiert: IVehicleMapper und ISensorMapper für den Fahrzeug- bzw. Sensormapper. Bei deren Implementierung werden das verwendete Messwertgrid und die verwendeten Strukturen für die benötigten Sensor- und Konfigurationsdaten als Template-Parameter übergeben.

Mit den Klassen LidarDSMapper, SMRRadarDSMapper und VehicleDSMapper stehen exemplarische Mapper-Implementierungen für unseren Anwendungsfall bereit. Welche Mapper-Implementierungen zum Einsatz kommen, wird über die zugehörigen Typedefs im Header GridDefinitions ausgewählt. Handelt es sich um Mapper für bisher noch nicht unterstützte Sensorarten (z.B. Sonarsensor), sind zusätzlich kleinere Anpassungen im OGBuilder notwendig, die bei Betrachtung der Implementierung offensichtlich sind.

Eine weitere Gemeinsamkeit von Sensor- und Fahrzeugmapper ist ihre Instanziierung. Das externe Programm, das die OG-Bibliothek verwendet, erzeugt ganz zu Beginn eine Instanz des OGBuilder als Schnittstelle (vgl. Abschnitt 4.1.3). Im Rahmen seiner eigenen Instanziierung erzeugt der OGBuilder, auf Basis der ihm zur Verfügung gestellten Konfigurationsdaten, auch die benötigten Instanzen der Mapper – ein Mapper je Umfeldsensor und zusätzlich ein Mapper für das Fahrzeug selbst. Der OGBuilder überreicht ihnen dabei eine Reihe von Informationen, die sie – wie sich gleich zeigen wird – für ihre spätere Arbeit benötigen: die vom Benutzer über die graphische Benutzeroberfläche gewählte Zellgröße in Zentimeter, die ihnen jeweils zur sensorlokalen Kartierung zugeteilte Ebene des Messwertgrids (vgl. Abschnitt 4.2.2.1) und die Konfigurationsdaten des ihnen zugeordneten Sensors und des Fahrzeugs.

4.2.4.1 Fahrzeugmapper

Der Fahrzeugmapper dient hauptsächlich der Bestimmung der Pose des eigenen Fahrzeugs im Grid. Im Allgemeinen erfolgt dies nicht nur zellgenau (Struktur CellPosition), sondern auch die Position innerhalb der Zelle wird ermittelt (Struktur InCellPosition).

Als Eingabe erhält der Fahrzeugmapper dazu die jeweils aktuellen Eigenzustandsdaten (Typedef VehicleRawData) vom OGBuilder. Wie diese Daten zu Stande kommen und woraus sie bestehen, ist abhängig vom jeweiligen Anwendungsfall. Im Idealfall enthalten die Eigenzustandsdaten bereits die genaue Pose in einem anwendungsspezifischen Koordinatensystem. Die Aufgabe des Mappers besteht dann lediglich darin, diese Pose in das Gridkoordinatensystem zu überführen. In der Praxis ist dies jedoch oftmals nicht der Fall, so dass der Mapper die Eigenpose auf Basis der verfügbaren Eigenzustandsdaten schätzen muss. In Abschnitt 3.3.1 wurden Verfahren vorgestellt, die dafür üblicherweise eingesetzt werden.

Das Ergebnis der Eigenposenbestimmung stellt der Mapper zusammen mit weiteren Informationen zu Eigenzustand und Fahrzeugkonfiguration in der Struktur VehicleData bereit.

Optional kann der Fahrzeugmapper auch verwendet werden, um die vom eigenen Fahrzeug belegte Fläche zu kartieren [50]. Die dahinterstehende Überlegung ist, dass aus dem Umstand, dass das Fahrzeug sich gerade an der jeweiligen Position befindet, Wissen extrahiert werden kann, von dem das Occupancy Grid möglicherweise profitiert. Konkret: Man weiß, dass die gerade vom Eigenfahrzeug eingenommenen Zellen durch ein dynamisches Objekt belegt sind – entsprechend kann für diese Zellen beispielsweise ein niedriger Belegtheits- und ein hoher Dynamikwert in das Messwertgrid fusioniert werden. Der Fahrzeugmapper besitzt dort dazu eine eigene Kartierungsebene – analog zu einem Sensormapper, denn das eigene Fahrzeug liefert Daten über das Umfeld und wird entsprechend als weiterer, gleichberechtigter „Sensor" neben den Umfeldsensoren betrachtet.

Die Klasse VehicleDSMapper implementiert das Interface IVehicleMapper und stellt eine exemplarische Fahrzeugmapper-Implementierung für unseren Anwendungsfall dar. Die Verwendung ist nur mit der dort eingesetzten Dempster-Shafer-Theorie und den dort eingesetzten Strukturen für die Fahrzeugkonfigurationsdaten und Eigenzustandsdaten möglich (vgl. Abschnitt 4.1.1) – dies ist unvermeidbar, da die Implementierung direkt damit arbeiten muss.

Die Klasse VehicleDSMapper sieht neben einer Schätzung der Eigenpose auch eine Kartierung der eigenen Fahrzeugfläche vor. Letztere erfolgt, indem für die durch das eigene Fahrzeug belegten Zellen hohe Evidenzen für *frei* (da befahrbar und deshalb nicht durch ein statisches Objekt belegt) und *dynamisch* (da das eigene Fahrzeug ein dynamisches Objekt ist) fusioniert werden (siehe oben).

Die Schätzung der Eigenpose ist notwendig, da diese nicht direkt in den verfügbaren Eigenzustandsdaten enthalten ist. Sie wird deshalb – wie in diesem Fall üblich (vgl. Abschnitt 3.3.1) – unter Verwendung eines Bewegungsmodells aus den verfügbaren Messwerten der Eigenzustandssensorik geschätzt. Es kommt das *Modell konstanter Gierrate und Beschleunigung* (engl. Constant Turn Rate and Acceleration, CTRA) zum Einsatz, da es im Vergleich mit anderen Bewegungsmodellen gute Resultate erzielt [43]. Zudem stehen die als Eingangsdaten benötigten Messwerte (Gierrate und longitudinale Beschleunigung) nicht nur in unserem Anwendungsfall, sondern generell oft zur Verfügung, was die Wieder-

verwendbarkeit dieser Beispiel-Implementierung erhöht.

Das CTRA-Modell besitzt den Zustandsraum $\vec{s} = (x,y,\theta,v,a,\omega)^T$, wobei (x,y) die Position, θ die Orientierung (d.h. den Gierwinkel), v die longitudinale Geschwindigkeit, a die longitudinale Beschleunigung und ω die Gierrate des Fahrzeugs beschreibt. Sei T der Zeitraum zwischen der letzten Aktualisierung zum Zeitpunkt t und dem aktuellen Zeitpunkt $t+T$, zu dem neue Messwerte der longitudinalen Beschleunigung a und der Gierrate ω eintreffen. Diese beiden Größen werden als konstant über den Zeitraum T angenommen, daher der Name des Modells. Damit ergibt sich als Gleichung für den Zustandsübergang nach [43]:

$$\vec{s}(t+T) = \begin{pmatrix} x(t+T) \\ y(t+T) \\ \theta(t+T) \\ v(t+T) \\ a \\ \omega \end{pmatrix} := \vec{s}(t) + \begin{pmatrix} \Delta x(T) \\ \Delta y(T) \\ \omega T \\ aT \\ 0 \\ 0 \end{pmatrix}$$

mit

$$\Delta x(T) := \frac{1}{\omega^2}[(v(t)\omega + a\omega T)\sin(\theta(t)+\omega T) + a\cos(\theta(t)+\omega T) - v(t)\omega\sin(\theta(t)) - a\cos(\theta(t))],$$

$$\Delta y(T) := \frac{1}{\omega^2}[(-v(t)\omega - a\omega T)\cos(\theta(t)+\omega T) + a\sin(\theta(t)+\omega T) + v(t)\omega\cos(\theta(t)) - a\sin(\theta(t))].$$

Eine theoretische Voraussetzung des Occupancy Grid Mappings ist, dass die Position des eigenen Fahrzeugs exakt bekannt ist – dies ist in unserem Anwendungsfall nicht erfüllt. Obwohl das CTRA-Modell offensichtlich einen relativ hohen Berechnungsaufwand mit sich bringt, scheint sein Einsatz gerechtfertigt, um die Voraussetzung *bestmöglich* mit den vorhandenen Mitteln zu erfüllen.

4.2.4.2 Sensormapper

Jeder Sensormapper soll die Messergebnisse des ihm zugeordneten Umfeldsensors kartieren. Wie diese Messwerte zu Stande kommen und woraus sie bestehen, ist abhängig vom jeweiligen Anwendungsfall. Der prinzipielle Ablauf ist aber in der Regel gleich.

Zunächst erhält der Sensormapper den jeweils aktuellen Messwert seines Umfeldsensors und die aktuelle Eigenpose vom OGBuilder als Eingabe. Entsprechend der aktuellen Pose wird dann der vom Messwertgrid repräsentierte Bereich so angepasst, dass das Sichtfeld des Sensors und dadurch auch sein Messwert im Grid Platz haben (vgl. Abschnitt 4.2.5.1). Vor der Kartierung des Messwerts muss dieser im Allgemeinen noch aus seinem sensorspezifischen in das Gridkoordinatensystem transformiert werden (vgl. Abschnitt 4.2.2.2), wozu der Mapper die aktuelle Pose des Fahrzeugs, die Zellgröße und die initial vom OGBuilder erhaltenen Sensor- und Fahrzeugkonfigurationsdaten verwendet. Nach der Koordinatensystemtransformation kann der Messwert kartiert werden. Dazu ist ein inverses Sensormodell notwendig (vgl. Abschnitt 3.1.2), das ihn in eine Evidenz für Belegtheit und/oder Dynamik überführt und diese an der richtigen Stelle im Messwertgrid einträgt (in der diesem Sensormapper zugeteilten Grid-Ebene, vgl. Abschnitt 4.2.2.1).

Die Klassen LidarDSMapper und SMRRadarDSMapper implementieren beide das Interface ISensorMapper. Sie stellen Sensormapper-Implementierungen für ein Lidar

bzw. Radar entsprechend unseres Anwendungsfalls dar. Die Verwendung ist nur mit der dort eingesetzten Dempster-Shafer-Theorie und den dort eingesetzten Strukturen für die Konfigurations- und Sensordaten (vgl. Abschnitt 4.1.1) möglich – dies ist unvermeidbar, da die Implementierung direkt damit arbeiten muss.

Betrachten wir zunächst die Klasse SMRRadarDSMapper, die Sensormapper-Implementierung für unseren Radarsensor.

Dieser liefert eine Liste erkannter Hindernisse, wobei dabei zu jedem Hindernis unter anderem dessen Geschwindigkeit relativ zum Eigenfahrzeug sowie seine radiale Entfernung und Richtung gegeben sind. Wie für Radarsensoren üblich, liefert er jedoch keine Aussagen über die Größe dieses punktförmig gegebenen Hindernisses und besitzt eine relativ große Messunsicherheit, welche sich mit zunehmender Hindernisentfernung in lateraler Richtung noch vergrößert.

Üblicherweise wird aufgrund dieser Messunsicherheit der Wert der Messung nicht punktuell, sondern anteilig in einer größeren Region kartiert, häufig mithilfe einer bivariaten Gaußverteilung (vgl. Abschnitt 3.1.2). Eine derartige Vorgehensweise verwendet auch die Klasse SMRRadarDSMapper.

Hier wird die Messunsicherheit des Sensors mithilfe einer modifizierten, bivariaten Gaußfunktion modelliert, bei der der Normalisierungsterm durch einen entfernungsabhängigen Term ersetzt wurde, so dass der resultierende Funktionswert direkt als Evidenz für *belegt* kartiert werden kann. Als Funktionsparameter verwendet der Mapper die in der Sensorkonfigurationsdatei enthaltene longitudinale und laterale Standardabweichung der Messung. Zusätzlich wird für jedes Hindernis eine bestimmte Größe um seine punktförmige Messstelle angenommen. Innerhalb eines entsprechenden, zweidimensionalen Bereichs wird dazu gleichmäßig der maximale Funktionswert als Evidenz für *belegt* kartiert. Außerhalb sorgt die Gaußfunktion für eine stufenweise Reduktion der Evidenz für *belegt* bis in gewissem Abstand völliges Unwissen angenommen wird.

Aufgrund der nur punktförmig gegebenen Hindernisse wird auf eine Freiraummodellierung verzichtet; da auf diese Weise nur *belegte* Zellen erkannt werden können, entfällt auch die sensorlokale, konfliktbasierte Dynamikerkennung (vgl. Abschnitt 4.2.3.1), die auf dem Wechsel zwischen *frei* und *belegt* basiert.

Anschließend wird der Dynamikwert bestimmt. Hierzu wird zunächst aus der gegebenen relativen die absolute Geschwindigkeit berechnet; anschließend führt die Anwendung eines doppelten Schwellwerts zum Dynamikwert.

Berechnete Belegtheits- und Dynamikwerte werden jeweils zellweise mithilfe Dempsters Kombinationsregel in die sensorspezifische Ebene des Messwertgrids integriert. Das Kartierungsergebnis ist ganz links in Abbildung 4.11 gezeigt; die einzelnen Messpunkte sind klar zu erkennen.

Die Klasse LidarDSMapper enthält die Sensormapper-Implementierung für unseren Lasersensor.

Er liefert vorverarbeitete Messwerte in Form einer Menge von erkannten Objekten. Zu jedem Objekt werden Position und fast immer Geschwindigkeit im kartesischen Sensorkoordinatensystem bereitgestellt. Außerdem sind meist auch Konturpunkte und eine Bounding Box (beschrieben durch Position, Größe und Orientierung) gegeben.

Die Klasse LidarDSMapper verarbeitet diese Daten mithilfe eines zweistufigen Sensormodells. Ein zweistufiger Ansatz ist üblich, um neben den gelieferten Objekten auch die wichtige – aber nur implizit vorhandene – Freirauminformation zu verwerten (vgl. Abschnitt

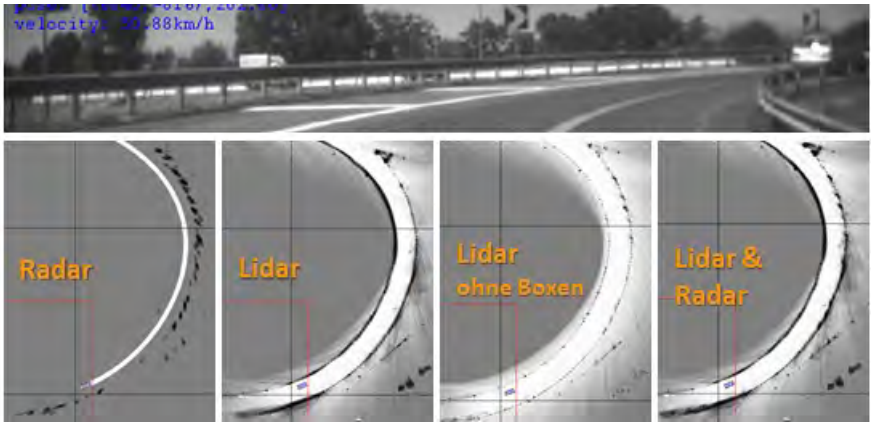

Abbildung 4.11: Kartierungsergebnisse der exemplarischen Sensormapper.

3.1.2).

Sind nur Konturpunkte für ein Objekt vorhanden, werden die zugehörigen Zellen einzeln als *belegt* kartiert. Ist eine Bounding Box vorhanden, werden allen in der Box enthaltenen Zellen gleichmäßig als *belegt* kartiert; die Zellen, in denen ein Konturpunkt liegt, erhalten jedoch einen leicht höheren Belegtheitswert als die restlichen. Wie ein Vergleich der beiden mittleren Grafiken in Abbildung 4.11 deutlich macht, führt die Verwendung der Boxen zwar insgesamt zu etwas gröberen, dafür aber zu viel kontrastreicheren Konturen; der Einsatz der Boxen ist also sehr lohnend.

Wurden alle Objektdaten in das Messwertgrid eingetragen, erfolgt die Freiraummodellierung. Mangels einer direkten Messung des Freiraums wird das in Abschnitt 3.1.2 beschriebene Raycasting eingesetzt, bei dem ausgehend vom Sensor die Strahlengänge der einzelnen Laserstrahlen modelliert werden. Es kommt eine lineare Freiraumfunktion zum Einsatz, die den Belegtheitswert in Abhängigkeit von der Entfernung ausdrückt: Je weiter sich die betrachtete, auf dem Strahl liegende Zelle vom Sensor entfernt befindet, desto höher ist die dort kartierte Evidenz für *frei*. Da es sich bei dem eingesetzten Lidar um einen Mehrebenen-Lidar handelt, können mehrere Hindernisse entlang eines einzelnen modellierten Laserstrahls auftreten. In diesem Fall wird nur der Bereich vor dem ersten Messwert als *frei* kartiert, die Bereiche zwischen den Hindernissen und hinter dem letzten dagegen als *unbekannt*. Auf den Einsatz einer komplexeren Funktion, die beispielsweise eine unterschiedliche Randbehandlung realisiert (vgl. Abschnitt 3.1.2), wird verzichtet, da die Winkelauflösung unseres Lidars konstant ist. Allerdings ist in den Randbereichen eine zu dominante Freiraummodellierung erkennbar, was auf eine schlechtere Hindernisdetektion in diesem Bereich zurückzuführen ist. Um die dortige Dominanz der Freiraummodellierung zu vermeiden, wird der Öffnungswinkel der Strahlen zur Freiraummodellierung gegenüber dem tatsächlichen Öffnungswinkel des Sensors reduziert.

Für alle von der Objektkartierung oder Freiraummodellierung betroffenen Zellen wird zudem der Dynamikwert bestimmt, basierend auf der sensorseitig durchgeführten Objektklassifikation. Als *dynamisch* werden diejenigen Zellen kartiert, in denen Messungen liegen, die den Klassen *PKW* oder *LKW* zugeordnet wurden; alle übrigen als *statisch*[14].

[14]Weitere Objektklassen (z.B. Motorräder, Fahrräder) und die Geschwindigkeitsmessungen das Lasersensors sollten berücksichtigt werden – leider war dies aus den später in Abschnitt 5.1 beschriebenen Gründen nicht möglich.

Die Belegtheitswerte werden zellweise mithilfe Dempsters Kombinationsregel integriert. Ein dabei in einer Zelle auftretender Konflikt (durch eine Belegtheitsänderung von *frei* zu *belegt* oder von *belegt* zu *frei*) wird als Indiz für ein dynamisches Objekt gewertet (vgl. Abschnitt 3.3.1) und führt deshalb zu einer Erhöhung des Dynamikwerts. Dieser Fusion der Belegtheitswerte folgt die Integration der Dynamikwerte, ebenfalls zellweise mithilfe Dempsters Kombinationsregel.

Das Kartierungsergebnis dieses Sensormappers ist in Abbildung 4.11 in der zweiten Grafik von links gezeigt. Ganz rechts ist zudem der deutliche Gewinn ersichtlich, den eine gleichzeitige Verwendung beider Sensoren und damit Sensormapper bringt. In vorliegender Szene ist dies insbesondere im Bereich oben rechts erkennbar.

Weiterführende Informationen über die genaue Algorithmik und die konkrete Höhe der einzelnen Evidenzen liefert ein Studium der Implementierung (siehe DVD im Anhang).

4.2.5 Die Grids

In diesem Abschnitt soll die Beschreibung der OG-Bibliothek mit einer Besprechung ihrer drei Grids abgeschlossen werden. Messwert- und Hauptgrid werden nur bibliotheksintern verwendet und sind sich in vielerlei Hinsicht ähnlich. Ihre Datenstrukturen sollen einen effizienten Zugriff auf jede Zelle und eine effiziente Anpassung des repräsentierten Bereichs an die Fahrzeugbewegung ermöglichen – und dabei möglichst generisch aufgebaut sein. Das Occupancy Grid dagegen wird für eine geeignete Ergebnisrepräsentation nach außen eingesetzt, entsprechend ist das Konzept dahinter ein völlig anderes.

4.2.5.1 Messwertgrid

Das Interface IMeasurementGrid kapselt die dadurch leicht austauschbare Implementierung des Messwertgrids. Das Messwertgrid dient der effizienten Kartierung von Messwerten im Umfeld des Eigenfahrzeugs. Die eigentliche Kartierung findet zellweise statt und wird deshalb bereits auf Zellebene unterstützt (vgl. Abschnitt 4.2.3). Die Aufgaben des Messwertgrids selbst bestehen deshalb darin, Möglichkeiten zur effizienten Navigation über einzelne Gridbereiche zu gewähren und den dargestellten Gridausschnitt an die aktuelle Fahrzeugposition anzupassen. Die effizienten Navigationsmöglichkeiten wurden schon in Abschnitt 4.2.3.2 besprochen. Hier konzentrieren wir uns deswegen auf die Anpassung des darzustellenden Bereichs an die aktuelle Fahrzeugposition.

Das Messwertgrid verwendet eine erdfeste Darstellung, das Fahrzeug bewegt sich also über das Grid (vgl. Abschnitt 4.2.2.2). Verändert das Fahrzeug seine Pose, so teilt der OGBuilder dem Grid die neue Pose im Gridkoordinatensystem mit.

Im Messwertgrid werden alle Messungen akkumuliert, die zwischen zwei aufeinanderfolgenden Gridfusionen eingehen, das heißt, zwischen zwei Integrationen des Messwertgrids in das Hauptgrid (vgl. Abschnitt 4.2.3.3). Da sich das Eigenfahrzeug im Allgemeinen in der Zwischenzeit fortbewegt und zwar in jede Richtung gleich wahrscheinlich, wird das Messwertgrid ausreichend groß und seine Position im Gridkoordinatensystem derart gewählt, dass sich das Fahrzeug initial beliebig orientiert in dessen Zentrum befindet. Sobald sich das Fahrzeug bewegt, ist es jedoch wahrscheinlich, dass es sich längere Zeit in nur eine Richtung über das Grid bewegt – dadurch werden große Teile des Grids nicht „befahren" und die Bearbeitung der gesamten bereitgestellten Größe ist ineffizient.

Das Messwertgrid unterscheidet deshalb zwischen seiner *logischen* und seiner *realen Größe*. Der betrachtete Gridausschnitt wird logisch auf den wirklich benötigten Bereich begrenzt, obwohl die reale Größe der zugrundeliegenden Datenstruktur weit größer ist. Das Vorhandensein einer ausreichend großen realen Datenstruktur vermeidet das aufwendige und zeitintensive Anlegen neuer Zellen (inklusive ihrer Nachbarschaftsbeziehungen) zur Laufzeit. Die zu verwendende reale Größe wird dem Messwertgrid bei seiner Instanziierung durch den OGBuilder mitgeteilt, der bei der Festlegung die maximale Reichweite der Sensorik, die maximal vom Eigenfahrzeug zurückgelegte Distanz und die Größe des Hauptgrids[15] berücksichtigt.

Das Fahrzeug bewegt sich nun über das Messwertgrid und eingehende Messwerte werden durch die Mapper darin kartiert. Das Messwertgrid passt dabei seine logische Größe entsprechend des jeweils aktuellen Größenbedarfs der Mapper zur Kartierung an – wie das geschieht, ist abhängig von der gewählten Messwertgrid-Implementierung (siehe auch Beispiel unten). Im Rahmen der nächsten Gridfusion wird das Messwertgrid in das Hauptgrid integriert und anschließend zurückgesetzt, das heißt, alle kartierten Daten werden verworfen. Das Fahrzeug wird anschließend wieder initial im Zentrum des Grids angenommen; da sich das Fahrzeug nun aber im Allgemeinen an einer anderen Position befindet, befindet sich der repräsentierte Gridausschnitt jetzt an einer anderen Stelle im Gridkoordinatensystem als der vorherige.

Die Klasse GrowGrid implementiert das Interface IMeasurementGrid und stellt damit eine fertige Messwertgrid-Implementierung bereit. Mittels Template-Parametern können unterschiedliche IMeasurementCell-Zellen und Iteratoren verwendet werden.

Wenn das GrowGrid mit einer bestimmten realen Größe instanziiert wird, legt es eine entsprechende Anzahl an Zellen an. Das GrowGrid verbindet diese zu seiner Gitterstruktur, indem es für jede innen liegende Zelle vier Nachbarn definiert, für die Zellen an den Rändern entsprechend weniger. Die logische Größe ist initial und nach jeder Gridfusion null. Verlässt das Fahrzeug den aktuellen logischen Gridausschnitt oder fordert einer der Mapper die Bereitstellung einer Fläche außerhalb an, wird der logische Bereich innerhalb der Grenzen der realen Gitterstruktur erhöht (vgl. Abbildung 4.12). Da das Grid dadurch schrittweise, entsprechend der Fahrzeugbewegung logisch „wächst" (engl. to grow), wurde diese Messwertgrid-Implementierung entsprechend GrowGrid getauft.

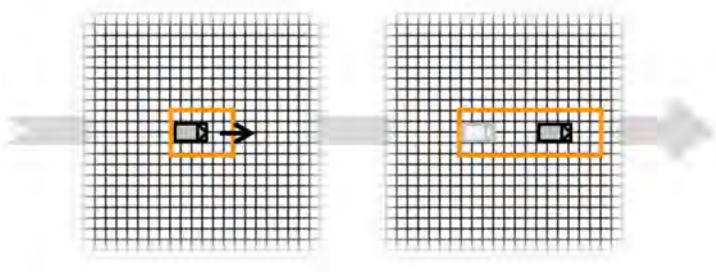

Abbildung 4.12: Anpassung der logischen Größe des GrowGrid (orange).

Noch nicht besprochen wurde, wie man das zu verwendende Messwertgrid und seine Bestandteile festlegt. Im Rahmen dieser Definition hat man eine Reihe von Wahlmöglich-

[15] Das Messwertgrid kann maximal die Größe des Hauptgrids besitzen, da es zeitlich in das Hauptgrid integriert wird.

keiten, die im Folgenden beschrieben werden; zusätzlich müssen jeweils die entsprechenden Typedefs im Header GridDefinitions angepasst werden.

Zunächst wählt man einen Evidenztyp für den Belegtheitswert und davon unabhängig auch einen für den Dynamikwert. Die OG-Bibliothek trifft keine Annahmen über die Art der Evidenzen, auch primitive Datentypen sind ohne Einschränkung möglich (z.B. Dempster-Shafer-Masse, double, ...).

Anschließend wird für den Belegtheits- und Dynamikwert jeweils ein Container gewählt, wieder voneinander unabhängig. Als Container kann eine beliebige Klasse verwendet werden, die das Interface ICellDataFusionContainer mit dem Datentyp der bereits gewählten Evidenz als Template-Parameter implementiert. Mit der Klasse CellDataFusionContainer steht eine fertige Implementierung zur Verwendung bereit, die man Dank entsprechendem Template-Argument mit einem beliebigen Evidenztyp verwenden kann.

Die Implementierung des Interfaces IMeasurementCell führt schließlich zu einer einzelnen Messwertzelle. Bei der Implementierung des Interfaces übergibt man die gewählten Belegtheits- und Dynamik-Container als Template-Argumente[16]. Mit der Klasse MeasurementCell steht eine Implementierung zur Verfügung, die mit zwei beliebigen, als Template-Parameter übergebenen ICellDataFusionContainer-Containern funktioniert (jeweils einer für den Belegtheits- und Dynamikwert).

Hat man eine einzelne Messwertzelle auf diese Weise definiert, kann das Messwertgrid gewählt werden, das das Interface IMeasurementGrid implementiert. Dabei wird die eben definierte IMeasurementCell-Zelle als Template-Parameter übergeben, zusammen mit den sechs zu verwendenden Iterator-Implementierungen. Für letztere stehen mit den Klassen BaseBlockIterator, BaseGeometricIterator und BaseSequentialIterator verwendbare Implementierungen bereit (aus einer Klasse kann jeweils die normale und die *const*-Version erzeugt werden, vgl. Abschnitt 4.2.3.2). Mit der Klasse GrowGrid existiert zudem eine Implementierung für das Interface IMeasurementGrid selbst. Mittels Template-Parametern können unterschiedliche IMeasurementCell-Zellen und Iteratoren verwendet werden.

Als Beispiel einer Messwertgrid-Definiton kann die exemplarische Implementierung unseres Anwendungsfalls dienen, die als Messwertgrid die Klasse GrowGrid nutzt. Dieses verwendet die sechs Iteratoren, die sich aus den Klassen BaseBlockIterator, BaseGeometricIterator und BaseSequentialIterator ergeben (jeweils normale und *const*-Version). Als Messwertzelle kommt die Klasse MeasurementCell zum Einsatz, die als Container sowohl für den Belegtheits- als auch für den Dynamikwert die Klasse CellDataFusionContainer enthält. Den Belegtheitswert bilden die Dempster-Shafer-Massen für *frei/belegt/unbekannt*, den Dynamikwert die Dempster-Shafer-Massen für *statisch/dynamisch/unbekannt*.

4.2.5.2 Hauptgrid

Die Implementierung des Hauptgrids wird durch das Interface IMainGrid gekapselt und ist dadurch leicht austauschbar. Wohingegen das eben beschriebene Messwertgrid der Durchführung der Kartierung dient und deshalb auf den mit Sensorik abtastbaren Bereich im unmittelbaren Fahrzeugumfeld beschränkt ist, dient das Hauptgrid der zeitlichen Integration der Messwerte in einem größeren Bereich um das Fahrzeug. Dennoch sind sich die beiden

[16] Über die gewählten Container werden implizit auch die von der Zelle genutzten Evidenzen spezifiziert.

Grids in vielerlei Hinsicht ähnlich: Das Hauptgrid besitzt ebenfalls eine feste Größe, die bei seiner Instanziierung durch den OGBuilder festgelegt wird, und verwendet ebenfalls eine erdfeste Darstellung im Gridkoordinatensystem (vgl. Abschnitt 4.2.2.2), wodurch sich das Fahrzeug auch hier über das Grid bewegt. Wiederum besitzt das Grid eine bestimmte Position im Gridkoordinatensystem und wiederum teilt der OGBuilder dem Grid die neue Fahrzeugpose jeweils mit, so dass die Fahrzeugposition im Grid bekannt ist. Allerdings verwendet das Hauptgrid einen anderen Ansatz zur Anpassung des von ihm repräsentierten Bereichs.

Nimmt man das Fahrzeug anfangs im Grid-Zentrum an, so nähert es sich durch seine Bewegung im Allgemeinen nach einiger Zeit einem Rand des Grids. Nun muss das Hauptgrid reagieren, damit sich das Fahrzeug nicht aus dem Grid hinaus bewegt. Das Messwertgrid geht mit diesem Umstand um, indem es sich schrittweise mit dem Fahrzeug mitbewegt: in Abständen verwirft es alle Daten und beginnt dann an einer neuen Position von vorne. Beim Hauptgrid ist eine derartige Neuinitialisierung von Zeit zu Zeit nicht möglich – denn seine Aufgabe ist es, die Messwerte zeitlich zu integrieren. Stattdessen müssen zur Erschließung eines neuen Gridbereichs in Fahrtrichtung neue Zellen hinzukommen, die – da die Gridgröße konstant ist – ältere Zellen ersetzen. Die genaue Realisierung ist abhängig von der gewählten Hauptgrid-Implementierung.

Die OG-Bibliothek bietet mit der Klasse RingGrid eine Implementierung des Interfaces IMainGrid. Mittels Template-Parametern können unterschiedliche IMainCell-Zellen und Iteratoren verwendet werden.

Das RingGrid erzeugt bei seiner Instanziierung entsprechend seiner Größe alle Zellen und verbindet diese zu seiner Gitterstruktur, indem es für jede Zelle die Vierernachbarschaften definiert. Anders als beim GrowGrid (vgl. vorheriger Abschnitt) existieren auch an den Rändern je Zelle vier Nachbarn, da die dortigen Zellen direkt mit den Zellen am gegenüberliegenden Ende verbunden werden. Es entsteht dadurch die Möglichkeit, die Zellen ringförmig zu durchlaufen, weshalb dieser Klasse der Name RingGrid gegeben wurde.

Der Grund für die ringförmige Struktur liegt in oben geschilderter Notwendigkeit, in Fahrtrichtung neue Zellen hinzuzufügen und sie am anderen Ende gleichzeitig zu entfernen. In [54][35] werden entsprechend neue Zellen instanziiert und alte entfernt, wodurch in diesem Bereich zusätzlich die Nachbarschaftsbeziehungen neu definiert werden müssen. In [52] wird ein komplett neues Grid erstellt, sobald das Fahrzeug einem Rand zu nahe kommt; Werte im Überlappungsbereich werden aus dem alten Grid in das neue kopiert. Beide Vorgehensweisen sind jedoch ineffizient und aufwendig.

Das RingGrid kann durch seine Ringstruktur stattdessen gedreht werden, so dass in Fahrtrichtung neue Zellen erscheinen, während sie in Gegenrichtung verschwinden – man kann sich das illustrativ als eine Art Laufband vorstellen. Dadurch ändert sich die Position des repräsentierten Gridausschnitts. Die Zellen im neu erschlossenen Gridbereich werden zurückgesetzt, so dass sie keine Informationen ihrer vorherigen Position mehr enthalten.

Um den damit verbundenen Aufwand zu reduzieren, wird das Grid logisch in 3×3 gleich große und quadratische Blöcke aufgeteilt, wie in Abbildung 4.13 gezeigt, wodurch eine Architektur entsteht, die der in [16][17] beschriebenen ähnelt. Das Fahrzeug wird immer im mittleren Block gehalten, in dem es sich auch initial befindet. Verlässt das Fahrzeug diesen Block durch seine Bewegung, so wird das Grid so gedreht, dass das Fahrzeug sich wieder im mittleren Block befindet.

Die Neuinitialisierung der Zellen und die Anpassung der Gridposition ist dadurch nicht

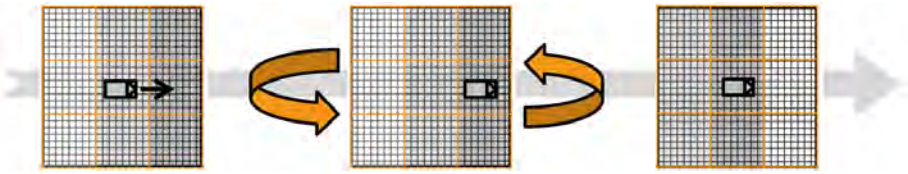

Abbildung 4.13: Anpassung des vom RingGrid repräsentierten Ausschnitts.

bei jeder Fahrzeugbewegung notwendig, sondern nur, wenn das Fahrzeug den mittleren Block verlässt. Dies führt zu einer deutlichen Reduzierung des Aufwands; außerdem steht auf diese Weise zu jedem Zeitpunkt mindestens ein vollständiger Block hinter dem Fahrzeug als Historie zur Verfügung, was in manchen Anwendungen den Einsatz robusterer Verfahren ermöglichen kann (z.B. kann der Straßenverlauf vor dem Fahrzeug zuverlässiger geschätzt werden, wenn man den hinter dem Fahrzeug miteinbezieht [33]).

Die Festlegung der Implementierungsbestandteile, die als und im Hauptgrid verwendet werden sollen, erfolgt analog[17] zur im vorherigen Abschnitt beschriebenen Vorgehensweise beim Messwertgrid. Als Beispiel für eine solche Hauptgrid-Definition kann die exemplarische Implementierung unseres Anwendungsfalls dienen, die als Hauptgrid die Klasse RingGrid nutzt. Dieses verwendet die sechs Iteratoren, die sich aus den Klassen BaseBlockIterator, BaseGeometricIterator und BaseSequentialIterator ergeben (jeweils normale und *const*-Version). Als Hauptzelle kommt die Klasse MainCell zum Einsatz, die als Container sowohl für den Belegtheits- als auch für den Dynamikwert die Klasse CellDataFusionContainer enthält. Den Belegtheitswert bilden die Dempster-Shafer-Massen für *frei/belegt/unbekannt*, den Dynamikwert die Dempster-Shafer-Massen für *statisch/dynamisch/unbekannt*.

4.2.5.3 Occupancy Grid

Nun wurde alles besprochen bis auf den letzten Schritt, in dem aus dem Hauptgrid das Occupancy Grid erzeugt wird – an dem das Programm, das die OB-Bibliothek verwendet, interessiert ist. Dabei ist es die Aufgabe des Occupancy Grids, den aktuellen Inhalt des Hauptgrids wie gewünscht in einer zur Ergebnisrückgabe geeigneten Datenstruktur darzustellen. Dazu gibt es einige Möglichkeiten, die mithilfe des Interfaces IOccupancyGrid gekapselt werden.

Betrachten wir zuerst die Datenstruktur. Das Hauptgrid verwendet eine Gitterstruktur bestehend aus miteinander verknüpften Zellen, wobei jede Zelle einen komplexen Aufbau besitzt. Diese Datenstruktur eignet sich offensichtlich nicht dazu, von der OG-Bibliothek als Ergebnis an das externe Verwender-Programm zurückgegeben zu werden. Stattdessen wird deshalb ein Vektor bestehend aus einzelnen Bytes zur Rückgabe verwendet. Dabei enthält der Vektor sequentiell die Werte aller Zellen des Occupancy Grids, von der Ecke links oben bis zur Ecke rechts unten, wobei der Wert jeder Occupancy-Grid-Zelle durch ein einzelnes Byte dargestellt wird; solch eine Darstellung ist deutlich einfacher vom externen Programm verwendbar.

Für den Zellwert des Occupancy Grids sind unterschiedliche Möglichkeiten denkbar und

[17] Die Interfaces einer einzelnen Hauptzelle und des Hauptgrids lauten IMainCell bzw. IMainGrid, ihre Beispiel-Implementierungen sind die Klassen MainCell bzw. RingGrid.

von der Grid-Implementierung wählbar. Zu berücksichtigen ist dabei, dass jede Zelle des Hauptgrids nicht nur einen Belegtheitswert, sondern auch noch einen Dynamikwert besitzt (vgl. Abschnitt 4.2.3.1). Prinzipiell hat man zwei Möglichkeiten, wie man den Dynamikwert einer Hauptzelle zur Bestimmung des zugehörigen Occupancy-Grid-Zellwerts verwenden kann: Man kann ihn zur Filterung oder zur Visualisierung dynamischer Bereiche nutzen.

Gehen wir zunächst davon aus, dass das Occupancy Grid die übliche Darstellung besitzen soll, das heißt, dass in jeder Zelle die Evidenz dafür enthalten ist, dass dieser Bereich des Umfelds durch ein statisches Hindernis belegt ist. In diesem Fall kann der Dynamikwert zur Dynamikfilterung eingesetzt werden. Eine oft geeignete Möglichkeit dazu: Man verwendet den Belegtheitswert der Hauptzelle unverändert als Wert der Occupancy-Grid-Zelle, wenn über die Dynamik keine Informationen vorliegen oder die Informationen auf eine statische Region hindeuten. Besitzt die Hauptzelle jedoch einen hohen Dynamikwert, so wird die Zelle im Occupancy Grid – unabhängig vom eigentlichen Belegtheitswert im Hauptgrid – als *frei* eingetragen, da es nur statische Objekte enthalten soll. Die dahinterstehende Idee ist, dass ein Bereich der Karte, in dem sich dynamische Objekte bewegen, nie durch statische Objekte belegt sein kann. Unabhängig davon, ob dieser Bereich aktuell wirklich durch ein dynamisches Objekt belegt ist oder nicht, ist er deshalb als *frei* zu kartieren.

In erster Linie dient der Dynamikwert zur internen Verwaltung und – beispielsweise wie eben demonstriert – zur Filterung dynamischer Objekte. Bei Bedarf kann er jedoch auch visualisiert werden[18]. In manchen Fällen kann sich eine Szene bei visualisierter Dynamik deutlich klarer darstellen, da auf diese Weise dynamische Objekte und statische Freiflächen unterschieden werden können – die sonst beide als *frei* kartiert würden. Außerdem befinden sich oftmals Verschattungen im Grid, deren Ursache nicht erkennbar ist, da keine Hindernisse vor diesen Verschattungen kartiert wurden. Die Visualisierung dynamischer Bereiche zeigt explizit die dafür ursächlichen dynamischen Objekte und kann deshalb lohnend sein.

In Abbildung 4.14 sind exemplarisch vier Darstellungsmöglichkeiten derselben Szene gezeigt. (1) zeigt dabei die übliche Darstellung, in der nur statische Hindernisse kartiert sind; der Dynamikwert wurde entsprechend zur Dynamikfilterung benutzt. In (2) wird er dagegen blau visualisiert. Der Unterschied ist gut an dem vorausfahrenden Fahrzeug zu erkennen, das als dynamisches Objekt in (1) weiß und in (2) blau dargestellt wird; der Grund für die Verschattung vor dem eigenen Fahrzeug ist im zweiten Fall dadurch offensichtlicher.

Üblicherweise visualisiert ein Occupancy Grid die Unsicherheiten der Belegungen durch eine Graustufen- und gegebenenfalls Blaustufenskala, mit derer Hilfe die Höhe der Evidenzen abgestuft dargestellt werden können – wie beispielsweise in (1) und (2). Während für die meisten Anwendungen eine solche informationsreichere Repräsentation geeigneter scheint, ist es für manche Anwendung vorteilhaft, die Evidenzen in diskrete Werte umzuwandeln. Dadurch entsteht zum Beispiel ein Occupancy Grid, das nur die drei Zustände *belegt*, *frei* und *unbekannt* enthält – wie das in Abbildung 4.14 (3). (4) besitzt darüber hinaus einen Wert für *dynamisch*. Die kontinuierlichen Evidenzen in den Hauptzellen können zur Diskretisierung zum Beispiel mit der Maximum-a-Posteriori-Entscheidungsregel oder durch die Anwendung eines einfachen oder doppelten Schwellwerts in den diskreten Wertebereich der Zellen des Occupancy Grids überführt werden [19].

Wie an dieser Stelle deutlich wird, arbeitet das Occupancy Grid direkt mit den Evidenzen der Hauptzellen, das heißt, es muss auf deren Werte zugreifen, und benötigt deshalb

[18] Im Falle der Dynamikvisualisierung handelt es sich streng genommen nicht mehr um Occupancy Grids, sondern um eine Erweiterung davon.

Abbildung 4.14: Vier unterschiedliche Darstellungsformen derselben Situation.

eine evidenzspezifische Implementierung.

Schließlich unterscheiden sich die Implementierungen noch darin, wie das Occupancy Grid aus dem Hauptgrid erstellt wird. Die Erstellung ist aufgrund der in der Regel hohen Zellanzahl aufwendig. Deshalb versucht eine typische Implementierung, das bereits vorhandene Occupancy Grid zu aktualisieren, anstatt ein neues zu erstellen. Wie eine solche Aktualisierung konkret aussieht und unter welchen Rahmenbedingungen sie möglich ist, ist genauso von einer konkreten Implementierung abhängig wie die Art und Weise, auf die ein Occupancy Grid erzeugt wird, falls noch gar keines zur Aktualisierung vorliegt.

Zusammengefasst können sich Occupancy-Grid-Implementierungen in vielerlei Hinsicht unterscheiden, weshalb sie mithilfe des Interfaces IOccupancyGrid gekapselt werden. Dadurch ist ein leichter Austausch der Implementierung möglich; die zu verwendende Implementierung wird über die entsprechende Typedef im Header GridDefinitions festgelegt.

Mit den Klassen DetailedDSOccupancyGrid und MaxValueDSOccupancyGrid stehen zwei unterschiedliche Implementierung des Interfaces IOccupancyGrid zur Verfügung, die alternativ zueinander verwendet werden können.

Die Klasse DetailedDSOccupancyGrid berücksichtigt Unsicherheiten in der Darstellung wie (1, 2) in Abbildung 4.14, die Klasse MaxValueDSOccupancyGrid beseitigt sie dagegen durch eine Diskretisierung auf drei (3) bzw. vier (4) Werte. Beide Klassen unterstützen zudem drei Alternativen zum Umgang mit dem Dynamikwert[19]: Sie können ihn wie (1, 3) zur Filterung dynamischer Bereiche nutzen, ihn wie (2, 4) zur Visualisierung dynamischer Bereiche verwenden oder ihn vollständig ignorieren.

Darüber hinaus führen beide die Aktualisierung des Occupancy Grids auf dieselbe Weise durch. Das Occupancy Grid wird effizient aktualisiert, wenn sich das zu erstellende Occupancy Grid wieder an derselben Position wie das vorherige befindet: die Aktualisierung beschränkt sich auf den Teil, indem sich das Hauptgrid laut OGBuilder verändert hat (vgl. Abschnitt 4.2.2.1). Hat sich die Position geändert[20], wird das gesamte Occupancy Grid aktualisiert, was in etwa einer Neuerstellung gleich kommt.

[19] Wie mit der Dynamik umgegangen werden soll, lässt sich leicht durch Ein- und Auskommentieren der entsprechenden „Defines" im Quellcode festlegen.

[20] Im Falle des RingGrid als Hauptgrid (vgl. Abschnitt 4.2.5.2) ist das der Fall, wenn eine Rotation der Blöcke erfolgte.

4.3 Verwendung in ADTF

Die OG-Bibliothek wird in dieser Arbeit beispielhaft zusammen mit dem Framework ADTF verwendet. Um dies zu ermöglichen, wurde mit dem OGFilter eine eigene ADTF-Komponente entwickelt, die im Folgenden vorgestellt werden soll. Einführend wird dazu zunächst ein kurzer Überblick über ADTF gegeben.

4.3.1 ADTF im Überblick

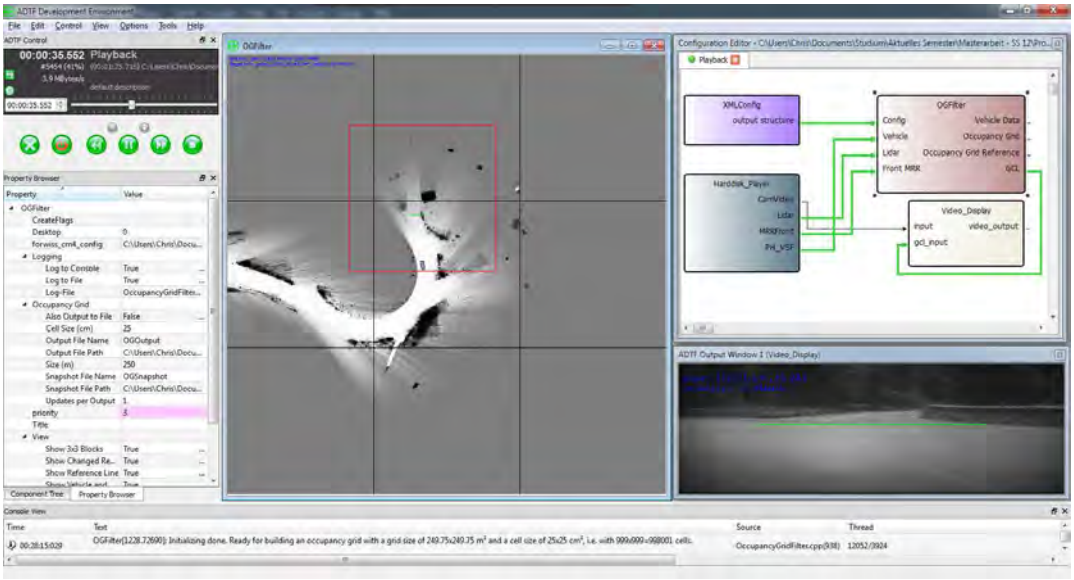

Abbildung 4.15: Screenshot von ADTF während der Erstellung eines Occupancy Grids.

Das *EB Assist **A**utomotive **D**ata and **T**ime-Triggered **F**ramework*, kurz *ADTF*, wurde ursprünglich ab 2001 von Audi entwickelt. Seit 2008 wird es durch ElektroBit auch an andere Automobilhersteller und -zulieferer vertrieben. Auch in Projekten von FORWISS kommt es zum Einsatz.

ADTF bietet eine Infrastruktur zur leichteren Entwicklung neuer Funktionalitäten für Fahrerassistenzsysteme unter den Betriebssystemen Windows und Linux. Es erlaubt sowohl das Aufzeichnen von Daten, zum Beispiel im Rahmen von Messfahrten, als auch die spätere Wiedergabe und Visualisierung der aufgezeichneten Daten in Echtzeit im Rahmen einer Simulation. Dazu stehen bereits einige Komponenten zur Verwendung bereit.

Die Erstellung und Integration eigener ADTF-Komponenten ist über die veröffentlichten Schnittstellen und Datenformate möglich. Dabei wird man durch eine C++-Entwicklungsumgebung unterstützt, deren Klassenbibliothek viele, weitgehend betriebssystemunabhängige Funktionalitäten bereitstellt. Zudem besteht für graphische Aufgaben Unterstützung für Qt und OpenGL. Ferner stehen eine Dokumentation und Beispiele zur Verfügung.

Die Simulation findet im ADTF-Laufzeit-System statt. Ein wichtiger Bestandteil davon ist der Konfigurationseditor, der über eine graphische Benutzeroberfläche die Erstellung von ADTF-Anwendungen erlaubt. Eine ADTF-Anwendung besteht aus mehreren *Filtern*, die zu einem *Filtergraphen* verbunden werden. Jeder Filter stellt eine gewisse Funktionalität bereit. Der Filtergraph verbindet die Ein- und Ausgabepins der Filter und definiert damit

den Informationsfluss zwischen ihnen. Die übertragenen Informationen sind so genannte *MediaSamples*, die beliebige Daten enthalten können.

4.3.2 Der OGFilter

Mit dem OGFilter wurde ein eigener ADTF-Filter entwickelt, der anderen ADTF-Komponenten die Erstellung von Occupancy Grids ermöglicht. Zur Bereitstellung dieser Funktionalität verwendet er die OG-Bibliothek (vgl. Abschnitt 4.2).

Die Implementierung der Klasse OGFilter (inklusive der genutzten OG-Bibliothek) ist an unser exemplarisches Anwendungsszenario angepasst, lässt sich aber leicht – durch kleine, offensichtliche Änderungen – auch in anderen ähnlichen Situationen einsetzen.

4.3.2.1 Einbindung in eine ADTF-Anwendung

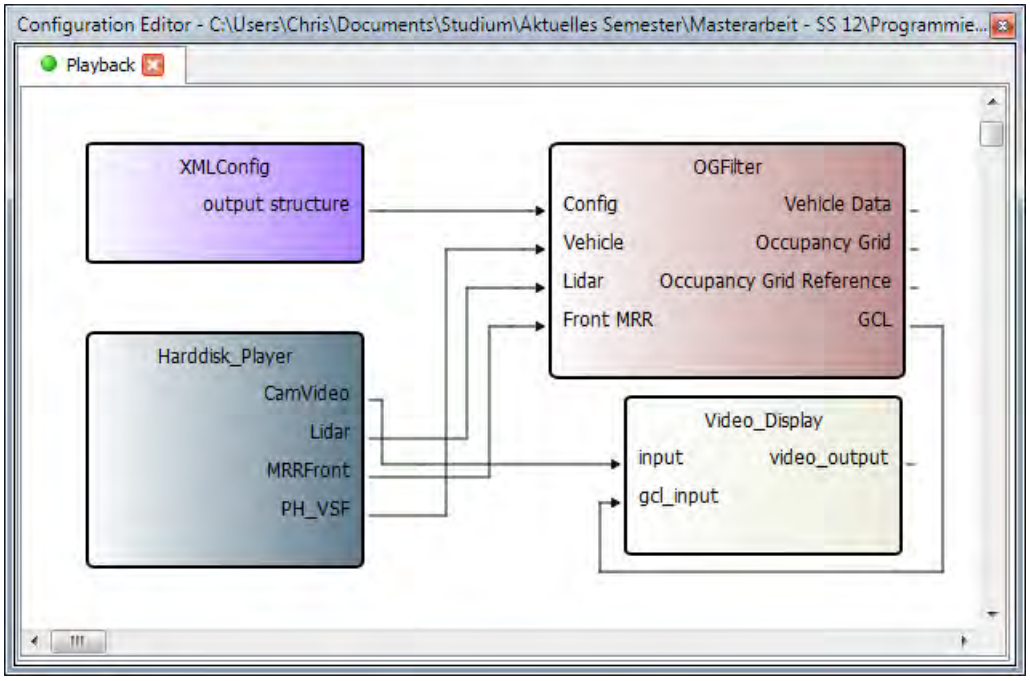

Abbildung 4.16: Mögliche Konfiguration zur Erstellung eines Occupancy Grids.

Um den OGFilter zur Erstellung von Occupancy Grids in ADTF zu verwenden, integriert man ihn in eine ADTF-Anwendung, indem man die Ein- und Ausgabepins mit denen anderer ADTF-Komponenten verbindet. Während die Verwendung der Ausgabepins optional ist, sind die Eingabepins notwendig, da der OGFilter darüber die zu verwendenden Sensor- und Konfigurationsdaten erhält.

Abbildung 4.16 zeigt ein Beispiel für eine einfache ADTF-Anwendung, die nichtsdestotrotz ausreichend ist zur Erstellung von Occupancy Grids und auch im Rahmen unseres exemplarischen Anwendungsfalls dazu benutzt werden kann. Der OGFilter besitzt vier Eingabe- und vier Ausgabepins. Über den ersten Eingabepin erhält er die Fahrzeug- und Sensorkonfigurationsdaten, in unserem Beispiel vom Filter *XMLConfig*, der diese aus XML-Dateien liest. Über die restlichen drei Eingabepins empfängt der OGFilter die aktuellen Eigenzustands-, Lidar- bzw. Radardaten; hier übermittelt vom *Harddisk-Player*, der sie in

Echtzeit aus einer Datei liest. Dieser ist ein Beispiel für eine mitgelieferte, fertige ADTF-Komponente; der XMLConfig-Filter wurde uns dagegen maßgeschneidert zur Verfügung gestellt.

Über die Ausgabepins des OGFilter erhalten andere ADTF-Komponenten das aktuell erstellte Occupancy Grid (als Kopie oder zwecks Performanz als Referenz) und die aktuelle Eigenzustandsschätzung. Ferner wird eine Videoüberlagerung ausgegeben (beschriftet mit *GCL*, vgl. nächster Abschnitt). Diese können zum Beispiel wie hier zusammen mit dem wieder vom Harddisk-Player bereitgestellten Referenzvideo als Eingabe für das *Video-Display* dienen, einer weiteren mitgelieferten ADTF-Komponente, die das Video inklusive der Überlagerung anzeigt.

4.3.2.2 Graphische Benutzeroberfläche

Werfen wir nun der Reihe nach einen Blick auf die drei Teile der graphischen Benutzeroberfläche, die direkt mit dem OGFilter zu tun haben.

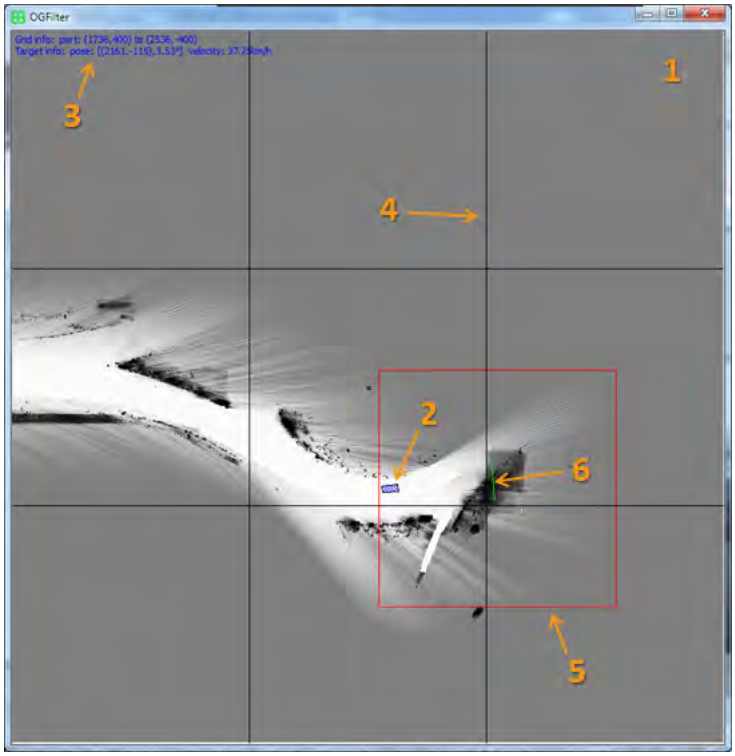

Abbildung 4.17: Die Visualisierungen des OGFilter in ADTF.

Betrachten wir zunächst den wichtigsten Teil, das in Abbildung 4.17 gezeigte, OGFilter-eigene Ausgabefenster. Es stellt das aktuelle Occupancy Grid (1) und das eigene Fahrzeug (2) in Originalgröße dar. Der Benutzer kann die Darstellungsgröße der Anzeige anpassen (+- und −-Tasten). Passt das Grid nicht vollständig in das Fenster, kann er mittels Drag-&-Drop in ihm navigieren. Außerdem kann er zu jedem Zeitpunkt eine Momentaufnahme davon speichern (*S*-Taste), Name und Pfad werden über das Eigenschaftsfenster spezifiziert (siehe unten). Dort können zudem weitere Visualisierungen ein- und ausgeblendet werden:

– Ein blaues *Head-up-Display* (HUD), das die aktuelle Fahrzeugpose und -geschwindigkeit sowie den aktuell vom Grid repräsentierten Ausschnitt (gegeben durch die Positionen zweier gegenüberliegender Eckpunkte) angibt (3).

– Schwarze Linien, die das Grid in 3×3 Blöcke zerteilen (4).

– Ein roter Rahmen um den Bereich, in dem Änderungen gegenüber dem zuletzt angezeigten Occupancy Grid berücksichtigt wurden (5).

– Eine grüne Linie, die auch im Video angezeigt wird und dadurch als Referenz verwendet werden kann (6).

Wie im vorherigen Abschnitt beschrieben, stellt der OGFilter zudem über einen seiner Ausgabepins eine Videoüberlagerung bereit. Bei dieser Ausgabe handelt es sich um Befehle in der so genannten *Graphics Command Language* (GCL). Damit kann in anderen ADTF-Komponenten, die GCL-Befehle über einen Eingabepin akzeptieren, eine Visualisierung erzeugt werden. Der OGFilter erzeugt damit eine graphische Videoüberlagerung, die mit einer entsprechenden ADTF-Komponente (z.B. das Video-Display, vgl. vorheriger Abschnitt) visualisiert werden kann. Die Ausgabe umfasst wieder ein blaues Head-up-Display und eine grüne Referenzlinie, die beide identisch zu den eben gerade vorgestellten Visualisierungen sind, dieses mal aber als Überlagerung in einem Video angezeigt werden (vgl. Abbildung 4.18).

Abbildung 4.18: Die Visualisierungen im Video-Display von ADTF.

Als dritter Teil der graphischen Benutzeroberfläche existiert mit dem *Property Browser* ein Eigenschaftsfenster, in dem der Benutzer etliche Einstellungen für den OGFilter und das zu erstellende Occupancy Grid vornehmen kann. Er ist in Abbildung 4.19 gezeigt. Die Änderungen können ständig vorgenommen werden, bei laufender Simulation werden sie allerdings erst nach deren Stopp berücksichtigt.

Unter dem übergeordneten Punkt *Occupancy Grid* befinden sich alle Einstellungen, die das Occupancy Grid selbst betreffen; insbesondere kann hier die Größe einer einzelnen Zelle in Zentimeter und die Größe des gesamten Grids in Meter festgelegt werden. Erfüllen die Benutzereingaben für diese beiden Gridparameter nicht die Vorgaben der OG-Bibliothek, werden sie automatisch entsprechend angepasst (vgl. nächster Abschnitt). Außerdem kann festgelegt werden, ob jedes neue Occupancy Grid zusätzlich zur Anzeige in einer Datei gespeichert werden soll. Für diese regelmäßige Dateiausgabe kann jeweils Pfad und Name festgelegt werden (der Name wird um eine fortlaufende Nummer ergänzt); gleiches gilt für die manuell veranlassten Grid-Momentaufnahmen (siehe oben). Außerdem kann gewählt

werden, nach wie vielen durchgeführten Gridfusionen (vgl. Abschnitt 4.2.5.3) eine neue Ausgabe erfolgen soll – dies betrifft die Bildschirmanzeige, die Ausgabepins als auch die Dateiausgabe.

Die Einstellungen unter *View* ermöglichen die Festlegung, welche der vier optionalen Visualisierungen angezeigt werden sollen (siehe oben).

Der OGFilter kann Informationen und Fehler protokollieren. Die *Logging*-Eigenschaften legen dabei fest, ob in die Konsole und/oder eine Textdatei protokolliert werden soll, in letzterem Fall gegebenenfalls auch in welche.

Die Eigenschaft *forwiss_cm4_config* legt den Pfad zu einer *cm4*-Datei fest, die für die optionale Anzeige einer grünen Linie als Videoüberlagerung (siehe oben) benötigt wird.

Bei den restlichen in der Abbildung gezeigten Eigenschaften, die hier nicht beschrieben wurden, handelt es sich um allgemeine Eigenschaften, die nicht speziell den OGFilter betreffen.

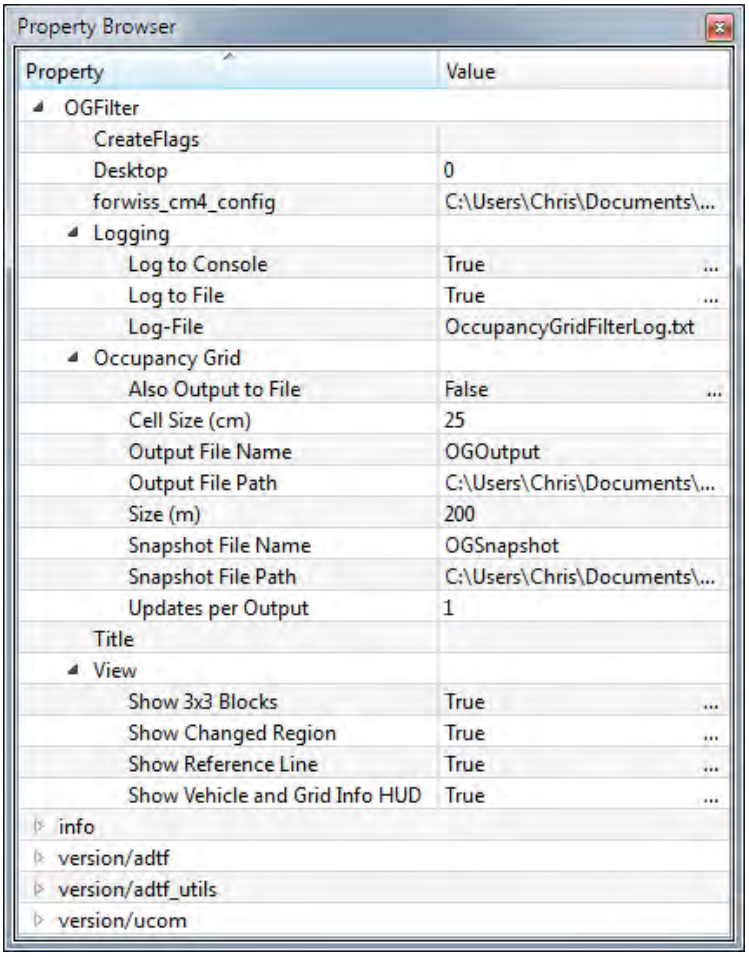

Abbildung 4.19: Eigenschaftsfenster des OGFilter.

4.3.2.3 Ablauf

Wurde der OGFilter in die ADTF-Anwendung eingebunden, werden die Größe einer einzelnen Zelle und die Größe des gesamten Grids über die graphische Benutzeroberfläche

festgelegt. Die OG-Bibliothek setzt dabei voraus, dass die Zellgröße nicht über der Gridgröße liegt und dass die Gridgröße ein ganzzahliges Vielfaches von drei und ungerade ist. Der OGFilter passt die Benutzereingaben automatisch an, wenn sie diese Vorgaben nicht erfüllen, und protokolliert die verwendeten Gridparameter in der *Console View* von ADTF (vgl. Abbildung 4.15 unten).

Nachdem der OGFilter in die ADTF-Anwendung eingebunden und die gewünschten Einstellungen vorgenommen wurden, kann die Simulation gestartet werden. Sobald der OGFilter die anwendungsspezifischen Konfigurationsdaten über den entsprechenden Eingabepin erhalten hat, erstellt er damit eine Instanz des OGBuilder als Schnittstelle zum Zugriff auf die Funktionalität der OG-Bibliothek. Diese Instanziierung des OGBuilder führt zur Erstellung von Haupt- und Messwertgrid, wodurch es für größere, hoch aufgelöste Grids zu einer bemerkbaren Verzögerung kommen kann. Um diese Wartezeit bei nachfolgenden Simulationsausführungen zu vermeiden, erfolgt eine Neuinstanziierung des OGBuilder und damit eine Neuerstellung der Grids nur, wenn sich die dann empfangenen Konfigurationsdaten oder die in der Benutzeroberfläche eingetragenen Gridparameter geändert haben. Ansonsten werden beide lediglich in den Ursprungszustand zurückgesetzt, anstatt sie neu zu erstellen.

Mit der Erstellung des OGBuilder ist der OGFilter bereit zum Empfang von Eigenzustands- und Umfelddaten über die entsprechenden Eingabepins. Der Datenempfang erfolgt – wie im Übrigen auch die graphische Ausgabe – nebenläufig und der OGFilter ist entsprechend darauf ausgelegt. Während bereits empfangene Daten verarbeitet werden, können weiterhin neue Daten empfangen werden. Da die OG-Bibliothek dagegen keine nebenläufige Verwendung unterstützt, stellt der OGFilter einen entsprechend synchronisierten Zugriff darauf sicher.

Bis zum Ende der Simulation empfängt der OGFilter nun fortlaufend und asynchron von anderen ADTF-Komponenten die Eigenzustands- und Umfelddaten; letztere werden nach Erhalt zwischengespeichert. Sobald Eigenzustandsdaten empfangen wurden, werden diese und etwaig zwischengespeicherte Umfelddaten dem OGBuilder übergeben – dadurch erfolgt die Integration der Sensordaten in die OG-Bibliothek synchron mit den Eigenzustandsdaten, wodurch die zu diesem Zeitpunkt aktuelle Position berechnet werden kann.

Die OG-Bibliothek verarbeitet alle Daten und meldet anschließend, ob die neuen Daten bereits zu einer Gridfusion und damit zu im Occupancy Grid sichtbaren Änderungen geführt haben (vgl. Abschnitt 4.2.2.1).

Wurde eine Gridfusion durchgeführt und dadurch seit der letzten Ausgabe die vom Benutzer festgelegte Anzahl an Gridfusionen erreicht (vgl. vorheriger Abschnitt), fordert der OGFilter das aktuelle Occupancy Grid und die aktuelle Eigenzustandsschätzung beim OGBuilder an und aktualisiert entsprechend seine Ausgabe. Diese umfasst die graphische Ausgabe, Ausgaben an den Ausgabepins und optional die Ausgabe in eine Datei, wie sie in den vorherigen beiden Abschnitten bereits beschrieben wurden.

Kapitel 5

Praktische Ergebnisse

Im vorherigen Kapitel wurde die OG-Bibliothek vorgestellt, die zur Erstellung von Occupancy Grids verwendet werden kann. In diesem Kapitel werden die Herausforderungen beschrieben, die bei der Umsetzung der OG-Bibliothek auftraten, Möglichkeiten zur Evaluierung des vorgestellten Ansatzes besprochen und konkrete Ergebnisse präsentiert, indem beispielhafte Szenen unseres konkreten Anwendungsfalls besprochen werden.

5.1 Herausforderungen bei der Umsetzung

Bei der praktischen Umsetzung des Implementierungsziels (vgl. Abschnitt 4.1.2) ergaben sich eine Reihe von Herausforderungen.

Eine davon war die hohe Kartendimensionalität. Diese machte es notwendig, die Berechnungen und den Speicherbedarf je Zelle möglichst gering zu halten. Anfänglich bestanden Probleme, da die Gridstrukturen zu viel Speicherplatz benötigten. Bei den ersten Testläufen kam es sogar zu Abstürzen des Testrechners, da dem Betriebssystem der Speicher ausging. Später kam insbesondere der Laufzeit und damit Berechnungskomplexität Bedeutung zu – lange Zeit lag die Ausführungszeit um einen Faktor von 5 und mehr über der eigentlichen Simulationsdauer. Vor allem der Komplexität der inversen Sensormodelle war hier Aufmerksamkeit zu schenken, da diese einen relativ hohen Rechenaufwand erfordern und gleichzeitig sehr oft ausgeführt werden. Aber auch die Wahl der Dempster-Shafer-Theorie bedeutet einen wesentlich höheren Aufwand gegenüber dem klassischen Ansatz.

Bezüglich der für den exemplarischen Anwendungsfall bereitgestellten Daten (vgl. Abschnitt 4.1.1) ergaben sich gleich eine Reihe von Herausforderungen.

Für die Sensoren, von denen die Daten stammen, standen keine Datenblätter zur Verfügung, sondern nur unvollständige und teilweise widersprüchliche Kommentare im Quellcode eines Projekts, das diese Daten ebenfalls nutzte. Völlig unklar blieb zum Beispiel die – offensichtlich sehr entscheidende – Position des Messwertkoordinatensystems des Lasersensors. In Ermangelung besserer Alternativen erfolgte die Ermittlung dessen (ungefährer) Lage über ein manuelles, visuelles Abgleichen der im Videobild erkennbaren Hindernisse mit den dort zu diesem Zweck ebenfalls visualisierten Messwerten. Außerdem führt der Lasersensor eine interne Klassifizierung der Objekte durch, allerdings war die Bedeutung der einzelnen, durch Zahlen repräsentierten Klassen nicht bekannt. Auch die Drehrichtung der Winkelmessung des Radars war nirgends spezifiziert; auf den ersten Blick ein kleineres Problem, das man mutmaßlich leicht durch simples Ausprobieren in Erfahrung bringt. Allerdings war

dieses Ausprobieren erst möglich, nachdem die OG-Bibliothek bereits zu großen Teilen fertig gestellt war, da Kartierung und Visualisierung dazu bereits notwendig sind. Dies führte dazu, dass die Fehlerursache nicht mehr leicht einzugrenzen und deshalb lange nicht zu finden war.

Die Daten sind außerdem vorverarbeitet bzw. gefiltert, wobei auch hier keinerlei genaueren Informationen vorlagen; zur Verfügung stand lediglich das jeweils ausgegebene Ergebnis, nicht bekannt war allerdings der genaue Entstehungsprozess, was die Erstellung eines adäquaten inversen Sensormodells behinderte. Hinzukam, dass Occupancy Grids typischerweise auf Basis unvorverarbeiteter und ungefilterter Rohdaten erstellt werden [22]; dies erleichtert das in Abschnitt 3.1.2 beschriebene Raycasting und ermöglicht die Berücksichtigung der mit der Messung verbundenen Unsicherheiten. Die Vorverarbeitung führt dagegen typischerweise ein Tracking durch, wodurch diese Unsicherheiten verloren gehen. Bei vorverarbeiteten Daten, wie sie in unserem Fall vorliegen, reduziert sich aus diesem Grund auch der Gewinn, der sich durch die zeitliche Integration in die Karte ergibt, und – schlimmer noch – ist diese Vorverarbeitung fehlerhaft, so wird auch die Griddarstellung an dieser Stelle fehlerhaft. Denn eigentlich dient die zeitliche Integration in das Grid der Verringerung der Messunsicherheiten; wird aber ein getracktes Objekt eingetragen, wurden diese Unsicherheiten bereits während des Trackings berücksichtigt und durch dessen Modell eliminiert – das Objekt wird „träger". Befindet es sich dabei an der falschen Stelle, so wird es aufgrund dieser Trägheit längere Zeit an dieser Stelle vermutet und entsprechend oft dort in die Karte eingetragen, wohingegen im Umfeld um die getrackte Position – wo sich das falsch getrackte Objekt eigentlich irgendwo befindet – keine Kartierung stattfindet (was aber bei der Verwendung der unsicheren Rohdaten sehr wohl der Fall wäre).

Ein weiteres Problem war, dass die Daten teilweise ungenau und fehlerhaft sind, was aufgrund der eben beschriebenen Situation äußerst problematisch ist – die falschen Schätzungen gehen mehrmals ein. Die Geschwindigkeitsschätzung auf Basis der Datenvorverarbeitung des Lidars stimmt oftmals gut mit der Realität ein, in anderen Fällen liegt sie allerdings weit daneben. Beispielsweise werden für eigentlich statische Leitplanken manchmal absolute Geschwindigkeiten in der Größenordnung von 40 km/h geschätzt. Dies führt unter anderem dazu, dass die Verwendung der Geschwindigkeiten zur Dynamikerkennung – was die logische Vorgehensweise wäre – fehlschlägt, da viele eigentlich statische Objekte dadurch fälschlicherweise als dynamisch klassifiziert werden und entsprechend nicht wie eigentlich erwünscht in der Kartendarstellung enthalten sind. Zudem werden für die Messpunkte des Lidars *Bounding Boxes* berechnet, deren Orientierung und Größe oftmals sehr gut die Messpunkte beschreibt, teilweise allerdings nicht nachvollziehbar ist. Die vorverarbeiteten Radardaten enthalten teilweise falsch gesetzte Flags oder nicht wirklich die angegebene Anzahl an Hindernissen, sondern weniger.

Der Einsatz des Frameworks ADTF (vgl. Abschnitt 4.3.1) bot einige Vorteile, insbesondere konnten damit die aufgezeichneten Daten leicht zur Simulation verwendet werden. Allerdings lag genau da auch ein Nachteil: Testen mit den aufgezeichneten Daten war dadurch nur über den Umweg ADTF möglich, nicht aber in der eigentlichen Entwicklungsumgebung. Aus gleichem Grund konnte mit dem Testen erst richtig begonnen werden, als ein Großteil der Implementierung bereits fertiggestellt war und insbesondere eine sichtbare Ausgabe erzeugt werden konnte. Zu diesem Zeitpunkt gestaltete sich eine Fehlersuche bereits als äußerst schwierig, so dass für künftige Arbeiten wohl eine alternative Herangehensweise gewählt werden wird.

In fast jedem einschlägigen Paper wird die theoretische Grundlage des Occupancy Grid Mappings zu Beginn kurz eingeführt. Dagegen finden sich in fast keinem Paper hilfreiche Angaben zur praktischen Umsetzung. Bezüglich der Implementierung startete die Arbeit deshalb bei Null. Von Beginn an stellte sich so immer wieder die Frage der praktischen Umsetzung: Welche Datenstruktur wird für das Grid verwendet? Wie wird das Auto im Grid gehalten? Wie gelingt ein Ansatz, der „halbwegs schnell genug" ist? Wie kann die zur Kartierung notwendige Daten-Synchronisierung der asynchron arbeitenden Eigenzustands- und Umfeldsensorik erreicht werden? ...

Neben der prinzipiellen Frage nach dem „Wie?" stellte die Allgemeingültigkeit des Ansatzes eine zusätzliche Herausforderung dar. Anstatt einen einzelnen konkreten Lösungsansatz (z.B. Detektion dynamischer Objekte durch Radargeschwindigkeitsmessungen) im Rahmen einer konkreten Anwendung (z.B. Verwendung eines einzelnen Radar- und eines einzelnen Lasersensors zur Straßenranddetektion) umsetzen zu können, musste eine sehr abstrakte, allgemeingültige Lösung gefunden werden. Insgesamt betrachtet stellte dies sogar eine der wesentlichsten Herausforderungen dar.

Ebenfalls herausfordernd war die Überführung der Messwerte in Evidenzen. Im Falle unserer evidenztheoretischen Beispiel-Implementierung war jeder Messwert durch eine geeignete Massefunktion in Massen für die drei möglichen Aussagen *frei*, *belegt* und *unbekannt* zu überführen (vgl. Abschnitt 3.2.2). Die Wahl dieser sensorspezifischen Massefunktion bestimmt maßgeblich die Qualität des Ergebnisses und ist, anders als man zunächst vielleicht vermuten würde, keine triviale Aufgabe: Die einem Sensor zugeordnete Massefunktion muss die Massen so wählen, dass diese intern konsistent sind (z.B. müssen Hinderniskartierung und Freiraummodellierung aufeinander abgestimmt werden) und zudem die Messunsicherheit und Zuverlässigkeit des Sensors im Vergleich zu den übrigen Sensoren berücksichtigen (z.B. müssen in unserem Fall die teilweise fehlerhaften Bounding Boxes des Lidars geeignet eingebracht werden). Des Weiteren war ein Augenmerk darauf zu legen, dass die Massefunktion nicht entsprechend einer bestimmten Umgebung ausgewählt wird (z.B. für eine wenig befahrene Autobahn mit massiven beidseitigen Betonwänden), sondern derart, dass gute Ergebnisse in unterschiedlichen Szenarien erreicht werden können (z.B. auch bei starkem Verkehr in einer verwinkelten Altstadtgasse). Es musste also ein geeigneter Kompromiss gefunden werden, was nicht leicht war. Dies ist allerdings weder ein spezielles Problem der vorliegenden Implementierung, noch der verwendeten Dempster-Shafer-Theorie – die gleiche Problematik gilt es in anderen Arbeiten zu lösen, auch in solchen, die die klassische Variante verwenden [28].

Eine weitere Herausforderung war ganz persönlicher Natur: der Autor dieser Arbeit hatte zuvor noch nie ein Programm in der hierfür verwendeten Programmiersprache C++ geschrieben. Trotz der syntaktischen Ähnlichkeiten zur von ihm erlernten Sprache Java gibt es doch etliche Besonderheiten und Eigenheiten von C++, auf die erst im Verlauf der Arbeit gestoßen wurde oder die erst dann klar wurden. Aus diesem Grund existiert der in den ersten Wochen erstellte Quellcode nicht mehr in seiner ursprünglichen Form, sondern wurde im Verlauf der Arbeit – mit dem bis dahin erworbenen, neuen Wissen – von Grund auf überarbeitet. Ebenfalls Neuland waren für den Autor zudem ADTF und die Grafikbibliothek Qt, die im Rahmen des OGFilter zum Einsatz kommt.

5.2 Möglichkeiten zur Evaluierung

Dieses Kapitel möchte das Ergebnis der OG-Bibliothek möglichst unabhängig von der exemplarischen Implementierung unseres Anwendungsfalls evaluieren – es soll eine Bewertung stattfinden, die auch auf andere Anwendungsszenarien übertragbar ist. Generell bestehen zwei Möglichkeiten, um Occupancy Grids zu evaluieren: durch visuelle Inspektion oder durch den Einsatz einer Metrik.

Im Rahmen einer visuellen Inspektion wird das erzeugte Occupancy Grid evaluiert, indem es manuell mit den realen Gegebenheiten verglichen wird, welche typischerweise in Form einer Videoreferenz vorliegen; falls notwendig können auch die der Kartierung zugrundeliegenden Sensordaten einbezogen werden. Wenngleich ein solches Vorgehen zu subjektiven Ergebnissen führen kann, ist es sehr verbreitet [23].

Eine objektivere Alternative zur Evaluierung von Occupancy Grids stellen Metriken dar. Steht eine so genannte *Ground Truth* zur Verfügung, die verlässlich die tatsächlichen Gegebenheiten beschreibt, ist deren Einsatz leicht möglich. Beispielsweise kann die *OverallError*-Metrik aus [9] verwendet werden:

$$OverallError := \sum_{i=1}^{sensedSize} |grid_i - truth_i|.$$

$sensedSize$ gibt die Anzahl der Zellen an, die sich aktuell im Messbereich befinden, und $grid_i$ und $truth_i$ repräsentieren die Belegtheitswahrscheinlichkeiten der Zelle mit Index i im zu evaluierenden Grid bzw. Ground-Truth-Grid. Da das Ground-Truth-Grid den wahren Zustand jeder Zelle beschreibt, besitzt es nur die Wahrscheinlichkeiten 0 (Zelle ist zweifelsfrei *frei*) und 1 (Zelle ist zweifelsfrei *belegt*). Je niedriger das Ergebnis der OverallError-Metrik ist, desto besser wird die Kartenqualität angenommen. Für evidenztheoretische Occupancy Grids kann die erweiterte Formulierung aus [8] verwendet werden, die berücksichtigt, dass in diesem Fall statt einer Wahrscheinlichkeit zwei Massen (für *belegt* und *frei*) einzubeziehen sind.

Leider steht in der Praxis meist – wie auch im Falle vorliegender Arbeit – keine Ground Truth zur Verfügung, da die Erzeugung verlässlicher Grundwahrheiten ein aufwendiges und schwieriges Unterfangen ist [32], das oftmals manuell durchgeführt werden muss. Hinzukommt, dass eine – vor allem manuell generierte – Ground Truth typischerweise aus sehr aufgabenspezifischen Merkmalen besteht und dadurch für eine allgemeine Evaluierung wenig aussagekräftig ist. Zum Beispiel enthält die in [49][34] zur Evaluierung einer Straßenverlaufserkennung manuell erzeugte Ground Truth nur die dafür relevanten Merkmale des Straßenverlaufs.

Kommt die Dempster-Shafer-Theorie zum Einsatz, ergibt sich aus der Con-Metrik (vgl. Abschnitt 3.2.2) eine interessante Alternative. In [9][21] wird Con in der als *AverageConflict* bezeichneten Metrik verwendet, um auch ohne Ground Truth ein Maß für die Kartenqualität zu erhalten:

$$AverageConflict := \sum_{i=1}^{size} \frac{Con_i}{sensedSize}.$$

$sensedSize$ gibt wieder die Anzahl der Zellen an, die sich aktuell im Messbereich befinden, und $size$ beschreibt die Gesamtzahl aller Zellen im Grid; Con_i stellt den in der Zelle mit Index i aufgetretenen Konflikt dar. Die zugrundeliegende Annahme ist, dass, da Con ein

Maß für detektierte Inkonsistenzen ist, der durchschnittliche Konflikt auf eine mindere Kartenqualität schließen lässt.

Dies ist im Allgemeinen richtig. In unserem Fall wird der Konflikt jedoch zur Dynamikerkennung benutzt, da er auch ein Indiz für dynamische Objekte darstellt (vgl. Abschnitt 4.2.3.1). Der Konflikt entsteht also korrekter- und – im Sinne der Dynamikerkennung – auch erwünschterweise, womit die AverageConflict-Metrik für uns kein geeignetes Maß für die Qualität des Occupancy Grids ist.

Es gibt noch viele andere Qualitätsmetriken für Karten, die hauptsächlich aus der Robotik stammen, da Karten dort eine längere Tradition besitzen. Nach [23] lassen sich aufgabenabhängige und aufgabenunabhängige Metriken unterscheiden. Im Falle aufgabenabhängiger Metriken werden aus der Karte zunächst Merkmale extrahiert, die relevant für die betrachtete Aufgabe sind, und dann für die Bewertung genutzt; dadurch lassen sich die Ergebnisse üblicherweise nicht auf andere Aufgaben übertragen. Da die OG-Bibliothek in ganz unterschiedlichen Anwendungsszenarien zum Einsatz kommen kann, scheint eine aufgabenunabhängige Metrik geeigneter.

Allerdings kommt eine Untersuchung [23] zu dem Ergebnis, dass die zellweise und aufgabenunabhängig operierenden Metriken aus der Robotik in scheinbar systematischer Art und Weise im automobilen Umfeld scheitern. Zurückgeführt wird das auf die unterschiedlichen Rahmenbedingungen: In der Robotik werden oftmals offline globale Karten erzeugt, wobei das Hauptziel eine große Karte hoher Qualität darstellt; im Automobilbereich werden typischerweise Karten des lokalen Fahrzeugumfelds in Echtzeit erzeugt, die aufgrund der Fortbewegung des Fahrzeugs aus relativ wenigen Messungen resultieren und in erster Linie als Grundlage für zu treffende Entscheidungen dienen (vgl. Abschnitt 2.1). [23] empfiehlt deshalb die Verwendung aufgabenspezifischer Metriken, die zudem auch die automobilen Rahmenbedingungen berücksichtigen. Eine solche Metrik hat aber eben im Rahmen dieser Arbeit wenig Bedeutung, da keine konkrete Aufgabe existiert.

Im nächsten Abschnitt werden die mit der OG-Bibliothek erzielten Resultate deshalb nicht formal mithilfe einer Metrik evaluiert, sondern durch eine manuelle, visuelle Inspektion bewertet. Wie erwähnt ist ein solches Vorgehen – nicht zuletzt aufgrund der eben beschriebenen Situation – üblich, wenngleich dies zu subjektiven Ergebnissen führen kann. Eine konkrete Anwendung, die die OG-Bibliothek zur Erstellung von Occupancy Grids nutzt, sollte deshalb zusätzlich eine metrikbasierte, aufgabenspezifische Evaluierung durchführen. Die Erstellung der dafür notwendigen Ground Truth könnte durch Einsatz eines unterstützenden Tools [23] vereinfacht werden.

5.3 Anwendungsspezifische Ergebnisse

In diesem Abschnitt werden praktische Ergebnisse anhand des exemplarisch implementierten Anwendungsfalls beschrieben. Dabei werden hauptsächlich charakteristische Szenen gezeigt und erläutert; auf eine weitergehende Evaluierung wird aus den im vorherigen Abschnitt aufgezeigten Gründen verzichtet. Hauptanliegen dieses Abschnitts ist es, die Leistungsfähigkeit und die Möglichkeiten der OG-Bibliothek zu demonstrieren.

5.3.1 Besprechung charakteristischer Szenen

Für den implementierten Anwendungsfall wurden neun Datensätze bereitgestellt, die bei Messfahrten aufgezeichnet wurden. Aus diesen Datensätzen wurden durch ADTF-Simulationen (vgl. Abschnitt 4.3) im Rahmen intensiver Tests mit der OG-Bibliothek Occupancy Grids erstellt. Entsprechend der dabei erzielten Ergebnisse flossen zahlreiche Änderungen in die Implementierung der OG-Bibliothek ein, die dadurch ihre in Kapitel 4 beschriebene Gestalt erhielt. Die Änderungen betrafen dabei insbesondere die Steigerung der Ausführungsgeschwindigkeit und das Finden geeigneter Massefunktionen (vgl. Abschnitt 5.1).

Nach Fertigstellung der OG-Bibliothek wurden zu Evaluierungszwecken noch weitere ADTF-Simulationen durchgeführt, deren Resultate in diesem Abschnitt präsentiert werden. Da mehrere Stunden simuliert wurden, beschränken sich die Ausführungen auf eine Auswahl charakteristischer Szenen. Als Ergänzung zur nachfolgenden, textuellen Beschreibung sei an dieser Stelle auf die DVD im Anhang hingewiesen, auf der sich Aufzeichnungen einiger durchgeführter Simulationen befinden. Diese Videos sollen vor allem ein Gefühl für die Dynamik des Ablaufs und die Ausführungsgeschwindigkeit vermitteln.

Bei der nachfolgenden Besprechung einzelner Szenen werden die Occupancy Grids aus Platzgründen auf den jeweils relevanten Bereich beschnitten und teilweise nicht in Originalgröße abgebildet. Die Situation wird durch Bilder des Referenzvideos dargestellt, die den Bereich vor dem eigenen Fahrzeug an seiner aktuellen Position zeigen. Die aktuelle Position des eigenen Fahrzeugs ist im Occupancy Grid durch eine blau schraffierte Box dargestellt. Die Fahrtrichtung ist leicht an den Strahlen der Freiraummodellierung zu erkennen, da sie immer nach vorne gerichtet sind (Beispiel: In Abbildung 5.1 fährt das Fahrzeug in allen Occupancy Grids nach rechts). Es werden orange Hervorhebungen und Nummerierungen eingefügt, um die Besprechung zu erleichtern. Die teilweise vorhandenen schwarzen und roten Linien stammen von den in Abschnitt 4.3.2 beschriebenen Visualisierungen; sie können im Rahmen der Besprechung ignoriert werden.

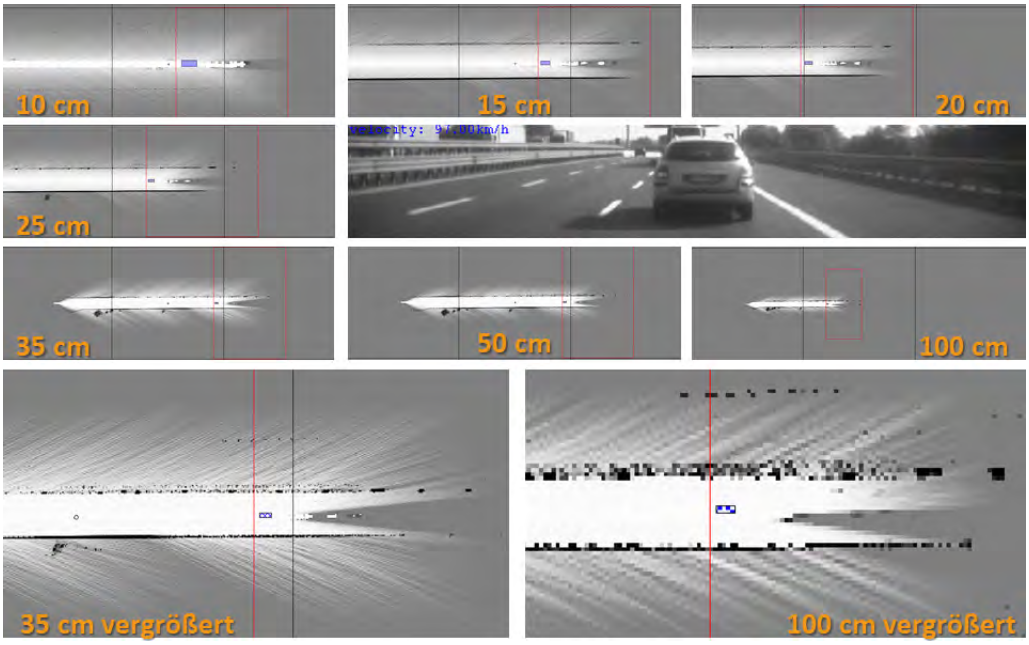

Abbildung 5.1: Eine Szene in sieben unterschiedlichen Auflösungen.

Beachtung verdienen dagegen die über die graphische Benutzeroberfläche des OGFilter in ADTF gewählte Größe und Auflösung (= Größe einer einzelnen Zelle) des Occupancy Grids, da sie entscheidend dessen Aussehen beeinflussen. Abbildung 5.1 demonstriert dies an sieben Occupancy Grids unterschiedlicher Auflösung, die alle dieselbe, im Referenzvideobild gezeigte Situation darstellen. Die unterschiedliche Auflösung führt dazu, dass sie sich deutlich voneinander unterscheiden, was insbesondere an den beiden vergrößerten Darstellungen unten in der Abbildung erkennbar ist.

Um die nachfolgenden Ausführungen zu vereinfachen, wird für alle im Folgenden betrachteten Occupancy Grids – sofern nicht explizit anders angegeben – eine Auflösung von 25 Zentimeter und eine Größe von $999 \times 999 = 998.001$ Zellen (entspricht bei genannter Auflösung $249{,}75 \times 249{,}75$ m^2) gewählt[1]. Diese Wahl ist realistisch und führt zu guten Kartierungsergebnissen. Sie schafft zum Einen einen vernünftigen Kompromiss zwischen repräsentiertem Bereich und Genauigkeit, zum Anderen ist die Ausführung auf dem zum Simulieren verwendeten Laptop (Intel Core i7-720QM, ATI Mobility Radeon HD 4650, 8 GB Arbeitsspeicher, Windows 7 Ultimate 64 Bit) praktisch in Echtzeit möglich, selbst wenn das Occupancy Grid zu jedem Zeitschritt berechnet und ausgegeben wird (vgl. Abschnitt 5.3.3).

In jedem Fall überrascht es, dass die OG-Bibliothek mit einer Auflösung von 25 Zentimetern bei knapp einer Million Zellen noch flüssig ausgeführt werden kann. Andere Arbeiten verwenden zwar ähnliche Auflösungen; typischerweise liegen sie zwischen 20 [54][4] oder 25 [48] Zentimetern und 50 Zentimetern [20][36], selten auch niedriger (z.B. 1 Meter in [35]). Bis auf [36] verwenden diese Arbeiten jedoch die klassische Variante, die weniger aufwendig ist, als die von der OG-Bibliothek implementierte evidenztheoretische Variante.

Betrachten wir nun die erste Szene, die in Abbildung 5.2 gezeigt ist. Das Fahrzeug hat gerade die Autobahn verlassen und befindet sich noch in der einspurigen Abfahrtsschleife (siehe Referenzvideobild). Das zugehörige Occupancy Grid ist im linken Teil der Abbildung zu sehen. Gut zu erkennen sind die einzelnen Strahlen der Freiraummodellierung (1), die mit zunehmender Entfernung immer schwächer werden und schließlich verschwinden.

Der orange eingerahmte Bereich (2) ist im rechten Teil detaillierter dargestellt. (2a) zeigt die zugehörige Situation. Vor dem Fahrzeug befinden sich zwei andere Fahrzeuge, die korrekt als *dynamisch* erkannt und entsprechend als *frei* kartiert werden (2b, 2c). Auf der rechten Seite des Fahrzeugs befinden sich zudem zwei Aussparungen in der Leitplanke, die gut in (2b) bis (2f) erkennbar sind. (2b) bis (2d) zeigen, wie sich das Fahrzeug an den Leitplanken vorbeibewegt und dabei das Occupancy Grid mit immer mehr Messungen aktualisiert wird. Während sich das Ergebnis von (2b) zu (2c) erheblich verbessert, verschlechtert es sich leicht von (2c) zu (2d). Dies liegt an einer falsch orientierten, rechteckigen Bounding Box des Lasersensors. Durch die Wahl der Massefunktionen ist deren Einfluss allerdings gering: Die eigentliche Kontur bleibt gut erkennbar und der Bereich davor wird von der Freiraummodellierung weitgehend bereinigt. (2e) zeigt das Szenario, wenn keine Boxen kartiert werden, sondern nur die einzelnen Messpunkte. Man sieht deutlich, dass sich ein kompletter Verzicht auf die Boxen nachteilig auswirkt, da die Ränder deutlich an Kontrast verlieren; wenngleich sie erkennbar bleiben[2]. (2f) zeigt das Szenario bei einer Zellgröße von einem Meter – hier werden die Boxen noch besser durch die Freiraummodellierung korrigiert.

[1] Dass nicht eine Größe von 1.000×1.000 Zellen verwendet wird, liegt an den automatischen Anpassungen des OGFilter, die dieser basierend auf den Vorgaben des OGBuilder durchführt (vgl. Abschnitt 4.3.2).

[2] Eine Alternative zu Bounding Boxes ist es, eine größere Region um die Messpunkte zu kartieren.

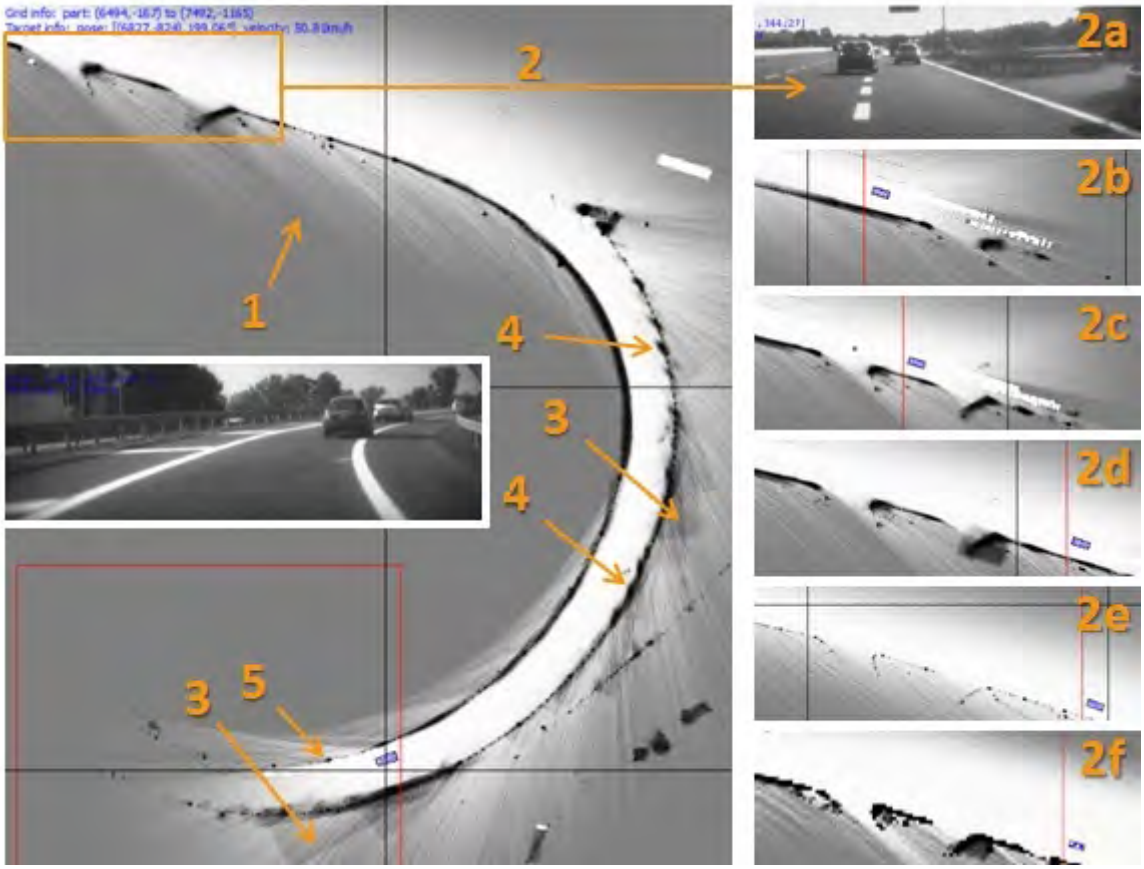

Abbildung 5.2: Detailbetrachtung einer Autobahnausfahrt.

In der linken Hälfte der Abbildung sieht man weitere Stellen, an denen die Boxen falsch orientiert sind, aber gut korrigiert werden (3).

Der Hauptteil der Kartierung erfolgt auf Basis der Lasersensordaten, da dieser mit 110 Grad einen großen horizontalen Öffnungswinkel besitzt, wohingegen der Radarsensor mit 28 Grad einen deutlich kleineren besitzt. In diesem engen Bereich vor dem Fahrzeug liefert der Radarsensor jedoch oftmals nützliche Daten, die die Laserdaten gut ergänzen. Die Radardaten sind deutlich an den größeren Punkten erkennbar (4), die durch die Verwendung einer zweidimensionalen Gaußfunktion im inversen Sensormodell des Radarsensors entstehen; die Lasermesspunkte werden punktförmig und damit kleiner kartiert (5).

Mithilfe der OG-Bibliothek kann die Dynamik von Objekten blau visualisiert werden (vgl. Abschnitt 4.2.5.3), was manchmal vorteilhaft sein kann, wie die folgenden beiden Szenen verdeutlichen. Abbildung 5.3 zeigt oben eine Szene mit zwei vorausfahrenden Autos. Beide werden in (1) korrekterweise als *dynamisch* erkannt und deshalb als *frei* kartiert. Bei visualisierter Dynamik (2) ist die Szene jedoch deutlich klarer, da die beiden dynamischen Objekte von den ebenfalls vorhandenen statischen Freiflächen unterschieden werden können. Unten in der Abbildung herrscht reger Verkehr. In (3) sind Verschattungen vor dem eigenen Fahrzeug erkennbar, allerdings ist die Ursache nicht ersichtlich, da dort kein Hindernis kartiert ist. Auch in diesem Szenario ist die Visualisierung dynamischer Bereiche lohnend (4), da diese explizit die ursächlichen, vorausfahrenden Fahrzeuge darstellt.

Abbildung 5.3: Vorteile der Dynamikvisualisierung.

Abbildung 5.4: Zurückbleibende Artefakte bei Kartierung eines dynamischen Objekts.

Wie in Abschnitt 4.2.4.2 beschrieben wurde, ist in unserem Fall die Verwendung der vom Lasersensor gelieferten Geschwindigkeiten nicht möglich, da der Fehler einiger Messungen zu groß ist. Deshalb wird stattdessen direkt die sensorinterne Klassifikation verwendet. Auch diese ist allerdings nicht fehlerfrei, wie Abbildung 5.4 zeigt. Zunächst nähert sich das eigene Fahrzeug einem LKW und dieser wird korrekt als solcher klassifiziert; entsprechend werden – er ist ein dynamisches Objekt – die zugehörigen Zellen als *frei* kartiert (1). Dann ändert sich allerdings die Klassifikation: der LKW wird nicht mehr als dynamisches Objekt erkannt und folglich fälschlicherweise kartiert (2). Nahe am LKW ändert sich zwar die Klassifikation wieder zur korrekten Klasse *LKW* (3), aber es bleiben erkennbare Artefakte zurück. Eine erneute Fehlklassifikation führt zu weiteren Artefakten (4). Diese bleiben bestehen (5), bis sich das eigene Fahrzeug am LKW vorbeibewegt hat und dieser aus dem Erfassungsbereich des Sensors verschwunden ist (6). Ein solches Verhalten scheint aufgrund der fehlerhaften Sensordaten nur mit unverhältnismäßig großem Aufwand vermeidbar.

Die bisherigen Beispiele zeigen: Meist ist der Verlauf der Straße schön im Occupancy Grid ersichtlich, da die Objekte, die den Straßenrand beschreiben, sehr gut kartiert werden. Ist keine Leitplanke vorhanden, ist der Straßenrand oft erkennbar, wenn man die Kartierungen der Leitpfosten am Straßenrand zu einer Begrenzung verbindet. Dies ist gut an den Szenen (1) und (2) in Abbildung 5.5 zu erkennen. Auch in (3) ist der Straßenverlauf deutlich erkennbar. Sehr gut kartiert ist die massive Betonbande links; die Leitplanke (in Fahrtrichtung rechts neben dem eigenen Fahrzeug) resultiert in einer durchgezogenen, linienförmigen Kartierung und die Leitpfosten davor zeigen gut den Verlauf durch einzelne Punkte. (4) zeigt eine ganz ähnliche Situation, allerdings scheitert die Kartierung der rech-

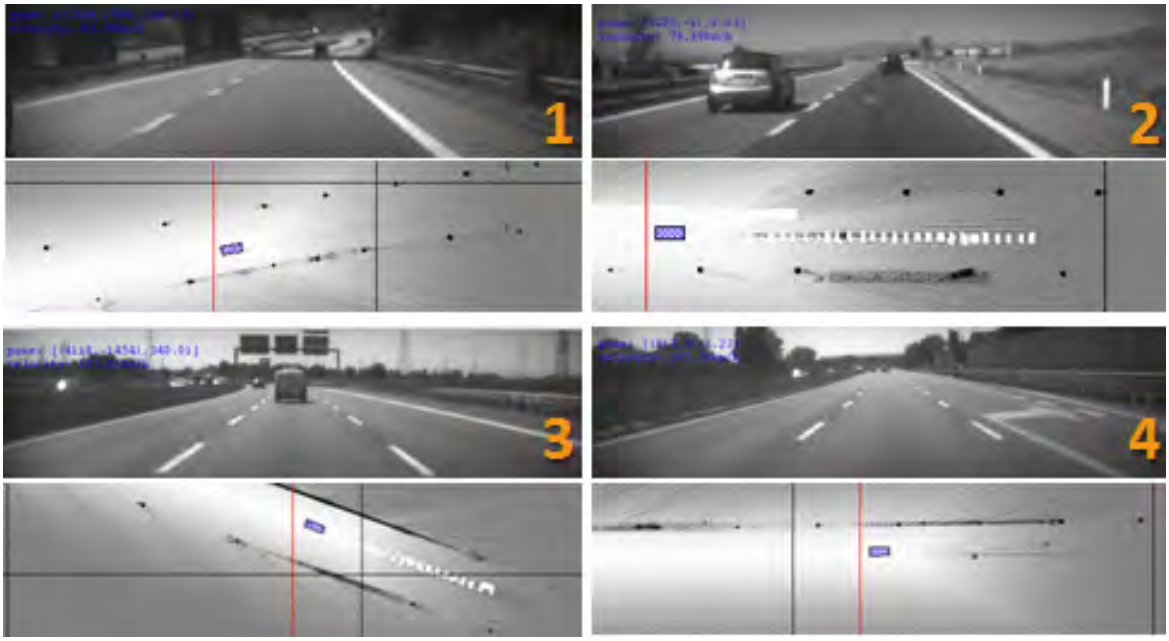

Abbildung 5.5: Weitere Erscheinungsformen des kartierten Straßenverlaufs.

ten Bande, die aus einer Leitplanke und einer Lärmschutzwand besteht. Auch hier ist das Verhalten auf die Sensorik zurückzuführen, das Occupancy Grid ist bezüglich der bereitgestellten Daten korrekt: Eine manuelle Sichtung der Sensordaten zeigt, dass hier einfach keine Messwerte vorhanden sind.

Landstraßen-Szenarios besitzen üblicherweise keine festen Begrenzungen (z.B. Leitplanken), trotzdem werden auch diese sehr gut kartiert – ein Beispiel ist in Abbildung 5.6 zu sehen. Die Begrenzungen bilden Schilder und Leitpfosten (1, 2). Diese führen dazu, dass tatsächlich nur die Fahrbahn als frei kartiert wird, die sich dadurch gut von der restlichen Umgebung unterscheiden lässt. Das vorausfahrende Fahrzeug wird meist korrekt als dynamisches Objekt erkannt (2, 4), genauso wie der Gegenverkehr. Zwischenzeitlich ist allerdings die vom Sensor gelieferte Klassifizierung fehlerhaft, so dass es zu unerwünschten Artefakten in der Karte kommt (3).

Ein ähnliches Szenario zeigt Abbildung 5.7. In (1) ist die Verschattung, die Verkehrsschilder und Leitpfosten während der Freiraummodellierung erzeugen, sehr deutlich erkennbar. Im Vorbeifahren wird der Bereich im Schatten noch teilweise abgetastet (2), so dass der verschattete Bereich deutlich kleiner wird (3).

Ein weiteres interessantes Ergebnis stammt von einer Fahrt durch einen Kreisverkehr, es ist in Abbildung 5.8 (1) gezeigt. Das Eigenfahrzeug kommt von links und verlässt den Kreisverkehr auch nach links. Dass der Kreisverkehr wirklich kreisförmig in der Karte aussieht überrascht – dies war aufgrund der Verwendung von Odometriedaten nicht unbedingt zu erwarten und ist vor allem auf die Verwendung eines relativ komplexen Bewegungsmodells zurückzuführen (vgl. Abschnitt 4.2.4.1).

Die dreieckige Verkehrsinsel an der genommenen Ein- und Ausfahrt ist schön kartiert, genauso die Mittelinsel im Zentrum des Kreisverkehrs und auch die anderen Ein- und Ausfahrten sind ersichtlich. Der äußere Rand ist meist ebenfalls gut erkennbar – lediglich unten links fehlt er. Oben mittig und unten rechts kommt es zu Verschmierungen durch die Boun-

Abbildung 5.6: Landstraße ohne feste Begrenzung.

Abbildung 5.7: Verschattung durch Leitpfosten und Schilder.

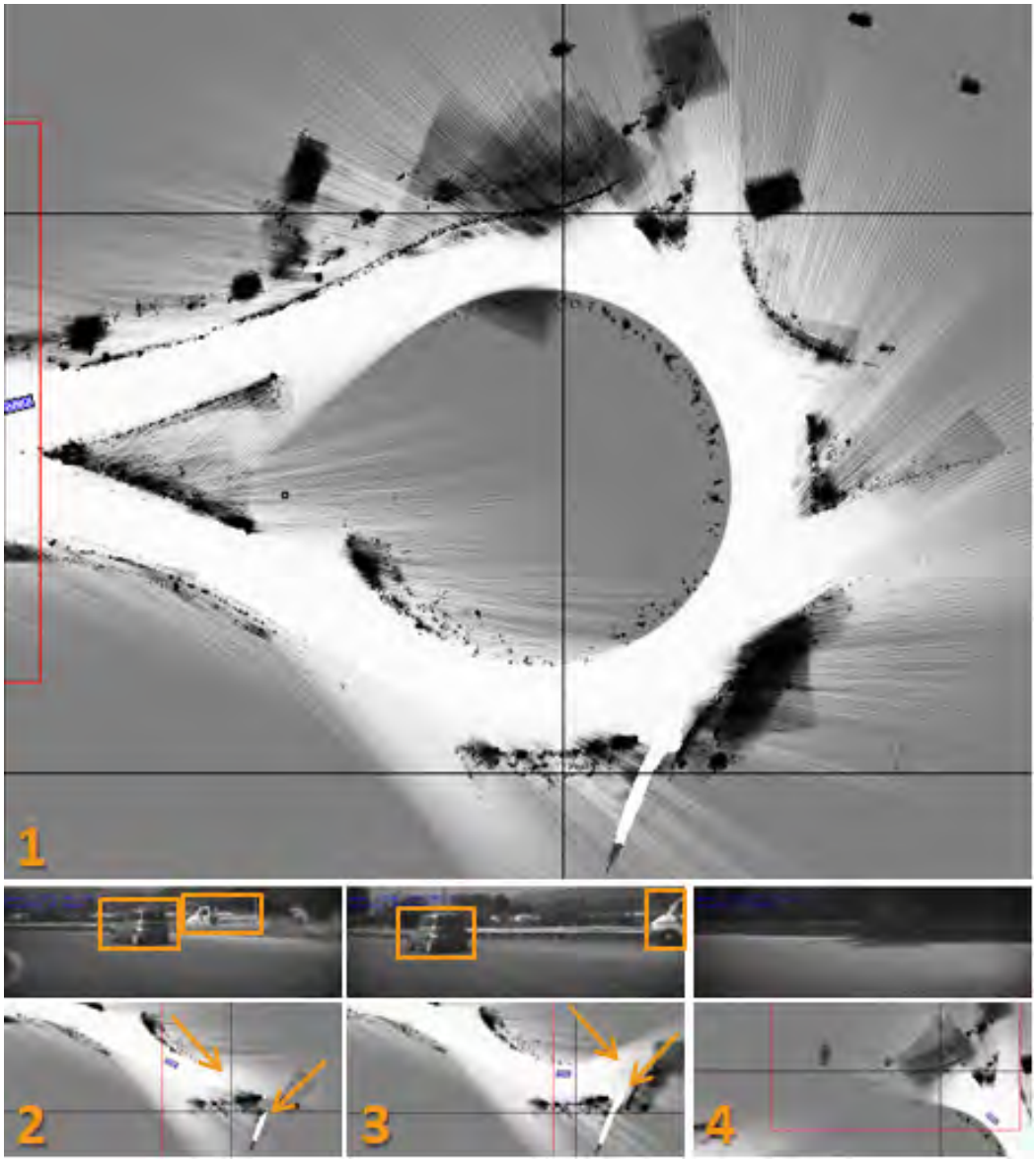

Abbildung 5.8: Durchfahren eines Kreisverkehrs.

ding Boxes des Lidars, die aber durch die Freiraummodellierung kompensiert werden. (4) zeigt die obere Verschmierung bevor die Freiraummodellierung zu dem verbesserten Ergebnis von (1) führt.

(2) und (3) illustrieren, wie dynamische Objekte zur Kartierung des Freiraums beitragen, insbesondere in Ein- und Ausfahrten. Durch die Freiraummodellierung selbst werden Ein- und Ausfahrten nur angedeutet, da der Öffnungswinkel der Freiraummodellierung beschränkt wurde (vgl. Abschnitt 4.2.4.2). Für viele Anwendungen sollte dies jedoch ausreichend sein, da man in erster Linie an dem Bereich vor dem Fahrzeug interessiert ist und nicht an dem seitlich davon.

In Abbildung 5.9 ist ein Stadtszenario gezeigt. In der ersten Zeile (1, 2) ist zu sehen, wie sich das eigene Fahrzeug an einer Einfahrt vorbeibewegt. Diese ist in der Karte zwar zu erkennen, aber der kleine Öffnungswinkel der Freiraummodellierung verhindert wieder

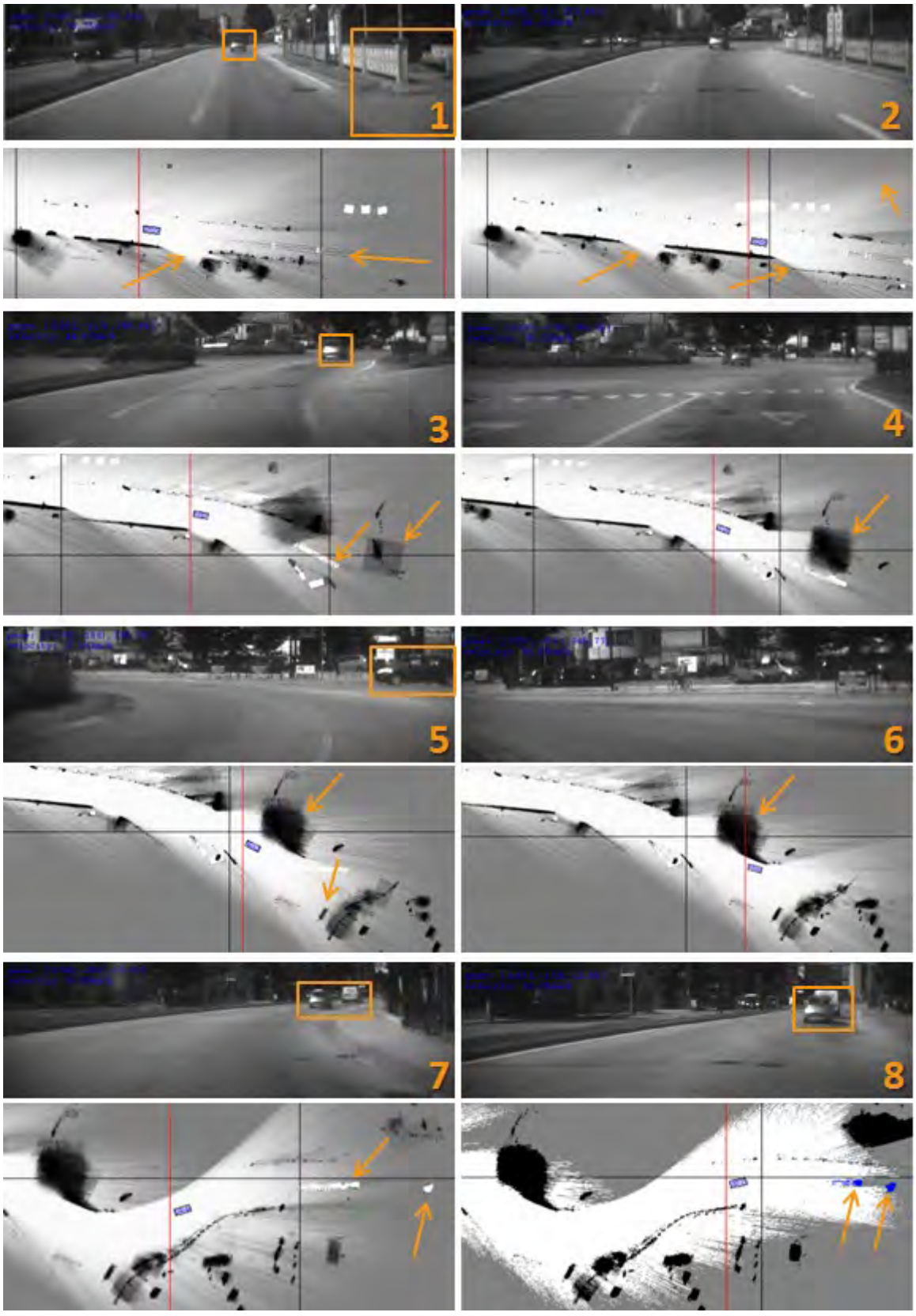

Abbildung 5.9: Stadtszenario mit Mittelinsel und Kreisverkehr.

Abbildung 5.10: Gut kartierte Abzweigung.

eine deutlichere Kartierung. Besser werden Abzweigungen kartiert, die etwa parallel zur Fahrtrichtung orientiert sind (vgl. Abbildung 5.10; dort sind auch schön die Radardaten zu sehen, die die Hauptstraße und die abzweigende Straße deutlich voneinander separieren).

Der rechte Straßenrand ist in Abbildung 5.9 sehr gut, durchgehend kartiert. Der linke ist ebenfalls erkennbar, wenngleich nicht ganz so gut. Das vorausfahrende Fahrzeug wird bereits erkannt, obwohl es erst für kurze Zeit verfolgt wird; gleiches gilt für den Gegenverkehr.

(3) bis (8) zeigen das gerade Durchfahren und Verlassen eines Kreisverkehrs. Wieder deutlich erkennbar sind die Verschmierungen durch die Lidar-Bounding-Boxes und deren Korrektur durch die Freiraummodellierung. Ein von rechts kommendes, an der Einfahrt wartendes Fahrzeug wird unerwünschterweise als statisch kartiert (5), da es sich während der ganzen Vorbeifahrt nicht bewegt; ein solches Verhalten ist nicht vermeidbar.

In der letzten Zeile (7, 8) befinden sich zwei Fahrzeuge vor dem eigenen Fahrzeug, die korrekt klassifiziert und kartiert werden. In (8) warten beide bereits längere Zeit an einem Zebrastreifen, dennoch werden sie weiterhin korrekt als *dynamisch* klassifiziert. Dies ist insbesondere auf das sensorinterne Objekttracking zurückzuführen, das die Klassifizierung über einen längeren Zeitraum verfolgt. Zudem zeigt (8) in Vergleich mit (7), dass auch eine Darstellung mit nur vier Werten (weiß/grau/schwarz für den Belegtheitswert und blau für den Dynamikwert) eine adäquate Repräsentation sein kann: Es geht zwar die Information über die Unsicherheiten verloren, dafür ist aber der Straßenverlauf deutlicher erkennbar.

In Abbildung 5.11 ist der Fortgang des Szenarios gezeigt: Die beiden Fahrzeuge bleiben stehen, während ein Fußgänger die Fahrbahn überquert. Zunächst werden beide Fahrzeuge noch immer korrekt als *dynamisch* klassifiziert (1). Mit zunehmender Standzeit ändert sich ihre Klassifizierung jedoch zu *statisch*, wodurch sie nun unerwünschterweise kartiert werden (2, 3a). Den Übergang von einem dynamischen zu einem statischen Objekt kann man erkennen, wenn man die Dynamik visualisiert (3b). Auch der Fußgänger wird zunächst kartiert (1 - 3b), da seine Klasse in der vorliegenden Implementierung nicht als dynamisches Objekt berücksichtigt wird – die Freiraummodellierung beseitigt ohnehin rasch dessen Spur (4, 5). Eine Erwähnung wert sind in diesem Szenario noch die Radardaten, die zu einer deutlichen Kartierung der Fahrbahnmittelinsel führen (2); hier wären die sonst dominanten Lidardaten alleine zu wenig.

Ein weiteres innerstädtisches Szenario, dieses Mal mit stärkerem Verkehr, ist in Abbildung 5.12 gezeigt. In (1) erfasst der Radarsensor zwar nicht die Mittelinsel selbst, macht jedoch deren Enden durch das Verkehrsschild und den Laternenmast deutlich erkennbar. Gleiches gilt für (3), nur dass hier die Laternen durch den Lasersensor detektiert werden.

Zudem werden die vielen Fahrzeuge richtigerweise als *frei* kartiert, der Bereich hinter ihnen ist erkennbar verschattet. Vorteile bietet deshalb wieder die Dynamikvisualisierung (2), die nicht nur den Grund für die Verschattung deutlicher macht, sondern auch über das Verkehrsaufkommen informiert.

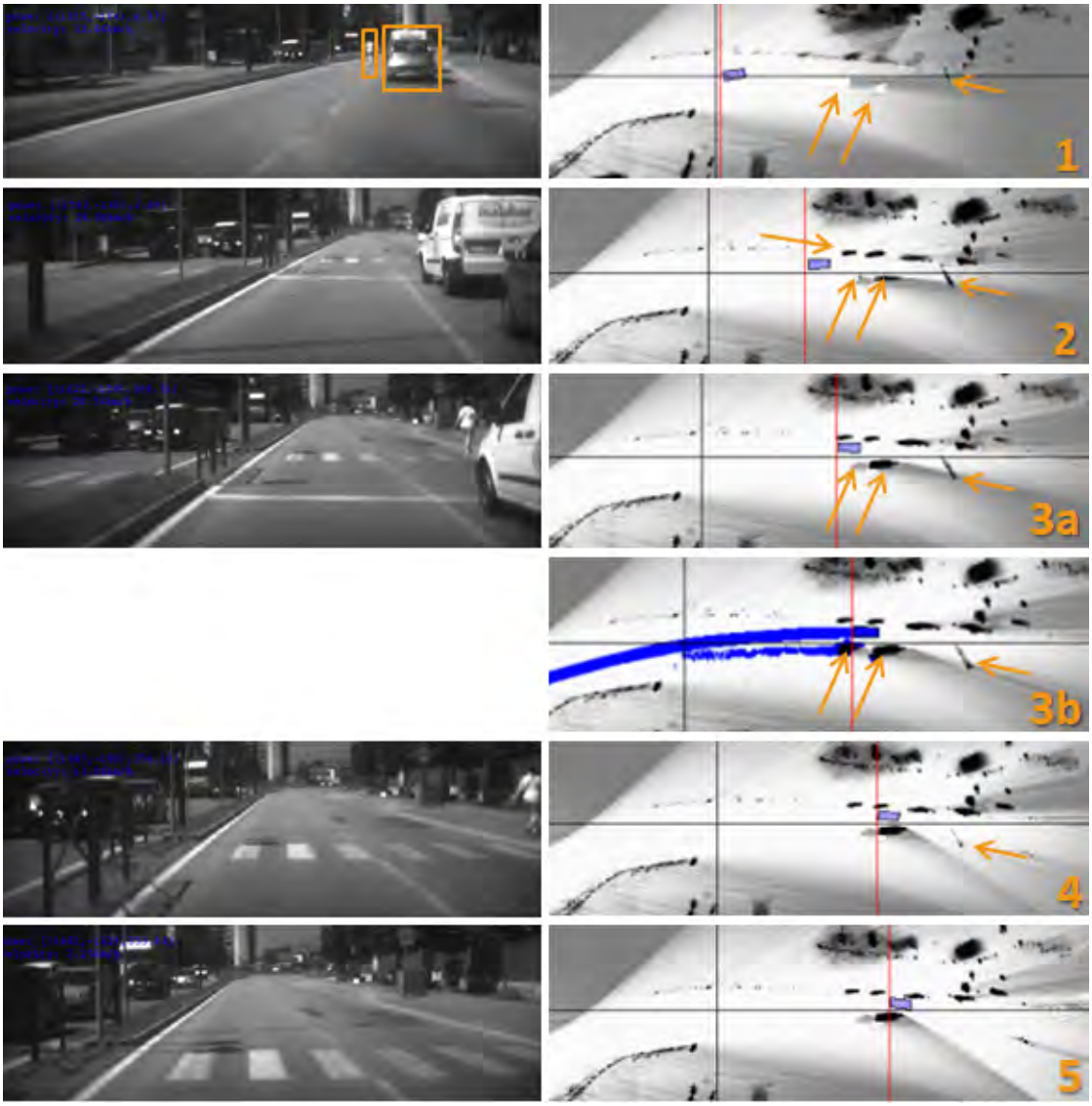

Abbildung 5.11: Kreuzender Fußgänger und stehende Fahrzeuge.

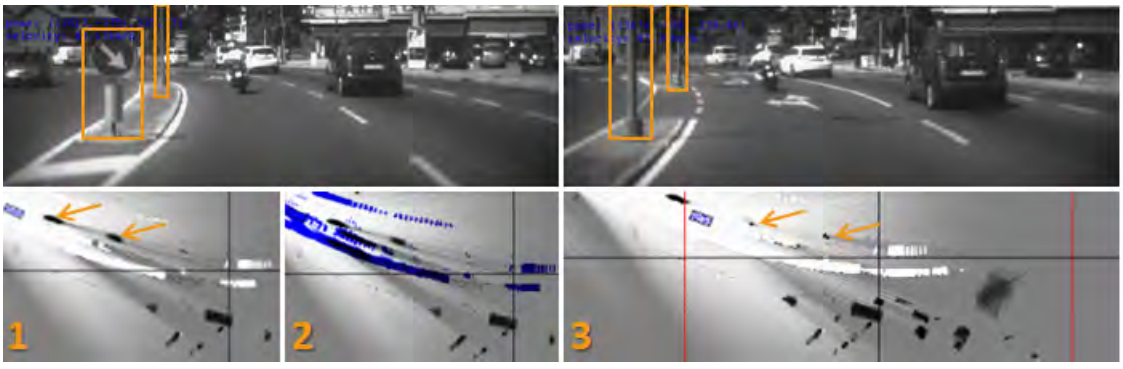

Abbildung 5.12: Mittelinsel in stärkerem Stadtverkehr.

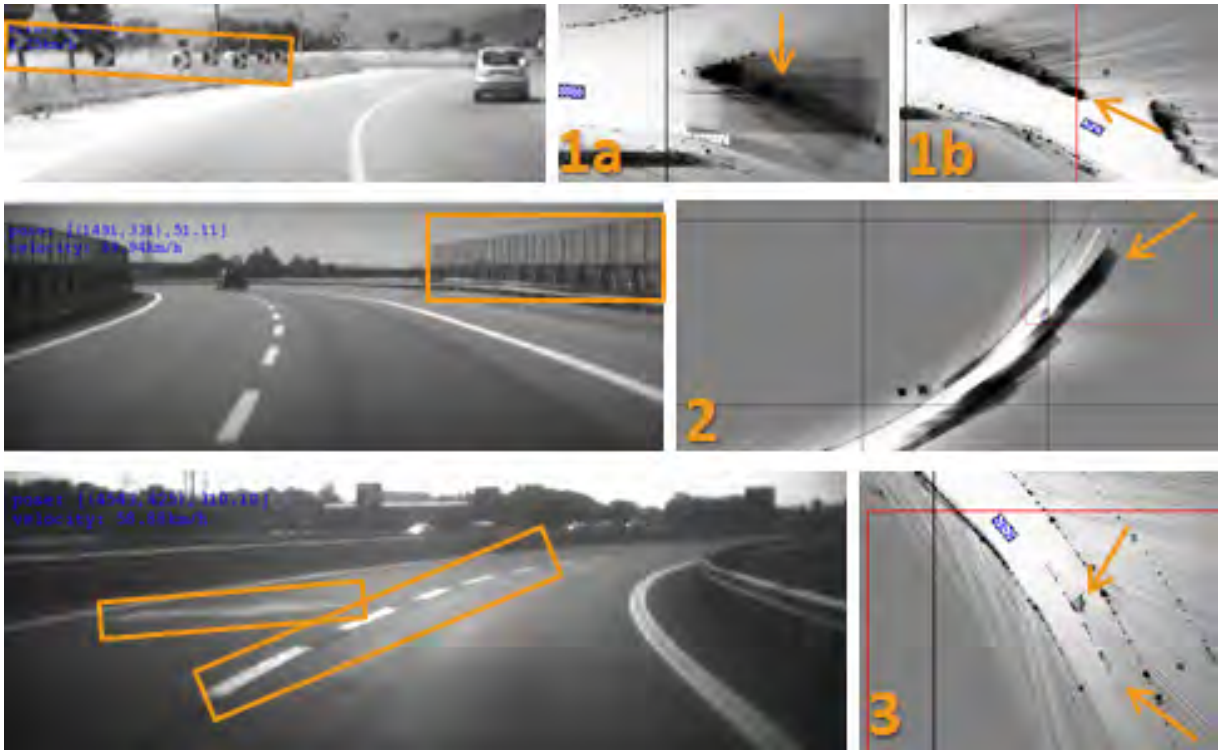

Abbildung 5.13: Situationen mit gestörten Sensordaten.

Abschließend sollen noch einige weitere Szenarien besprochen werden, in denen die Sensoren gestörte Daten liefern. Diese sind in Abbildung 5.13 gezeigt. In der ersten Zeile ist die Einfahrt in einen Kreisverkehr zu sehen, wobei sich auf der Verkehrsinsel an der Einfahrt Warnbaken befinden. Diese Warnbaken führen dazu, dass die interne Vorverarbeitung des Lasersensors falsch orientierte Bounding Boxes liefert, die starke Verschmierungen erzeugen (1a) – die die Freiraummodellierung jedoch gut kompensiert (1b).

Ähnliche Verschmierungen, allerdings deutlich intensiver, entstehen durch an den Leitplanken angebrachte Gitter; diese können aufgrund ihrer Intensität jedoch nur teilweise kompensiert werden (2).

Eine weitere Störungsquelle sind Bodenreflexionen, insbesondere von Straßenmarkierungen aufgrund derer retroreflektiver Eigenschaften. Diese können nicht von Hindernissen unterschieden werden und werden deshalb ebenfalls kartiert (3): Die Mittellinie und die Pfeilmarkierung sind gut erkennbar. Meist werden kartierte Bodenreflexionen jedoch relativ rasch durch die Freiraummodellierung beseitigt.

5.3.2 Vergleich mit anderen Arbeiten

Ebenfalls interessant ist ein Vergleich der eben vorgestellten Ergebnisse mit den Ergebnissen anderer Arbeiten. Die Abbildungen 5.14 bis 5.18 zeigen zu Vergleichszwecken einige Occupancy Grids anderer Ansätze. Es ist erkennbar, dass die im Rahmen dieser Arbeit mit der OG-Bibliothek erstellten Occupancy Grids (siehe vorheriger Abschnitt) den in anderen Arbeiten vorgestellten rein optisch in nichts nachstehen und teilweise sogar besser aussehen.

Offensichtlich ist die Aussagekraft eines solchen, rein visuellen Vergleichs sehr beschränkt, da viele Faktoren unberücksichtigt bleiben (z.B. Qualität der Simulationsdaten oder unter-

schiedliche Sensorik); mehr als Bildmaterial steht für einen Vergleich der Arbeiten aber leider nicht zur Verfügung, so dass wir es dennoch bei dieser optischen Gegenüberstellung belassen müssen.

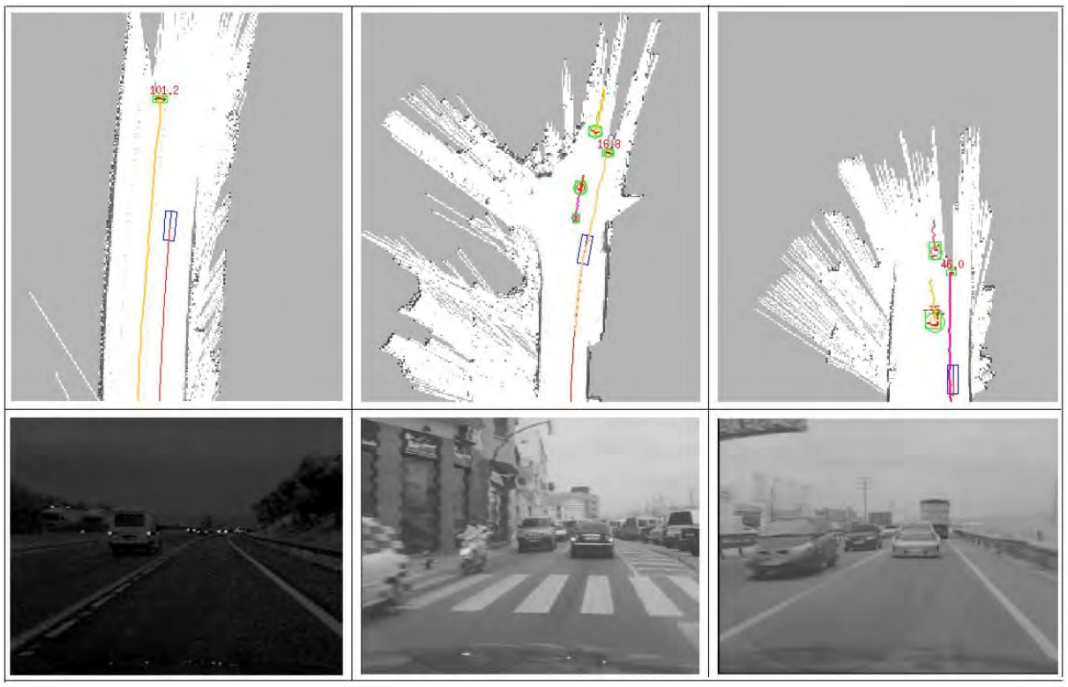

Abbildung 5.14: Drei in [3] präsentierte Szenen, wobei für dynamische Objekte auch das Tracking und die Geschwindigkeitsbestimmung erkennbar sind. Ebenfalls ersichtlich ist, dass die Freiraumfunktion für den ganzen Strahl einen konstanten Wert liefert, unabhängig von der Distanz.

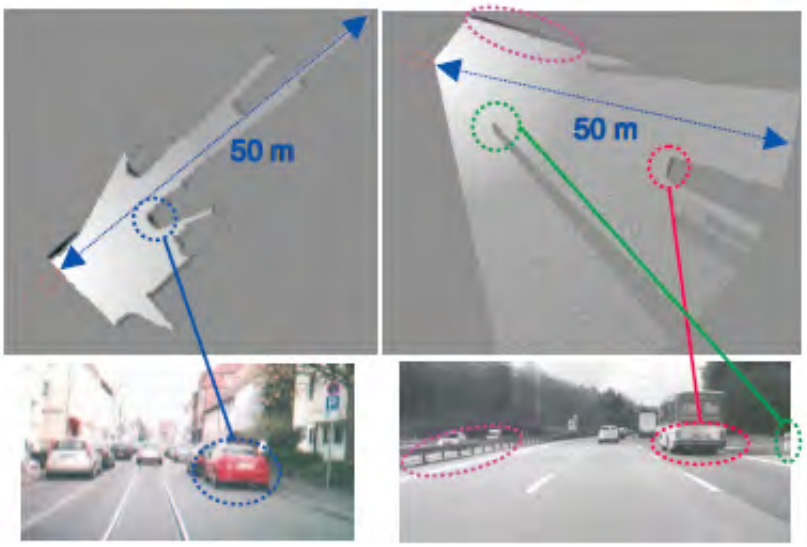

Abbildung 5.15: Zwei Szenen aus [54]. Auffallend ist die schöne Gleichmäßigkeit der Strahlen, allerdings entstehen Artefakte durch die dynamischen Objekte (links das Fahrzeug in der Mitte, rechts der Bus).

Abbildung 5.16: Ergebnis der drei Jahre später präsentierten Weiterentwicklung [33] auf einer Landstraße. Die Ränder werden durch das sehr hohe Gebüsch gut erkannt, der Blick voraus ist in dieser Szene beschränkt (die Kurve ist noch nicht kartiert).

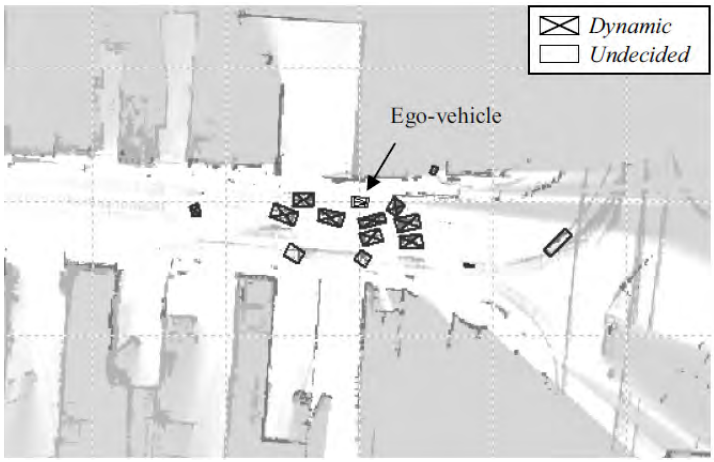

Abbildung 5.17: Das Occupancy Grid in [48] lässt gut den Einsatz eines 360-Grad-Laserscanners erkennen: die Freiraummodellierung erfasst auch die seitlichen Straßen vollständig. Ob es sich bei den grauen Spuren im rechten Teil um Fehler oder erwünschtes Verhalten handelt, bleibt aufgrund eines fehlenden Referenzbildes unklar.

Abbildung 5.18: Eine in [27] präsentierte Szene auf der Autobahn, das Ergebnis wurde parallelisiert mit dem Grafikprozessor berechnet. Auffällig sind die eigentlich unerwünschten Artefakte durch den vorausfahrenden LKW.

5.3.3 Fazit und Laufzeitoptimierungen für andere Anwendungen

Der vorherigen beiden Abschnitte haben ein Gefühl vermittelt, wozu die Implementierung unseres exemplarischen Anwendungsfalls in der Lage ist. Zusammenfassend kann man konstatieren, dass qualitativ hochwertige Occupancy Grids erzeugt werden – trotz der eingangs beschriebenen Herausforderungen, zu denen insbesondere eine teilweise unzureichende Datenqualität gehört. Das Resultat liegt vor allem in Bezug auf die Eigenpositionsbestimmung und die Simulationsgeschwindigkeit weit über den Erwartungen des Autors. Für andere Anwendungen, die ebenfalls auf Basis der OG-Bibliothek erstellt werden, wird deshalb sehr großes Potential gesehen – insbesondere, wenn ihnen bessere Daten zur Verfügung stehen und sie optimierte inverse Sensormodelle verwenden.

Allerdings bedeutet letzteres möglicherweise auch komplexere Berechnungen und dadurch eine höhere Ausführungszeit. Aus diesem Grund soll abschließend ein genauerer Blick auf die Laufzeit geworfen werden. Dazu wurde in mehreren ADTF-Simulationen die Ausführungsgeschwindigkeit der Beispiel-Implementierung gemessen, wobei die Größenparameter des dabei erstellten Occupancy Grids variiert wurden. Die Simulationen wurden wieder auf dem Testrechner aus Abschnitt 5.3.1 durchgeführt und jeweils mit den identischen Parametern für zwei unterschiedliche Szenarien wiederholt: eine zügige Autobahnfahrt inklusive Fahrtrichtungswechsel und eine langsamere Stadtfahrt inklusive Kreisverkehrdurchfahrt und Halt an einem Zebrastreifen (Szenen 1 und 2 auf der DVD im Anhang).

Die Ergebnisse sind in Tabelle 5.1 für das Autobahnszenario und in Tabelle 5.2 für das Stadtszenario gelistet; beide Tabellen besitzen den gleichen Aufbau. In der zweiten Spalte sind die Parameter des Occupancy Grids angegeben, das erzeugt wurde: die Zellgröße in Zentimeter und die Gridgröße in Zellen.

In der dritten Spalte finden sich die zugehörigen Laufzeitmesswerte in Mikrosekunden der durchgeführten ADTF-Simulationen. Es wurden jeweils drei Zeitmessungen vorgenommen und daraus der Mittelwert berechnet, um Schwankungen durch Nebenläufigkeiten und Hintergrundaktivitäten des Betriebssystems zu kompensieren. Die Zeitnahmen wurden mit den Profiling-Werkzeugen von ADTF durchgeführt; gemessen wurde dabei ab fertiggestellter Gridinitialisierung bis zum Simulationsende. Auf diese Weise wird wird die initiale Erzeugung der Grids von der Messung ausgenommen, da diese nichts über die Echtzeitfähigkeit aussagt (sie ist schließlich nur einmal zu Beginn nötig), aber das Messergebnis insbesondere für kürzere Szenen verfälschen würde.

In der letzten Spalte wird der Mittelwert der drei Laufzeitmesswerte mit der tatsächlichen Dauer des Szenarios verglichen, wobei das Ergebnis als Differenz (über die Szenariodauer hinausgehender Laufzeitbedarf in Millisekunden) und prozentual angegeben ist. Als tatsächliche Dauer des Szenarios wurde die Länge der hierfür verfügbaren Datenaufzeichnung gewählt (139.759.000 Millisekunden für das Autobahn- und 113.662.000 Millisekunden für das Stadtszenario). In die Laufzeitmessungen geht dadurch jeglicher im Rahmen der Simulation notwendiger Aufwand ein (z.B. Ausführung von ADTF, Betriebssystem, ...) – nicht nur der, der durch die OG-Bibliothek entsteht; es handelt sich also eher um pessimistische Ergebnisse.

Betrachtet man die Resultate der Laufzeitvergleiche, lassen sich einige Erkenntnisse gewinnen, von denen im Folgenden die wichtigsten skizziert werden sollen.

In den Simulationen A1/A5/A10 bzw. S1/S5/S10 bleibt die Gridgröße konstant bei 999×999 Zellen und nur die Zellgröße variiert (20, 25, 30 cm). Es ist erkennbar, dass

	OG-Größenparameter		Messungen (μs)				Vergleich	
	Zelle (cm)	Grid (Zellen)	1	2	3	Ø	Differenz (ms)	prozentual
A1	20	999×999	140303950	140275508	140284369	140287942,3	528,9423	100,3785
A2	20	1251×1251	163067904	162800791	162758999	162875898	23116,8980	116,5405
A3	25	603×603	139961287	139962483	139961260	139961676,7	202,6767	100,1450
A4	25	801×801	140000620	139994350	139994312	139996427,3	237,4273	100,1699
A5	25	999×999	139970169	139964274	139965524	139966655,7	207,6557	100,1486
A6	25	1203×1203	149936667	150378017	150285999	150200227,7	10441,2277	107,4709
A7	25	1401×1401	181434629	181992849	181944370	181790616	42031,6160	130,0744
A8	25	1599×1599	210022384	211491026	211587952	211033787,3	71274,7873	150,9984
A9	30	837×837	139961062	139962090	139961611	139961587,7	202,5877	100,1450
A10	30	999×999	139961263	139962308	139963827	139962466	203,4660	100,1456

Tabelle 5.1: Ergebnisse der Laufzeitmessungen für das Autobahnszenario.

	OG-Größenparameter		Messungen (μs)				Vergleich	
	Zelle (cm)	Grid (Zellen)	1	2	3	Ø	Differenz (ms)	prozentual
S1	20	999×999	113915504	113917671	113914638	113915937,7	253,9377	100,2234
S2	20	1251×1251	119774846	120059120	119765621	119866529	6204,5290	105,4588
S3	25	603×603	113880713	113878653	113879406	113879590,7	217,5907	100,1914
S4	25	801×801	113894962	113894233	113893611	113894268,7	232,2687	100,2044
S5	25	999×999	113911899	113917187	113914102	113914396	252,3960	100,2221
S6	25	1203×1203	115334964	115585567	115414265	115444932	1782,9320	101,5686
S7	25	1401×1401	130842026	132504887	131876332	131741081,7	18079,0817	115,9060
S8	25	1599×1599	149930775	150978015	150944983	150617924,3	36955,9243	132,5139
S9	30	837×837	113865288	113864382	113864516	113864728,7	202,7287	100,1784
S10	30	999×999	113865674	113865731	113864693	113865366	203,3660	100,1789

Tabelle 5.2: Ergebnisse der Laufzeitmessungen für das Stadtszenario.

die Ausführung für alle drei Zellgrößen praktisch in Echtzeit möglich ist. Allerdings ist zu berücksichtigen, dass – trotz zellmäßig konstanter Gridgröße – der repräsentierte Kartenausschnitt mit zunehmender Auflösung kleiner wird.

In den Simulationen A2/A5/A9 bzw. S2/S5/S9 wird deshalb die zellmäßige Gridgröße nicht konstant, sondern so gewählt, dass das Grid jeweils eine Größe von ungefähr 250×250 m^2 besitzt. Die Zellgröße beträgt wieder 20, 25 bzw. 30 cm. Während die niedrigen Auflösungen beide sehr und nahezu gleich schnell sind, ist die hohe Auflösung im Falle des Autobahnszenarios A2 mit einem Zeitbedarf von knapp 117% bezüglich der eigentlichen Ausführungsgeschwindigkeit deutlich langsamer. Diese Auflösung benötigt auch im Stadtszenario S2 mehr Zeit, wenngleich der Unterschied mit gut 105% nicht so deutlich ausfällt. Eine höhere Auflösung führt bei gleicher Gridfläche also erwartungsgemäß zu erhöhtem Zeitbedarf, wobei sich der Grad deutlich je nach Szenario unterscheiden kann.

Umgekehrt führt auch eine erhöhte Gridgröße bei gleichbleibender Zellgröße zu einem erhöhten Berechnungs- und dadurch Zeitaufwand. Dies veranschaulichen die Simulationen A3 bis A8 bzw. S3 bis S8 am Beispiel einer konstanten Auflösung von 25 cm. Die Anzahl der Zellen im Grid schwanken dabei zwischen gut 360.000 und deutlich über 2,5 Millionen

Zellen. Die durch das Grid repräsentierte Fläche liegt zwischen etwa 150×150 m² und etwa 400×400 m². Im Autobahnszenario ist eine verzögerungsfreie Ausführung bis zu knapp einer Million Zellen möglich (A5), im Stadtszenario sogar fast noch mit gut 1,4 Millionen Zellen (S6). Mit der höchsten simulierten Zellanzahl von gut 2,5 Millionen Zellen steigt der Laufzeitbedarf auf etwa 151% bzw. 133% (A8 bzw. S8).

Dass im Stadtszenario bei gleicher Laufzeit eine höhere Zellzahl möglich ist, ist auf die dort niedrigere Geschwindigkeit des Eigenfahrzeugs zurückzuführen. Diese führt dazu, dass das Fahrzeug länger im inneren Block des Hauptgrids verweilt. Dadurch sind weniger der aufwendigen und damit laufzeitintensiven Rotationen des RingGrid notwendig, das in unserer Simulation als Hauptgrid verwendet wird (vgl. Abschnitt 4.2.5.2).

So ist auch die Beobachtung zu erklären, dass A5 trotz größerem Grid schneller ausgeführt werden kann als A4. Zwar ist das Grid in A4 kleiner und die Berechnung damit prinzipiell weniger aufwendig als in A5, allerdings ist auch der innere Block kleiner, so dass das Fahrzeug diesen schneller verlässt, wodurch mehr Rotationen notwendig sind – dieser Aufwand überwiegt.

Wir haben gesehen, dass ein Occupancy Grid mit einer Größe von 999×999 Zellen bzw. etwa 250×250 m² und einer Auflösung von 25 cm praktisch in Echtzeit ausgeführt werden kann. Möglicherweise werden im Rahmen eigener Anwendungen jedoch komplexere inverse Sensormodelle oder ein noch größeres oder höher aufgelöstes Occupancy Grid benötigt; aufgrund der damit ansteigenden Berechnungskomplexität kommt es zu einem erhöhten Laufzeitbedarf, der möglicherweise deutlich von der Echtzeitfähigkeit entfernt liegt. Aus diesem Grund sollen an dieser Stelle noch kurz einige Maßnahmen aufgezeigt werden, für die eine wesentliche Erhöhung der Ausführungsgeschwindigkeit erwartet wird, die aber im Rahmen der Beispielimplementierung aufgrund der dort bereits sehr guten Geschwindigkeit nicht umgesetzt wurden.

Eine Möglichkeit – die in der Literatur oft, aber in der vorliegenden Arbeit noch überhaupt nicht zum Einsatz kommt – ist die Verwendung von Lookup-Tabellen. Deren Nutzung kann insbesondere in aufwendigen inversen Sensormodellen lohnen, beispielsweise zur Berechnung der Koordinatentransformationen, der Freiraummodellierungen oder – sofern eingesetzt – der zweidimensionalen Gaußfunktionen [4].

Die Implementierung arbeitet derzeit durchwegs mit doppelter Gleitkomma-Genauigkeit (Datentyp Double), die für die konkrete Anwendung möglicherweise nicht gewinnbringend ist. Es kann deshalb in Bezug auf die Laufzeit von Vorteil sein, die signifikanten Stellen zu reduzieren, wie zum Beispiel in [22] durchgeführt.

Auch die Ausführungsgeschwindigkeit der bereitgestellten Iteratoren kann bei Bedarf weiter gesteigert werden (oder es kommen eigene zum Einsatz). Insbesondere die notwendige Überprüfung, ob die aktuelle Zelle im gültigen Gridausschnitt liegt oder nicht, kann leicht optimiert werden. Derzeit wird initial überprüft, ob der gesamte Iterationsbereich gültig ist. Falls ja, ist die Überprüfung damit erledigt. Falls nicht, wird in jedem Iterationsschritt die jeweils aktuelle Zelle überprüft, was relativ aufwendig ist. Eine stufenweise Vorgehensweise (Überprüfung des halben, viertelten, ... Iterationsbereichs) wäre hier zum Beispiel im Allgemeinen schneller.

Wie eben bereits angesprochen, kommt es in der aktuellen Beispiel-Implementierung manchmal zu kurzen Verzögerungen, wenn das Eigenfahrzeug den inneren Block des Hauptgrids verlässt. Dies liegt daran, dass es dann zu einer Rotation des als Hauptgrid verwendeten RingGrid kommt und danach alle Zellen des Occupancy Grids neu berechnet werden

(vgl. Abschnitte 4.2.5.2 und 4.2.5.3). Nutzt man das Wissen, dass eine Rotation stattfindet, so lassen sich viele Zellwerte kopieren oder mit einem konstanten Wert initialisieren, anstatt sie zu berechnen, wodurch sich der Berechnungsaufwand stark reduziert. Die vorliegende Implementierung verzichtet darauf, um unabhängig von dem verwendeten Hauptgrid zu bleiben.

Eine alternative oder zusätzliche Möglichkeit ist es, das ausgegebene Occupancy Grid auf den relevanten Bereich um das Fahrzeug zu beschränken. Derzeit wird zu jedem Zeitpunkt das volle Occupancy Grid ausgegeben, das viele unbearbeitete Bereiche besitzt. Das Hauptgrid benötigt seine Größe, um unabhängig von der Fahrzeugausrichtung zu funktionieren; für die Ausgabe ist allerdings nur der kleinere, tatsächlich verwendete Bereich um das Fahrzeug interessant.

Kapitel 6

Zusammenfassung & Ausblick

In diesem Kapitel soll die Arbeit abschließend kurz zusammengefasst und ein Ausblick auf künftige Arbeiten gegeben werden.

6.1 Zusammenfassung

Erfassung und Modellierung des Fahrzeugumfelds sind ein wichtiger Bestandteil von Fahrerassistenzsystemen. Heutige Systeme erfassen das Umfeld typischerweise mit ein oder mehreren Umfeldsensoren und erstellen daraus ein objektbasiertes Modell, das die Umwelt durch die Eigenschaften einiger weniger dynamischer Objekte darstellt.

In künftigen Systemen wird der Einsatz mehrerer, heterogener Sensoren wichtiger, um die höheren Anforderungen bezüglich Verlässlichkeit, Zuverlässigkeit und Robustheit erfüllen zu können. Da auch die Zahl der Assistenzsysteme immer weiter steigt, muss das Umfeldmodell zudem eine Entkopplung zwischen Sensorik und Aufgaben schaffen, damit mehrere Systeme dieselbe Sensorik verwenden können. Künftige Umfeldmodelle benötigen außerdem eine weitergehende Modellierung, da die derzeit übliche, objektbasierte Modellierung einer der Hauptgründe für die aktuelle Beschränkung der Fahrerassistenzsysteme auf außerstädtische Regionen ist. Insbesondere die Verwendung eines hybriden Umfeldmodells erscheint erfolgversprechend, da das dabei hinzukommende kartenbasierte Modell aufgrund seiner flächigen Beschreibung des statischen Umfelds eine gute Ergänzung zur bereits vorhandenen Liste dynamischer Objekte darstellt. Prädestiniert für den Einsatz als ein solches kartenbasiertes Modell sind Occupancy Grids, zumal diese alle genannten Anforderungen an ein künftiges Umfeldmodell erfüllen können.

In der Literatur existieren einige Ansätze zur Erstellung von Occupancy Grids als kartenbasierte Fahrzeugumfeldmodelle. Typischerweise sind diese jedoch – vor allem aus Gründen der Einfachheit und Effizienz – maßgeschneidert auf ein konkretes Einsatzszenario und dadurch in der Wiederverwendbarkeit stark beschränkt.

Diese Arbeit stellt mit der OG-Bibliothek dagegen eine Bibliothek bereit, die den gleichberechtigten Einsatz beliebiger, unterschiedlicher Sensoren erlaubt und durch ihre Anpassbarkeit in ganz unterschiedlichen Szenarien zum Einsatz kommen kann. Dies wird durch die Realisierung einer Reihe neuer Ideen möglich; dazu gehören vor allem die anwendungsunabhängige und leicht anpassbare Architektur, das aus mehreren Ebenen bestehende Messwertgrid zur Integration heterogener Sensorik, die Speicherung eines Dynamikwerts je Zelle zur Filterung oder Visualisierung dynamischer Objekte, die Verwendung von Konfliktwer-

ten zum Umgang mit Widersprüchen und die effiziente Positionsmodellierung bestehend aus Zell-Position und In-Zell-Position.

Wie eine Anpassung der OG-Bibliothek an einen Anwendungsfall konkret aussieht und welch qualitativ hochwertigen Occupancy Grids mit der OG-Bibliothek erzeugt werden können, wurde anhand eines beispielhaften Anwendungsszenarios unter Verwendung eines Laser- und eines Radarsensors, der Dempster-Shafer-Theorie und des Frameworks ADTF demonstriert.

6.2 Ausblick

Nachdem in dieser Arbeit die OG-Bibliothek mithilfe des OGFilter in das Framework ADTF von Audi integriert wurde, ist für die Zukunft angedacht, die OG-Bibliothek zusätzlich auch in einem Framework vom BMW zu nutzen. Eine entsprechende Integration und Anpassung der OG-Bibliothek an einen exemplarischen Anwendungsfall sind bereits geplant.

Überhaupt wird aufgrund der vielversprechenden Evaluierungsergebnisse des vorherigen Kapitels erwartet, dass die OG-Bibliothek erfolgreich in nachfolgenden Arbeiten zur Erstellung automobiler Occupancy Grids eingesetzt werden kann – entweder durch deren Anpassung oder als Grundlage für eine speziellere Implementierung, die genau auf das jeweilige Anwendungsszenario zugeschnitten ist. In beiden Fällen erlaubt der Ansatz auch ein leichtes Vergleichen unterschiedlicher Möglichkeiten, um letztendlich ein Occupancy Grid erstellen zu können, das am besten die Anforderungen der Anwendung erfüllt.

Für die OG-Bibliothek sind einige Erweiterungen denkbar, die in dieser Arbeit noch nicht umgesetzt wurden, deren Umsetzung jedoch im Rahmen künftiger Arbeiten wünschenswert ist. Diese sollen zum Abschluss dieser Arbeit skizziert werden.

Die vorliegende Arbeit konzentriert sich schwerpunktmäßig auf die Kartierung, weshalb die Eigenposition davon unabhängig bestimmt wird. Wenngleich gute Ergebnisse auf diese Weise erzielt werden, sollte die Verwendung eines so genannten SLAM-Algorithmus' evaluiert werden, der bei der Lokalisierung nicht nur die entsprechenden, aktuellen Sensordaten nutzt, sondern auch das bisher erstellte Occupancy Grid einbezieht. Mögliche Ansätze sind in [52][4] beschrieben.

Die Auswirkungen eines „Vergessens-Faktors" für die Zellen, die nicht aktualisiert wurden [12][17], sollten ebenfalls untersucht werden. Die Werte aller nicht aktualisierten Zellen nähern sich dabei mit einer bestimmten Geschwindigkeit schrittweise wieder ihren initialen Werten. Der akkumulierte Lokalisierungsfehler wird unter einem gewissen Wert gehalten, da die Historie mithilfe des Faktors entsprechend beschränkt wird.

Je nach Anwendungsszenario ist es notwendig, die Ausführungsgeschwindigkeit weiter zu steigern. In Abschnitt 5.3.3 wurden bereits mehrere Möglichkeiten dafür aufgezeigt, die nach Bedarf umgesetzt werden sollten. Insbesondere die vorgeschlagene Beschränkung des ausgegebenen Occupancy Grids auf den tatsächlich verwendeten Bereich erscheint sinnvoll.

Interessant ist weiterhin die Implementierung einer Version der OG-Bibliothek, die den Grafikprozessor für parallelisierte Berechnungen nutzt [57][58]. [27][1] erzielen auf diese Weise deutliche Reduktionen der Laufzeit. Außerdem ist dadurch auch die effiziente Verwendung eines polaren Messwertgrids möglich. Dieses bietet sich für viele entfernungsmessende

Sensoren an, wie zum Beispiel Laser- und Radarsensoren, da diese ihre Daten messprinzipbedingt oftmals in polaren Koordinaten zurückliefern und dann eine polare Darstellung eine einfachere Kartierung ermöglicht.

Zudem wäre es auch wünschenswert, eine sensorspezifische Auflösung und Größe für das Messwertgrid wählen zu können. In der aktuellen Version der OG-Bibliothek sind beide Parameter für alle Sensoren gleich und entsprechen denen des zu erstellenden Occupancy Grids. Bei der vorgeschlagenen Erweiterung könnte man sich eine Faustregel zu Nutze machen, die besagt, dass Reichweite mal Auflösung für viele Sensoren gleich ist; das heißt, die Anzahl benötigter Gridzellen könnte dabei weiterhin konstant sein.

Weniger nützlich für die Aufgabenerfüllung des Assistenzsystems, aber dennoch hilfreich zu Forschungszwecken, kann ferner die Speicherung des Occupancy Grids als globale, zusammenhängende Karte sein. Ein möglicher Ansatz dazu findet sich in [34].

Literaturverzeichnis

[1] Adarve, J. D. ; Perrollaz, M. ; Makris, A. ; Laugier, C. : Computing Occupancy Grids From multiple Sensors Using Linear Opinion Pools. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on* IEEE, 2012, S. 4074–4079

[2] Aeberhard, M. ; Kaempchen, N. : High-Level Sensor Data Fusion Architecture for Vehicle Surround Environment Perception. In: *Intelligent Transportation (WIT), 8th International Workshop on*, 2011

[3] Aycard, O. ; Vu, T.-D. ; Baig, Q. : An occupancy grid based architecture for ADAS. In: *Advanced Microsystems for Automotive Applications 2010, Proceedings of the.* Springer, 2010, S. 199–210

[4] Bouzouraa, M. E. ; Hofmann, U. : Fusion of Occupancy Grid Mapping and Model Based Object Tracking for Driver Assistance Systems using Laser and Radar Sensors. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE* IEEE, 2010, S. 294–300

[5] Bouzouraa, M. E. ; Hofmann, U. : Fusion kartenbasierter Umfeldwahrnehmung mit modellbasierter Objektverfolgung für Fahrerassistenzsysteme. In: *7. Workshop Fahrerassistenzsysteme FAS 2011*, 2011, S. 115–123

[6] Bresenham, J. E.: Algorithm for computer control of a digital plotter. In: *IBM Systems journal* 4 (1965), Nr. 1, S. 25–30

[7] Burlet, J. ; Vu, T. D. ; Aycard, O. : Grid-based Localization and Online Mapping with Moving Object Detection and Tracking / Institut National de Recherche en Informatique et en Automatique (INRIA). Version: August 2007. `http://hal.inria.fr/inria-00167687/` (6276). – Forschungsbericht. – Zuletzt abgerufen am 11.08.2013

[8] Carlson, J. ; Murphy, R. R.: Use of Dempster-Shafer Conflict Metric to Detect Interpretation Inconsistency. In: *Uncertainty in Artificial Intelligence, Proceedings of the 21st Conference on*, 2005

[9] Carlson, J. ; Murphy, R. R. ; Christopher, S. ; Casper, J. : Conflict Metric as a Measure of Sensing Quality. In: *Robotics and Automation (ICRA), Proceedings of the 2005 IEEE International Conference on* IEEE, 2005, S. 2032–2039

[10] Coué, C. ; Pradalier, C. ; Laugier, C. : Bayesian Programming for Multi-Target Tracking: an Automotive Application. In: *Field and Service Robotics, Proceedings of the International Conference on*, 2003

[11] Coué, C. ; Pradalier, C. ; Laugier, C. ; Fraichard, T. ; Bessière, P. : Bayesian Occupancy Filtering for Multitarget Tracking: An Automotive Application. In: *The International Journal of Robotics Research* 25 (2006), Nr. 1, S. 19–30

[12] Darms, M. ; Komar, M. ; Lueke, S. : Map based Road Boundary Estimation. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE* IEEE, 2010, S. 609–614

[13] Darms, M. ; Winner, H. : Eine Systemarchitektur zur Fusion von Umfelddaten. In: *Aktive Sicherheit durch Fahrerassistenz, Proceedings of VDI-Tagung*, 2004

[14] Dempster, A. P.: A generalization of Bayesian inference. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1968), S. 205–247

[15] Dietmayer, K. : Evidenztheorie: Ein Vergleich zwischen Bayes- und Dempster-Shafer-Methoden. In: Beyerer, J. (Hrsg.) ; León, F. P. (Hrsg.) ; Sommer, K.-D. (Hrsg.): *Informationsfusion in der Mess- und Sensortechnik*. Universitätsverlag Karlsruhe, 2006, S. 39–49

[16] Effertz, J. : Hybride Sensorfusion zur Fahrzeug-Umfelderkennung und Fahrbereichsanalyse. In: *Workshop Fahrerassistenzsysteme FAS 2008*, 2008, S. 76–85

[17] Effertz, J. : *Autonome Fahrzeugführung in urbaner Umgebung durch Kombination objekt- und kartenbasierter Umfeldmodelle*, Technische Universität Carolo-Wilhelmina zu Braunschweig, Diss., 2009

[18] Elfes, A. : *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*, Carnegie Mellon University, Diss., 1989

[19] Elfes, A. : Using occupancy grids for mobile robot perception and navigation. In: *Computer* 22 (1989), Nr. 6, S. 46–57

[20] Garcia, R. ; Aycard, O. ; Vu, T.-D. ; Ahrholdt, M. : High Level Sensor Data Fusion for Automotive Applications using Occupancy Grids. In: *Control, Automation, Robotics and Vision 2008 (ICARCV), 10th International Conference on* IEEE, 2008, S. 530–535

[21] Grabe, B. ; Ike, T. ; Hötter, M. : Evidence Based Evaluation Method for Grid-based Environmental Representation. In: *Information Fusion 2009 (FUSION), 12th International Conference on*, 2009, S. 1234–1240

[22] Grewe, R. ; Hohm, A. ; Hegemann, S. ; Lueke, S. ; Winner, H. : Towards a Generic and Efficient Environment Model for ADAS. In: *Intelligent Vehicles Symposium (IV), 2012 IEEE* IEEE, 2012, S. 316–321

[23] Grewe, R. ; Komar, M. ; Hohm, A. ; Lueke, S. ; Winner, H. : Evaluation Method and Results for the Accuracy of an Automotive Occupancy Grid. In: *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*, 2012, S. 19–24

[24] Gâté, G. : *Reliable Perception of Highly Changing Environments - Implementations for Car-to-Pedestrian Collision Avoidance Systems*, Mines ParisTech, Diss., 2009

[25] Hammerschmidt, C. : "Das automatisierte Fahren ist eher eine Evolution". In: *VDI nachrichten* 15 (2013), S. 12

[26] Homm, F. ; Duda, A. ; Kaempchen, N. ; Waldmann, P. ; Ardelt, M. : Lidarbasierte Fahrstreifen-und Randbebauungserkennung mit Occupancy Grids für Spurhalte- und Spurwechselfunktionen. In: *Tagung Sicherheit durch Fahrerassistenz, München*, 2010

[27] Homm, F. ; Kaempchen, N. ; Ota, J. ; Burschka, D. : Efficient Occupancy Grid Computation on the GPU with Lidar and Radar for Road Boundary Detection. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE* IEEE, 2010, S. 1006–1013

[28] Koks, D. ; Challa, S. : An Introduction to Bayesian and Dempster-Shafer Data Fusion / Defence Science and Technology Organisation (Australia) Systems Sciences Laboratory. Version: 2003. `http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-1436.pdf` (DSTO-TR-1436). – Forschungsbericht. – Zuletzt abgerufen am 11.08.2013

[29] Konolige, K. : Improved Occupancy Grids for Map Building. In: *Autonomous Robots* 4 (1997), Nr. 4, S. 351–367

[30] Konrad, M. ; Dietmayer, K. : Occupancy Grid Mapping using the Dempster-Shafer-Theory. In: *Intelligent Transportation (WIT), 8th International Workshop on*, 2011, S. 167–172

[31] Konrad, M. ; Fuchs, M. ; Löhlein, O. ; Dietmayer, K. : Detektion und Tracking dynamischer Objekte in Occupancy Grids. In: *7. Workshop Fahrerassistenzsysteme FAS 2011*, 2011, S. 105–114

[32] Konrad, M. ; Nuss, D. ; Dietmayer, K. : Localization in Digital Maps for Road Course Estimation using Grid Maps. In: *Intelligent Vehicles Symposium (IV), 2012 IEEE* IEEE, 2012, S. 87–92

[33] Konrad, M. ; Szczot, M. ; Dietmayer, K. : Road Course Estimation in Occupancy Grids. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE* IEEE, 2010, S. 412–417

[34] Konrad, M. ; Szczot, M. ; Schüle, F. ; Dietmayer, K. : Generic Grid Mapping for Road Course Estimation. In: *Intelligent Vehicles Symposium (IV), 2011 IEEE* IEEE, 2011, S. 851–856

[35] Lundquist, C. ; Schön, T. B. ; Orguner, U. : Estimation of the Free Space in Front of a Moving Vehicle / Linköping University Electronic Press. Version: 2009. `http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-56197`. – Forschungsbericht. – Zuletzt abgerufen am 11.08.2013

[36] Moras, J. ; Cherfaoui, V. ; Bonnifait, P. : Credibilist Occupancy Grids for Vehicle Perception in Dynamic Environments. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, S. 84–89

[37] Moravec, H. ; Elfes, A. : High Resolution Maps from Wide Angle Sonar. In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference on* Bd. 2 IEEE, 1985, S. 116–121

[38] Moravec, H. P.: Sensor Fusion in Certainty Grids for Mobile Robots. In: *AI magazine* 9 (1988), Nr. 2, S. 61–74

[39] Murphy, R. R.: *Introduction to AI robotics*. MIT press, 2000

[40] Oriolo, G. ; Ulivi, G. ; Vendittelli, M. : Fuzzy maps: a new tool for mobile robot perception and planning. In: *Journal of Robotic Systems* 14 (1997), Nr. 3, S. 179–197

[41] Pagac, D. ; Nebot, E. M. ; Durrant-Whyte, H. : An Evidential Approach to Map-Building for Autonomous Vehicles. In: *Robotics and Automation, IEEE Transactions on* 14 (1998), Nr. 4, S. 623–629

[42] Richter, E. ; Lindner, P. ; Wanielik, G. ; Takagi, K. ; Isogai, A. : Advanced Occupancy Grid Techniques for Lidar based Object Detection and Tracking. In: *Intelligent Transportation Systems 2009 (ITSC), 12th International IEEE Conference on* IEEE, 2009, S. 450–454

[43] Schubert, R. ; Richter, E. ; Wanielik, G. : Comparison and Evaluation of Advanced Motion Models for Vehicle Tracking. In: *Information Fusion, 2008. 11th International Conference on* IEEE, 2008, S. 730–735

[44] Sentz, K. ; Ferson, S. : *Combination of Evidence in Dempster-Shafer Theory*. 2002

[45] Shafer, G. : *A mathematical theory of evidence*. Bd. 1. Princeton University Press, 1976

[46] Stepan, P. ; Kulich, M. ; Preucil, L. : Robust Data Fusion With Occupancy Grid. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 35 (2005), Nr. 1, S. 106–115

[47] Stüker, D. : *Heterogene Sensordatenfusion zur robusten Objektverfolgung im automobilen Straßenverkehr*, Carl-von-Ossietzky-Universität Oldenburg, Diss., 2003

[48] Suganuma, N. ; Matsui, T. : Robust environment perception based on occupancy grid maps for autonomous vehicle. In: *SICE Annual Conference 2010, Proceedings of* IEEE, 2010, S. 2354–2357

[49] Szczot, M. ; Serfling, M. ; Löhlein, O. ; Schüle, F. ; Konrad, M. ; Dietmayer, K. : Global Positioning Using a Digital Map And an Imaging Radar Sensor. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE* IEEE, 2010, S. 406–411

[50] Thrun, S. ; Burgard, W. ; D.Fox: *Probabilistic Robotics*. The MIT Press, 2005

[51] Thrun, S. : Learning Occupancy Grid Maps with Forward Sensor Models. In: *Autonomous Robots* 15 (2003), Nr. 2, S. 111–127

[52] Vu, T.-D. ; Aycard, O. ; Appenrodt, N. : Online Localization and Mapping with Moving Object Tracking in Dynamic Outdoor Environments. In: *Intelligent Vehicles Symposium (IV), 2007 IEEE* IEEE, 2007, S. 190–195

[53] Vu, T.-D. ; Burlet, J. ; Aycard, O. : Grid-based Localization and Online Mapping with Moving Object Detection and Tracking: new results. In: *Intelligent Vehicles Symposium (IV), 2008 IEEE* IEEE, 2008, S. 684–689

[54] Weiss, T. ; Schiele, B. ; Dietmayer, K. : Robust Driving Path Detection in Urban and Highway Scenarios Using a Laser Scanner and Online Occupancy Grids. In: *Intelligent Vehicles Symposium (IV), 2007 IEEE*, 2007, S. 184–189

[55] Winner, H. ; Hakuli, S. ; Wolf, G. : *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort.* 2., korrigierte Auflage. Vieweg+Teubner, 2012

[56] Wüst, C. : Automobile – Fahren ohne Fahrer. In: *Der Spiegel* 5 (2013), S. 98–102

[57] Yguel, M. ; Aycard, O. ; Laugier, C. : Efficient GPU-based Construction of Occupancy Grids Using several Laser Range-finders. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, S. 105–110

[58] Yguel, M. ; Aycard, O. ; Laugier, C. : *Efficient GPU-based Construction of Occupancy Grids Using several Laser Range-finders.* Online. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.220.3201&rep=rep1&type=pdf`. Version: 2006. – Zuletzt abgerufen am 04.04.2013

Anhang

Der Anhang besteht aus folgender DVD:

Auf dieser DVD befinden sich

- der Quellcode der OG-Bibliothek und die zugehörige Dokumentation,
- der Quellcode des OGFilter und die zugehörige Dokumentation sowie
- Aufzeichnungen einer Auswahl durchgeführter Simulationen.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Masterarbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle wörtlich oder sinngemäß übernommenen Ausführungen wurden als solche gekennzeichnet. Weiterhin erkläre ich, dass ich diese Arbeit in gleicher oder ähnlicher Form nicht bereits einer anderen Prüfungsbehörde vorgelegt habe.

Passau, 13. August 2013

. . . . . .
Christian Pieringer