Universität Bremen

Zentrum für Technomathematik

Adaptive Bewegungsplanung für taktische Entscheidungen autonomer Fahrzeuge im urbanen Umfeld

Der generische Hybrid A*

Dissertation

zur Erlangung eines Doktorgrades der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik an der Universität Bremen

vorgelegt von

Andreas Folkers

1. Gutachter: Prof. Dr. Christof Büskens, Universität Bremen
2. Gutachter: Prof. Dr.-Ing. Rick Voßwinkel, Hochschule Zwickau

Datum des Promotionskolloquiums: 10. Februar 2023

Vorwort

Diese Dissertationsschrift entstand während meiner Arbeit innerhalb des Forschungsprojektes OPA^{3}L in der Arbeitsgruppe für Optimierung und Optimale Steuerung am Zentrum für Technomathematik der Universität Bremen. Das Projekt OPA^{3}L (Zuwendungsnummer 50 NA 1909) wird seit März 2019 vom Deutschen Zentrum für Luft- und Raumfahrt durch die Mittel des (damaligen) Bundesministeriums für Wirtschaft und Energie finanziert.

Ich möchte einen besonderen Dank an die vielen Menschen aussprechen, die mich in den vergangenen Jahren begleitet und auf verschiedenste Arten unterstützt haben:

An Prof. Dr. Christof Büskens, für die einmalige Chance, eine Doktorarbeit in einem so spannenden und aktuellen Themenfeld wie der Algorithmenentwicklung für autonome Fahrzeuge schreiben zu können. Ich bin sehr dankbar, stets die Freiheiten gehabt zu haben, meine eigenen Ideen entwickeln und erproben zu können. Zudem schätze ich sehr, dass ich nicht nur gefördert, sondern stets auch mit neuen Aufgaben gefordert wurde, wodurch ich mich auch abseits der wissenschaftlichen Arbeit weiterentwickeln konnte.

An Prof. Dr.-Ing. Rick Voßwinkel, für die stets spannenden fachlichen Diskussionen sowohl während seiner Zeit im Projekt OPA^{3}L als auch nach seinem Ruf als Professor zur Hochschule Zwickau. Insbesondere für die umfangreichen Anmerkungen und den Austausch in der abschließenden Phase dieser Arbeit bin ich zu besonderem Dank verpflichtet.

An Dr. Matthias Knauer, dessen enthusiastische Art, Wissenschaft zu betreiben und zu vermitteln, mich bis heute inspiriert. Darüber hinaus hat er mich vor allem in den letzten zwei Jahren in unzähligen Gesprächen mit Anmerkungen und Fragen zu Inhalt und Darstellung dieser Arbeit motiviert und unterstützt.

An alle Freunde und Kollegen aus den verschiedensten Gruppen und Projekten, die mich seit 2015 auf meiner Reise durch die Welt des autonomen Fahrens begleiten. Hier möchte ich insbesondere Matthias Rick herausheben, mit dem ich während dieser ganzen Zeit Seite an Seite stehen und gemeinsam so viel lernen und Unvergessliches erleben durfte.

An alle Freunde und Kollegen aus der Arbeitsgruppe für Optimierung und Optimale Steuerung sowie von Topas, für die einzigartige und offene Arbeitsatmosphäre.

Dies gilt in besonderem Maße auch für Dr.-Ing. Mitja Echim, dessen Motivation, mit unseren Methoden und Ideen stets neue Wege zu gehen, mich immer wieder beeindruckt.

An alle weiteren Menschen, die mir in den letzten Wochen und Monaten mit vielen kleinen oder großen Hinweisen zu dieser Arbeit weitergeholfen haben: Lennart Evers, Helmrich und Tamara Folkers, Maria Höffmann, Malin Lachmann, Christian Meerpohl, Dr. Shruti Patel, Dr.-Ing. Carsten Rachuy, Margareta Runge sowie Bente Zippel.

An meine Familie, die mich auf meinem Weg stets bedingungslos unterstützt hat. Insbesondere an meine Frau Lisa, die auf so viele gemeinsame Sonntagvormittage verzichtet hat, um mir den nötigen Freiraum für diese Dissertation zu geben. Und an meine Tochter Carlotta, die mir mit jedem Lachen vor Augen führt, wie wertvoll das Leben ist.

Kurzzusammenfassung

Im Zentrum dieser Arbeit steht die Präsentation des generischen Hybrid A*-Algorithmus, welcher ein graphenbasiertes Suchverfahren zur Bewegungsplanung eines beweglichen Agenten darstellt. Mit den dabei vorgestellten Methoden wird der originäre Hybrid A*-Algorithmus weiterentwickelt, welcher ursprünglich als Pfadplaner für ein autonomes Fahrzeug eingeführt wurde. Die wesentlichen Modifikationen betreffen dabei zum einen die Berücksichtigung von statischen Umgebungsinformationen, welche hier durch die Berechnung eines Freibereichspolygons in abstrahierter Form dargestellt werden. Dies erlaubt nicht nur eine generische Anwendung des eigentlichen Suchalgorithmus auf eine beliebige Datengrundlage, sondern führt insbesondere für Daten aus Punktwolken auch schnell zu einem spürbaren Effizienzvorteil. Zusammen mit allen dynamischen Umgebungskomponenten wird das Freibereichspolygon zum anderen genutzt, um ein verallgemeinertes Voronoi-Potentialfeld zu definieren. Innerhalb des Suchverfahrens stellt dieses eine kontinuierliche Bewertung der zeitabhängigen Konfiguration des betrachteten Agenten bezüglich dessen Umwelt dar und setzt zeitgleich eine binäre Kollisionsüberprüfung um.
Einen wesentlichen Bestandteil des eingeführten Potentialfeldes repräsentieren dynamische Voronoi-Pfade. Diese werden als Teilmenge der Kanten von Voronoi-Diagrammen innerhalb eines Freibereichspolygons definiert. Durch die Entwicklung eines Algorithmus zur lokalen Aktualisierung eines statischen Voronoi-Diagramms können dabei zudem auch alle dynamischen Umgebungskomponenten effizient berücksichtigt werden. Zusätzlich wird gezeigt, wie die Voronoi-Pfade auch außerhalb des Potentialfeldes zur generischen Analyse der Umgebung des Agenten eingesetzt werden, um zum Beispiel Zielkonfigurationen für das Suchverfahren zu berechnen.
Die numerische Evaluierung des generischen Hybrid A*-Algorithmus wird im Kontext des autonomen Fahrens in urbanen Situationen durchgeführt. Dabei werden zum einen simulierte Szenarien betrachtet, bei denen ein besonderer Schwerpunkt auf den Verlauf des Suchverfahrens, die Qualität der resultierenden Lösungen sowie die benötigte Zeit für deren Berechnung gelegt wird. Zum anderen wird gezeigt, wie das Verfahren als Grundlage für die Bewegungsplanung und kurzfristige Entscheidungsfindung eines vollständig autonomen Fahrzeugs eingesetzt werden kann. Dies wird anhand von zwei autonomen Fahrten eines realen Forschungsfahrzeuges unter urbanen und dynamischen Rahmenbedingungen demonstriert.

Abstract

The focus of this work is the presentation of the generic hybrid-state A* algorithm, which represents a novel graph-based search method for motion planning of a moving agent. The presented methods enhance the original hybrid-state A* algorithm, which was originally introduced as a path planner for an autonomous vehicle. The main modifications concern on the one hand how static environmental information is taken into account. Here, they are represented in an abstract form through the computation of a free-space polygon. This not only allows a generic application of the actual search algorithm to an arbitrary data basis, but also quickly leads to a noticeable efficiency advantage, especially for data from point clouds. Combined with all dynamic environment components, the free-space polygon is used to define a generalized Voronoi potential field on the other hand. Within the search algorithm, this is used for a continuous evaluation of the time-dependent configuration of the considered agent with respect to its environment and simultaneously implements a binary check for collisions.

Dynamic Voronoi paths represent an essential component of the introduced potential field. They are defined as a subset of the edges of Voronoi diagrams inside a free-space polygon. Moreover, by developing an algorithm for local updates of a static Voronoi diagram, all dynamic environment components can be efficiently considered in this process as well. Furthermore, it is shown how the Voronoi paths are also applied beyond the potential field to generically analyze the agent's environment to, e. g., compute target configurations for the search process.

The numerical evaluation of the generic hybrid-state A* algorithm is performed in the context of autonomous driving in urban situations. On the one hand, simulated scenarios are considered, with a particular focus on the performance of the search process, the quality of the resulting solutions, and the time required to compute them. On the other hand, it is shown how the methods presented can be used as a basis for motion planning and short-term decision making of a complete system for autonomous driving. This is demonstrated by means of two autonomous drives of an actual research vehicle in urban and dynamic conditions.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Akronymverzeichnis

Akronym	Bezeichnung
5G	5. Mobilfunkgeneration
ABS	Antiblockiersystem
BMWi	Bundesministerium für Wirtschaft und Energie
BMWK	Bundesministerium für Wirtschaft und Klimaschutz
CAN	Controller Area Network
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
PROMETHEUS	Programme for European Traffic with Highest Efficiency and Unprecedented Safety
Radar	Radio Detection and Ranging
RTK	Real Time Kinematic
SAE	Society of Automotive Engineers
V2I	Vehicle-to-Infrastructure communication
V2V	Vehicle-to-Vehicle communication
V2X	Vehicle-to-X communication
VaMP	VaMoRs-PKW

Kapitel 1

Einleitung

Die gesellschaftliche Debatte um autonome, also selbstfahrende Fahrzeuge wird zunehmend intensiv und auch emotional geführt. Ängste wie der Verlust der persönlichen Freiheit (hinter dem Steuer) oder ein Misstrauen in die Technologie selbst stehen dabei zahlreichen erwarteten gesellschaftlichen und technologischen Vorteilen gegenüber [36]. Im Jahr 2019 wurden alleine in Deutschland mehr als 2,69 Millionen Verkehrsunfälle registriert. Infolge dieser sind 65 244 Menschen schwer verletzt worden und 3046 Menschen verstorben. In 94,1 % der Fälle ist die Unfallursache auf menschliches Fehlverhalten – zum Beispiel unangepasste Geschwindigkeit oder Missachtung von Vorfahrtsregeln – zurückzuführen [85]. Ein ähnliches Bild ergibt sich in den Vereinigten Staaten von Amerika, wo der Fahrzeugführer für 94,0 % der Verkehrsunfälle verantwortlich gemacht wird [89]. Es ist zu erwarten, dass dieser Anteil durch eine verstärkte Autonomisierung des Verkehrs deutlich gesenkt werden kann [103, 108]. Durch die Abgabe von Handlungsverantwortung im Straßenverkehr können sich zudem weitere positive und individuelle Effekte ergeben, wie die Reduzierung von fahrbedingtem Stress (und dessen messbare gesundheitliche Auswirkungen [45]) oder eine effektivere Nutzung von Transportzeiten [44]. Weiterhin ergäben sich Mobilitätsangebote für Menschen, die ansonsten nicht fahren können oder wollen [73].
Aus gesellschaftlicher Sicht haben autonom agierende Fahrzeuge das Potential,

Abbildung 1.1: Layout des Finalparcours [78] und das Siegerfahrzeug *Boss* [100] der DARPA Urban Challenge 2007

die Implementierung von völlig neuen Mobilitätsprinzipien zu ermöglichen. Als wichtiges Beispiel sind hier Mobility on Demand-Konzepte zu nennen, bei denen sich Fahrzeuge zum Beispiel bei Bedarf selbstständig einem Fahrgast bereitstellen und diesen gegebenenfalls transportieren können [10, 60, 73]. Unter anderem könnte dies zu einer wichtigen Entlastung von Städten und Umwelt durch weniger Individualfahrzeuge und sinkende CO_2-Emissionen führen [44].

Der Grad der Autonomiefähigkeit eines Fahrzeuges wird typischerweise gemäß der Norm SAE J3016 in sechs Kategorien unterteilt, wie in Tabelle 1.1 dargestellt. Assistenz- und teilautomatisierte Systeme, bei denen der Fahrer weiterhin die Kontrolle über das Fahrzeug behält, wurden in den letzten Jahrzehnten vielfach sehr erfolgreich bis zu Marktreife entwickelt und erhöhen bereits merklich die Sicherheit und den Komfort des Fahrens. Die Auswahl reicht heute in vielen Serienfahrzeugen vom Antiblockiersystem (ABS) aus den 1970er Jahren (SAE-Stufe 0) über Abstands- und Spurhaltesystemen (SAE-Stufe 1) bis hin zu Einparkassistenten (SAE-Stufe 2) [9]. Aber auch vollständig selbstfahrende Fahrzeuge werden (für bestimmte Anwendungsgebiete) immer wieder prototypisch entwickelt, um die genannten Visionen und Potentiale autonomen Fahrens Realität werden zu lassen. Als eine der ersten wegweisenden Arbeiten ist hier vor allem das Projekt PROMETHEUS zu nennen, welches 1994 in einer 1000 km langen Demonstration des Pkws VaMP resultierte. Dieses konnte selbstständig in einem dreispurigen Autobahnverkehr manövrieren, andere Verkehrsteilnehmer erfassen und Spurwechselentscheidungen treffen und durchführen [23].

Zehn Jahre später zeigten die DARPA Wettbewerbe neue technische Fortschritte und Möglichkeiten auf. In Rahmen der DARPA Grand Challenge 2005 schafften es fünf Fahrzeuge, einen 212 km langen Wüstenparcours vollständig ohne menschlichen Eingriff zu bewältigen [17]. Einen wesentlichen Meilenstein setzte jedoch vor allem die DARPA Urban Challenge 2007, in welcher der zu absolvierende Parcours in

Tabelle 1.1: Stufen der Automatisierung von straßengebundenen Kraftfahrzeugen gemäß der Norm SAE J3016, nach [16]

SAE-Stufe	Beschreibung	Fahrzeug-führung	Umgebungs-beobachtung	Rückfall-ebene
0 – **Keine Automation**	Der Fahrer fährt eigenständig, auch wenn unterstützende Systeme (zum Beispiel ABS) vorhanden sind.	Fahrer	Fahrer	keine
1 – **Assistenzsysteme**	Fahrerassistenzsysteme helfen bei der Fahrzeugbedienung bei Längs- oder Querführung (zum Beispiel ACC).	Fahrer und System	Fahrer	Fahrer
2 – **Teilautomatisierung**	Ein oder mehrere Fahrerassistenzsysteme helfen bei der Fahrzeugbedienung bei Längs- und gleichzeitiger Querführung.	System	Fahrer	Fahrer
3 – **Bedingte Automatisierung**	Autonomes Fahren mit der Erwartung, dass der Fahrer auf Anforderung zum Eingreifen reagieren muss.	System	System	Fahrer
4 – **Hochautomatisierung**	Automatisierte Führung des Fahrzeugs ohne die Erwartung, dass der Fahrer auf Anforderung zum Eingreifen reagiert. Ohne menschliche Reaktion steuert das Fahrzeug weiterhin autonom.	System	System	System
5 – **Vollautomatisierung (Autonom)**	Vollständig autonomes Fahren, bei dem die dynamische Fahraufgabe unter jeder Fahrbahn- und Umgebungsbedingung, welche auch von einem menschlichen Fahrer beherrscht wird, durchgeführt wird.	System	System	System

ein simuliertes urbanes Umfeld eingebettet wurde, vergleiche Abbildung 1.1 links. Die teilnehmenden Fahrzeuge mussten dabei unter anderem korrektes Verhalten bei Kreuzungen oder fehlenden Straßenmarkierungen zeigen, sich in fließenden Verkehr einfädeln, parken oder eigenständig Alternativpläne bei blockierten Straßen entwickeln. Sechs Teams konnten den Finalparcours erfolgreich beenden [18].
Auch wenn mit den Erfolgen aus dieser Zeit die Grundlage für autonomes urbanes Fahren gelegt wurde, waren die eingesetzten Prototypen noch weit von einem Serienfahrzeug entfernt. So wurden häufig eine Vielzahl von vergleichsweise teuren Sensoren verwendet und diese zum Beispiel sehr pragmatisch am Fahrzeugdach befestigt (siehe auch Abbildung 1.1 rechts). Um die Qualität und Seriennähe von autonomen Fahrfunktionen zu verbessern, werden bis heute große Anstrengungen unternommen – sowohl von klassischen Automobilherstellern [8, 110] als auch von neuen Akteuren wie der Google-Tochter Waymo [106] oder Tesla [97]. Während hier bereits erste partiell automatisierte Seriensysteme bereitstehen – wie ein Stauassistent [9] – ergeben sich zeitgleich auch immer neue technische Möglichkeiten zur Weiterentwicklung. Neben steigender Rechenleistung oder verbesserter Sensorik [107] wird dabei auch großes Potential in der Vernetzung autonomer Fahrzeuge durch den neuen Mobilfunkstandard 5G erwartet [1]. Parallel zum technischen Fortschritt werden zudem die notwendigen rechtlichen Rahmenbedingungen für den Einsatz selbstfahrender Fahrzeuge stetig erweitert. Zuletzt hat Deutschland, als erstes Land weltweit, im Jahr 2021 eine gesetzliche, landesweite Gesetzesgrundlage zum Regelbetrieb fahrerloser Fahrzeuge der SAE-Stufe 4 für bestimmte Einsatzszenarien geschaffen [15].

Trotz dieser dynamischen Entwicklungen ist ein vollständig autonomes, universal einsetzbares Fahrzeug jedoch bis heute nicht verfügbar [107]. Während weiterhin offene rechtliche [39] und ethische [64] Fragen diskutiert werden, ist auch eine technische und algorithmische Weiterentwicklung permanent notwendig, damit der sichere Betrieb in möglichst jeder Situation garantiert ist. Um dieses Ziel zu erreichen, werden immer wieder neue Forschungs-, Entwicklungs- und Denkansätze benötigt.

1.1 Autonomieebenen für autonomes Fahren

Die Intelligenz in einem autonomen Fahrzeug wird klassischerweise durch die Planung auf verschiedenen Abstraktions- und Zeitebenen verwirklicht, vergleiche Abbildung 1.2. Auf langfristiger Ebene findet dabei in der Regel eine Routenplanung statt, welche zum Beispiel die Strecke des Fahrzeuges in einem gegebenen Straßennetzwerk bestimmt. Auf kurzfristiger Ebene muss das System dagegen über konkrete Fahrmanöver entscheiden, die zur Erfüllung der langfristigen Ziele beitragen. Dabei kann gegebenenfalls eine vorgelagerte Pfadplanung bezüglich der näheren Umgebung des Fahrzeugs unterstützen [53]. Zuletzt wird eine Bewegungsplanung durchgeführt, die in eine konkrete Trajektorie zum Ausführen dieser Entscheidung resultiert. Die Aufgabe der Regelung ist schließlich, die bestmögliche Umsetzung dieser Vorgabe

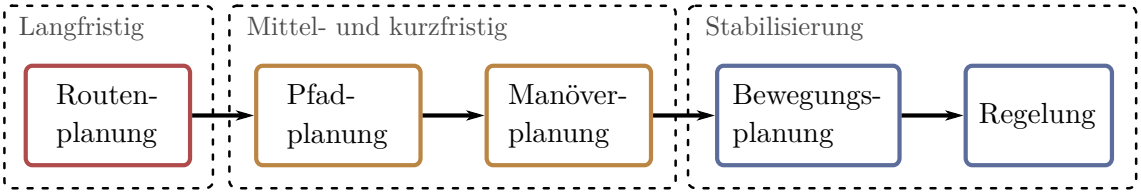


Abbildung 1.2: Klassische Planungsebenen des autonomen Fahrens, nach [53, 72, 98]

sicherzustellen.
Besonders die Pfad-, Manöver- und Bewegungsplanung haben einen starken Einfluss auf das Verhalten des Fahrzeuges bezüglich seiner unmittelbaren Umgebung und die Fähigkeit, ein sicheres und vorausschauendes Fahrverhalten zu ermöglichen. Im Folgenden wird daher ein kurzer Überblick über diese drei Planungsebenen gegeben, wobei sich die Begriffsbildung an [53] orientiert. Einen erweiterten Überblick geben [4, 53, 72].

1.1.1 Pfadplanung

Die Aufgabe der Pfadplanung entspricht der Suche nach einer geometrischen Bahn des Fahrzeugs zu einer festgelegten Endkonfiguration. Dabei beschreibt eine Konfiguration einen Satz von unabhängigen Koordinaten welche eindeutig Position und Orientierung des Fahrzeugs in einem festgelegten Koordinatensystem definieren. Der Konfigurationsraum entspricht dann dem Raum aller möglichen Koordinaten und das Ergebnis der Pfadplanung einer Sequenz von dessen Elementen. Das Suchergebnis gilt als zulässig, wenn keine enthaltene Konfiguration eine Kollision mit einem (bekannten) Hindernis impliziert. Je nach Anwendung müssen Transformationen zwischen zwei Konfigurationen zudem einem dynamischen beziehungsweise kinematischen Fahrzeugmodell genügen.

Es gibt zahlreiche Ansätze zum Lösen des Pfadplanungsproblems. Auf analytischer Ebene sind hier vor allem die Ansätze von Dubins [27] oder Reeds und Shepp [79] zu nennen, welche jeweils die Berechnung kinematisch zulässiger Pfade ermöglichen, bei denen jedoch keine Hindernisse berücksichtigt werden. Um Letzteres zu ermöglichen, werden in der Regel numerische Verfahren wie Rapidly-Exploring Random Trees [58] oder Suchalgorithmen basierend auf Bewegungsprimitiven [62] eingesetzt. Beide reduzieren den Suchraum durch entsprechende Datenstrukturen und erreichen so eine hohe Effizienz; es können jedoch Probleme durch stochastische Schwankungen der Ergebnisse im ersten oder Oszillationen im zweiten Fall auftreten [53]. Als generelle Alternative entwickeln sich zuletzt vermehrt auch datenbasierte Ansätze, zum Beispiel durch (Deep) Reinforcement Learning [77]. Hier stellt jedoch die Über-

tragbarkeit auf reale Situationen weiterhin ein Hauptproblem existierender Ansätze dar [5].

Allgemein kann die Robustheit von Pfadplanungsalgorithmen unter anderem durch Potentialfelder verbessert werden, welche zum Beispiel die Minimierung von Kollisionswahrscheinlichkeiten zum Ziel haben. Insgesamt gibt es dabei sehr unterschiedliche (teilweise anwendungsspezifische) Modellierungsansätze [2, 50, 109], wobei insbesondere das Voronoi-Potential hervorzuheben ist [25]. Dieses wurde zusammen mit dem Hybrid A*-Algorithmus im Rahmen der DARPA Urban Challenge 2007 entwickelt und ermöglicht eine intelligente Navigation in strukturierten sowie unstrukturierten Umgebungen (zum Beispiel mit und ohne explizite Spurinformationen). Um echtzeitfähige Laufzeiten bei der Berechnung des zugrunde liegenden Voronoi-Feldes zu erreichen, ist jedoch in der Regel eine Vorverarbeitung der Umgebungsdaten notwendig [59]. Zudem gibt es bisher keine Ansätze, um ebenfalls dynamische Informationen zu berücksichtigen [53].

1.1.2 Manöverplanung

Mithilfe der im vorigen Schritt geplanten Pfade wird eine konkrete Entscheidung bezüglich eines Fahrmanövers getroffen. Dabei werden in der Regel möglichst viele Informationen über die Umgebung des Fahrzeuges berücksichtigt, um zum Beispiel Verhalten von anderen Verkehrsteilnehmern zuverlässig zu interpretieren. Mögliche Entscheidungen können dann abstrakte Manöver wie *Spurhalten*, *Spurwechsel*, *Einparken* oder dynamische Restriktionen wie eine Zielgeschwindigkeit sein.

Ein genereller Ansatz zur Beschreibung einer vorliegenden Situation sind ontologiebasierte Algorithmen, welche die Beziehungen und Hierarchien zwischen Objekten formalisieren. Basierend auf festgelegten Regeln können daraus schließlich Handlungsintentionen abgeleitet werden [6]. Alternativ eignen sich auch hier datenbasierte Methoden, bei denen eine intelligente Autonomie zum Beispiel durch (Deep) Reinforcement Learning trainiert wird, um seine Umgebung interpretieren und Entscheidungen fällen zu können [3, 46]. In der Praxis generalisieren beide Ansätze jedoch nur eingeschränkt auf unbekannte Szenarien [53], weshalb sich viele Arbeiten auf starke Vereinfachungen wie (gerade) Autobahn- oder Straßenabschnitte beschränken. Speziell in diesem Rahmen werden häufig Spurwechselentscheidungen betrachtet, welche meist auf einfachen Regeln und Heuristiken bezüglich Geschwindigkeiten und der geometrischen Konstellation der Fahrzeuge basieren [7, 47, 52, 69]. Eine höhere Komplexität ergibt sich bei der Behandlung von (unregulierten) Kreuzungen. Um hier Entscheidungen zu treffen, die ein sicheres Passieren erlauben, wird häufig eine genauere Analyse der Intentionen und Dynamiken von anderen beteiligten Verkehrsteilnehmern benötigt [21, 57, 91].

Werden keine speziellen Anwendungsfälle betrachtet, so gehen entsprechende Einschränkungen an Straßengeometrien und Vorwissen über mögliche Verhaltensweisen

Abbildung 1.3: Beispiel einer komplexeren Situation in einem suburbanen Umfeld

Dritter verloren. Eine entsprechende Situation ist beispielhaft in Abbildung 1.3 gezeigt. Hier muss ein intelligentes Fahrzeug die Bewegung des Radfahrers antizipieren, um dies letztlich als Grundlage für die eigene Manöverentscheidung zu nutzen. Ein Hilfsmittel, um generelle Situationen in solchen (sub-)urbanen Kontexten zuverlässig zu interpretieren, ist die Evaluation von Risikomaßen wie die Time to Collision-Metrik [105]. Alternativ kann auch eine vorgelagerte oder simultane Bewegungsplanung durchgeführt werden, deren Ergebnisse dann als präzise Grundlage zur Analyse von Umsetzbarkeit und Sicherheit von Manöveroptionen genutzt werden können. Die Adaptionskapazität der Entscheidungsfindung ergibt sich dann direkt aus der Abstraktionsfähigkeit dieser Planungsebene, vergleiche zum Beispiel [12, 41, 53, 55].

1.1.3 Bewegungsplanung

Anders als bei der Pfadplanung wird bei der Bewegungsplanung (oder auch Trajektorienplanung) der Zustand des Fahrzeugs betrachtet, welcher zusätzlich Informationen über die Dynamik und die Zeit enthält (zum Beispiel im Kontext eines entsprechenden Fahrzeugmodells). Abhängig von der Anwendung und dem Modell können hier neben der Konfiguration beispielsweise verschiedene Geschwindigkeitsgrößen berücksichtigt werden. Das Ziel der Bewegungsplanung ist dann, im zugehörigen Zustandsraum eine zeitabhängige Abfolge von Zuständen zu generieren, die zum einen bezüglich Hindernissen zulässig sind und zum anderen dem Fahrzeugmodell genügen. Das Ergebnis der Bewegungsplanung wird auch Trajektorie genannt. Im Kontrast zur Pfadplanung eignet sich die Berechnung von Trajektorien aufgrund ihrer Zeitkomponente besonders auch für die Berücksichtigung dynamischer Objekte und führt so zu einem genaueren Ergebnis.

Als Basis für eine Bewegungsplanung kann – je nach Anwendung – zum Beispiel die Identifikation von Parametern geometrischer Kurven verwendet werden. Als Modellierungsgrundlage werden dabei zum Beispiel polynomiale Funktionen [80, 104] oder Splines [68] eingesetzt. Aufgrund eines fehlenden Dynamikmodells lassen sich diese Ansätze jedoch in der Regel oft nur für bestimmte Szenarien einsetzen. Dem

gegenüber stehen numerische Methoden der Optimalen Steuerung, welche sowohl dynamische Fahrzeugmodelle als auch umfassende Beschränkungen direkt in der Trajektorienberechnung berücksichtigen können [61, 90]. Dies gilt grundsätzlich auch für dynamische Hindernisse [32]. Obwohl es sehr effiziente Löser für die zugrunde liegenden Optimierungsprobleme gibt [48, 54], hängt die Robustheit und Echtzeitfähigkeit der Verfahren letztlich in starkem Maße von einer zulässigen und hinreichend guten Startschätzung ab [88].
Grundsätzlich lassen sich häufig auch bekannte Suchalgorithmen aus der Pfadplanung auf dynamische Kontexte erweitern [53], wodurch sich eine ganze Reihe von Algorithmen zur Trajektorienoptimierung in entsprechend reduzierten Suchräumen ergibt, vergleiche Abschnitt 1.1.1.

1.2 Das Forschungsprojekt OPA^{3}L

Seit 2019 fördert das damalige deutsche Bundesministerium für Wirtschaft und Energie (BMWi)* das Forschungsprojekt OPA^{3}L (***Op**timal **A**ssistierte, hoch **A**utomatisierte, **A**utonome und kooperative Fahrzeugnavigation und **L**okalisation*). Dessen Kernziel ist die Implementierung eines einheitlichen Autonomiesystems für autonome Fahrfunktionen in einem urbanen Umfeld. Als ein wesentlicher Anwendungsfall ist die prototypische Umsetzung eines Vehicle on Demand-Konzeptes in einem exemplarischen Vorstadtgebiet definiert. Daher ist vor allem die (Weiter-) Entwicklung von Autonomiefunktionen für den Einsatz in einem solchen Szenario ein Hauptziel von OPA^{3}L. Zudem sollen unter anderem neue Technologien wie der Galileo Positionierungsdienst der Europäischen Union oder Systeme zur Kommunikation zwischen Fahrzeugen untereinander (V2V) bzw. allgemein von Fahrzeugen mit der Infrastruktur (V2I) untersucht werden. Die Absicherung der entwickelten Algorithmen erfolgt über ein dreistufiges Testsystem, bestehend aus Simulation, Miniaturfahrzeugen und schließlich dem tatsächlichen Forschungsfahrzeug, ein modifizierter VW Passat GTE, vergleiche Abbildung 1.4.

Die Verwendung dieser unterschiedlichen Plattformen erlaubt auf der einen Seite ein schnelles und einfaches Testen, um so frühzeitig Risiken zu identifizieren. Auf der anderen Seite schafft der Ansatz, auf teilweise sehr unterschiedlichen Plattformen dieselben Algorithmen zu evaluieren, zusätzliche Anforderungen an deren Adaptivität. Diese Dimension in der Komplexität des Forschungsziels verläuft annähernd orthogonal zur vorgenannten Aufgabe, autonome Fahrfunktionen für möglichst allgemeine Situationen im (urbanen) Straßenverkehr zu entwickeln. Ein System, das diesen Anforderungen gerecht wird, benötigt unter anderem besonders adaptionsfähige Algorithmen sowie deren möglichst effiziente Interaktion miteinander.

*Seit 8. Dezember 2021: Bundesministerium für Wirtschaft und Klimaschutz (BMWK)

(a) CARLA-Simulator [26]

(b) Miniaturfahrzeug im Maßstab 1:8, aus [33]

(c) Forschungsfahrzeug mit Sicherheitsfahrer, aus [33]

Abbildung 1.4: Testplattformen in OPA³L. Eine Beschreibung der Sensorik ist in Tabelle 1.2 dargestellt.

Um Letzteres zu ermöglichen, folgt die funktionale Softwarearchitektur von OPA³L bekannten Ansätzen wie [87]. Sie ist in Abbildung 1.5 vereinfacht und im Kontext des Forschungsfahrzeuges dargestellt; Unterschiede zu den weiteren Plattformen bestehen allerdings nur in den Schnittstellen zwischen Software und Hardware der (simulierten) Fahrzeuge sowie in den zur Verfügung stehenden Daten. Für den Passat werden diese im Wesentlichen durch die Sensorik, bestehend aus IMU, RTK, V2X, LiDAR, Radar, einer Frontkamera sowie dem fahrzeuginternen CAN-Bus generiert. Zudem werden a priori Informationen in Form von Kartendaten berücksichtigt, vergleiche auch Abbildung 1.4c und Tabelle 1.2. Insgesamt sind in OPA³L dann vier Klassen von Modulen dafür verantwortlich, sämtliche vorhandenen Daten in intelligente Steuersignale zu konvertieren. Durch die Verarbeitung von Sensor- und Fahrzeugdaten im Rahmen der *Sensorfusion* werden dabei zunächst konsistente Repräsentationen des Zustandes des Fahrzeuges sowie von dessen Umwelt erzeugt. Diese beinhalten neben Informationen zu statischen und dynamischen Objekten zum Beispiel auch erkannte Fahrspuren oder Verkehrszeichen.
Die verarbeiteten Informationen werden anschließend durch die Autonomiemodule zur Planung und Entscheidungsfindung genutzt, vergleiche Abschnitt 1.1. Dabei steht auf der einen Seite die *Strategieautonomie*, welche auf höherer Ebene,

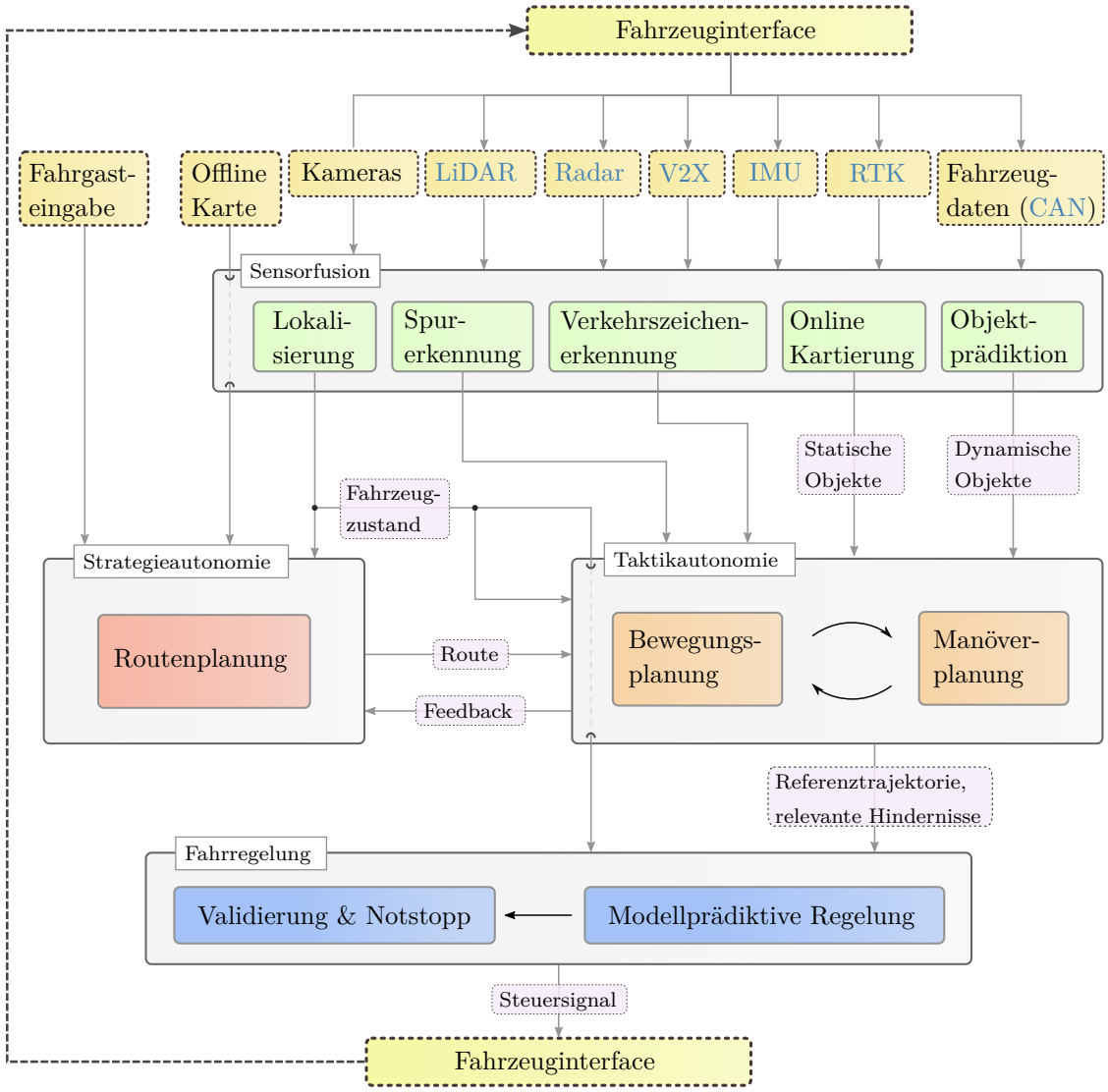


Abbildung 1.5: Vereinfachte funktionale Softwarearchitektur für das Forschungsfahrzeug in OPA^{3}L

basierend auf vorhandenem Kartenmaterial, eine Routenplanung zur Erfüllung von Fahrgastvorgaben durchführt. Auf der anderen Seite ist die Aufgabe der *Taktikautonomie*, kurzfristige Manöver so zu planen, dass eine sichere Fahrzeugführung ermöglicht wird, aber auch langfristig die strategischen Ziele erfüllt werden. Ist dies aufgrund von unerwarteten äußeren Einflüssen nicht möglich, erfolgt eine entsprechende Rückmeldung an die höhere Planungsebene der Strategieautonomie. Während strategische Entscheidungen nur sehr selten beziehungsweise situationsbedingt aktualisiert werden, berechnet die Taktikautonomie alle 100 ms eine neue Referenztrajektorie, um so eine hohe Reaktivität bezüglich der Umgebung zu gewährleisten.
Das jeweils aktuell geplante Fahrmanöver wird zusammen mit den zugehörigen

Tabelle 1.2: Kurzbeschreibung der Sensortypen des Forschungsfahrzeuges aus Abbildung 1.4c

Sensortyp	Beschreibung
Frontkamera	Nach vorne gerichtete Farbkamera zur Detektion und Klassifikation von Objekten sowie der Fahrspur.
LiDAR	Sensor zur präzisen Abstandsmessung basierend auf Messungen der reflektierten Laufzeiten von Lasersignalen. Ein Sensor misst bis zu einer Entfernung von 300 m und in einem Öffnungswinkel von 145°. Höheninformationen können durch die Messung in vier Ebenen gewonnen werden, welche sich in einem Winkel von 3,2° öffnen.
Radar	Sensor zur Messungen von Abständen und Geschwindigkeiten von Objekten. Er misst bis zu einer Entfernung von 80 m und in einem Öffnungswinkel von 40°.
RTK	Verfahren zur präzisen Bestimmung von Positionskoordinaten durch Korrektur von Satellitennavigationsdaten (GNSS) durch die Signale einer fest eingemessenen Basisstation als Referenz. Es wird eine Genauigkeit von ca. 2 cm erreicht.
IMU	Einheit zum Messen von Inertialdaten wie Beschleunigungen und Drehraten.
V2X	Kommunikationseinheit, mit der das Fahrzeug Informationen mit anderen Fahrzeugen (V2V) oder der Infrastruktur (V2I) über festgelegte Protokolle austauschen kann.
CAN	Serielles Bussystem über das Daten innerhalb des Fahrzeuges bereitgestellt werden (zum Beispiel die gemessene Eigengeschwindigkeit).

relevanten Hindernissen an die *Fahrregelung* übergeben. Ein modellprädiktiver Regler ist dann für die Berechnung und hochfrequente Bereitstellung von konkreten Steuersignalen zuständig, mit denen das vorgegebene Manöver schließlich ausgeführt wird. Im Fall von schwerwiegenden Fehlern im Fahrzeugzustand oder in kritischen Situationen mit hoher Kollisionswahrscheinlichkeit sorgt zudem ein Validierungs- und Notstopp-Modul dafür, das Fahrzeug möglichst schnell in den Stillstand zu überführen. Sämtliche Berechnungen werden dabei auf einer Intel Core i9-9900K CPU durchgeführt. Für hochgradig parallelisierbare Aufgaben (zum Beispiel bei der Bildverarbeitung) sind zudem zwei NVIDIA GeForce RTX 2080 Ti GPU's vorhanden. Eine ausführliche Beschreibung dieses Gesamtsystems ist in [33] dargestellt.

Der hierarchische Aufbau von Routenplanung, Manöverplanung und -entscheidung sowie Fahrregelung entspricht klassischen Techniken und erlaubt ein adaptives Verhalten auf verschiedenen räumlichen und zeitlichen Skalen, siehe Abschnitt 1.1. Da

der Fahrzeugzustand allen Planungs- und Regelungsmodulen bereitgestellt wird, können die entsprechenden Aufgaben simultan wahrgenommen werden. Diese *Hybride Architektur* ermöglicht einem autonomen System ein vorausschauendes Verhalten bei gleichzeitig hoher Reaktivität bezüglich abrupter Veränderungen in dessen Umwelt [4]. Hinzu kommt der spezielle Entwurf der Taktikautonomie, welcher die Manöverplanung sehr eng an eine vorgelagerte Bewegungsplanung koppelt. Dadurch kann zum einen von einer separaten Pfadplanung abgesehen werden. Zum anderen erlaubt dieser Ansatz ein hochgradig adaptives Verhalten bezüglich unbekannter, dynamischer Situationen, was vor allem in einem (sub-) urbanen Umfeld von besonderer Bedeutung ist, vergleiche Abschnitt 1.1.2.

1.3 Inhalte dieser Arbeit

Es werden kurz die inhaltlichen, theoretischen sowie numerischen Rahmenbedingungen der Arbeit zusammengefasst.

1.3.1 Ziel und Aufbau

Diese Arbeit stellt vor, wie taktische Entscheidungen für ein autonomes Fahrzeug basierend auf einem hochgradig adaptiven Bewegungsplaner umgesetzt werden können. Im Zentrum steht dabei die Entwicklung des *generischen Hybrid A**-Algorithmus, welcher die Berechnung von Trajektorien in beliebigen dynamischen Umgebungen ermöglicht. Die Auswertung der vorgestellten Verfahren wird in den Kontext von OPA^{3}L und der entsprechenden Taktikautonomie eingebettet, sodass vor allem (sub-) urbane Szenarien mit Geschwindigkeiten von maximal 50 $^{\mathrm{km}}/_{\mathrm{s}}$ betrachtet werden. Ein besonderer Fokus liegt zudem auf dem Einhalten der Rechenzeitbeschränkung von 100 ms. Unter diesem Aspekt werden die in dieser Arbeit vorgestellten Verfahren stets durch numerische Laufzeitmessungen analysiert; soweit möglich wird zudem die unter theoretischen Gesichtspunkten berechenbare asymptotische Laufzeit genannt. Eine Detailvorstellung der weiteren Komponenten des OPA^{3}L-Systems aus Abbildung 1.5 wird zum Beispiel in [81, 33] gegeben und ist nicht Teil dieser Arbeit.

Als theoretische Basis werden in Kapitel 2 zunächst die Grundlagen von graphenbasierten Suchverfahren dargestellt. Dabei wird sowohl das Verfahren von Dijkstra als auch der ordinäre A*-Algorithmus eingeführt sowie theoretische und numerische Eigenschaften präsentiert. Das Kapitel endet mit der Definition des Hybrid A*-Algorithmus, welcher die Ausgangslage für den in dieser Arbeit entwickelten Bewegungsplaner repräsentiert.
Kapitel 3 stellt dessen Komponenten zunächst detailliert vor, wobei die Definition und Berechnung eines *verallgemeinerten Voronoi-Potentials* hier eine zentrale Rolle

spielt. Dieses basiert zum einen auf einer effizienten Verallgemeinerung des Suchraumes um Zeit- und Geschwindigkeitskoordinaten. Zum anderen wird mithilfe von Freibereichspolygonen eine adaptive und generische Beschreibung der Umgebungsinformationen erreicht. Dadurch werden eine einheitliche Grundlage zur Berechnung von Voronoi-Diagrammen gelegt sowie Schwachstellen des bisherigen Ansatzes gezielt verbessert, vergleiche Abschnitt 1.1.1. Zusätzlich wird in Abschnitt 3.3.5 ein spezieller Algorithmus vorgestellt, um die Berechnung des verallgemeinerten Voronoi-Potentials auch mit wenig Rechenkapazität für Szenarien mit vielen beweglichen Objekten in möglichst geringer Rechenzeit umsetzen zu können. Zum Abschluss des Kapitels werden die im Vorfeld eingeführten theoretischen und numerischen Betrachtungen in der Definition eines *generischen Hybrid A*-Algorithmus* zusammengebracht, welcher das Kernergebnis dieser Arbeit darstellt.
Die Effizienz, Robustheit und Zuverlässigkeit dieses Bewegungsplaners wird schließlich in Kapitel 4 für den Anwendungsfall des autonomen Fahrens ausgewertet. Die numerischen Analysen werden dabei sowohl anhand von verschiedenen simulierten Szenarien als auch beim Einsatz in realen Testfahrten mit dem Forschungsfahrzeug aus Abbildung 1.4c durchgeführt.
Kapitel 5 schließt diese Arbeit mit einer Zusammenfassung und Diskussion der Ergebnisse sowie einem Ausblick zu möglichen Folgearbeiten ab.

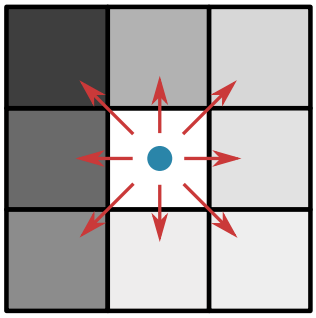
1.3.2 Rechnerarchitekturen und Programmierung

Zur Generierung von numerischen Ergebnissen und zum Treffen von Aussagen zu Laufzeiten werden in den Kapiteln 2 und 3 sowie in Abschnitt 4.1 eine AMD Ryzen 7 PRO 4750U CPU verwendet. Um numerische Artefakte dabei möglichst zu reduzieren, erfolgen Geschwindigkeitsmessungen stets durch Mittelung über mehrere konsekutive Ausführungen. Die Auswertungen auf dem Forschungsfahrzeug in Abschnitt 4.2 erfolgen mit der eingebauten Intel Core i9-9900K CPU, vergleiche auch Abschnitt 1.2. Der Quellcode wurde in allen Fällen in der Programmiersprache C++ umgesetzt und mit hoher Compileroptimierung übersetzt.

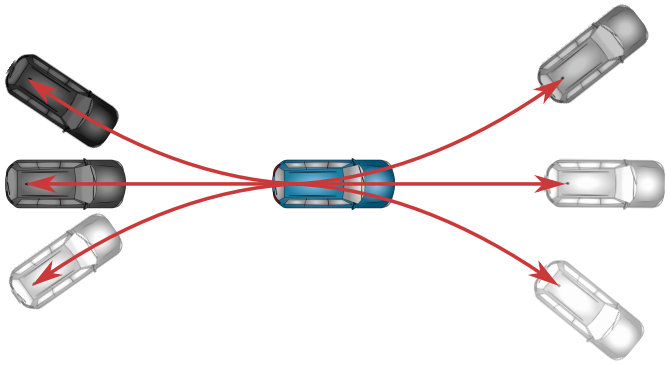
Kapitel 2

Optimale graphenbasierte Suchalgorithmen

Als Voraussetzung zur Beschreibung des Hybrid A*-Algorithmus werden in diesem Kapitel theoretische Grundlagen von graphenbasierten Optimierungsverfahren vorgestellt. Diese können zum Beispiel im Rahmen von Pfadplanungsaufgaben genutzt werden, um optimale Sequenzen von Elementen eines Suchraumes zwischen einer Start- sowie einer Endkonfiguration zu finden.
Eine solche Problemformulierung tritt nicht nur im Kontext von autonomen Fahrzeugen sondern generell in der Robotik häufig auf, weshalb sich die Beschreibungen in diesem Kapitel auf einen allgemeinen *Agenten* beziehen. Dieser kann basierend



(a) Bewegung eines abstrakten Agenten in einem zweidimensionalen Gitter



(b) Bewegung eines Fahrzeugs entlang vorgegebener Kurven

Abbildung 2.1: Beispiele für diskretisierte Bewegungsoptionen (in rot) von verschiedenen Agenten (in blau) zu benachbarten Konfigurationen (in unterschiedlichen Grauwerten). Die Tiefe des jeweiligen Farbtons beschreibt die Kosten zum Erreichen des entsprechenden Nachbarn.

auf seiner Wahrnehmung und seinen Handlungsoptionen mit der Umwelt interagieren, um festgelegte Ziele zu erreichen, vergleiche auch [86]. Häufig werden Agenten mit einer diskreten Menge an Handlungsmöglichkeiten betrachtet, bei welcher sich dieser zum Beispiel entlang vorgegebener Punkte oder anhand einer endlichen Anzahl kontinuierlicher Kurven durch seine Umgebung navigiert. Dies ist exemplarisch in Abbildung 2.1 dargestellt. Von einer gegebenen Konfiguration ist dann eine endliche Anzahl neuer Konfigurationen erreichbar, welche sich gegebenenfalls in ihren *Kosten* unterscheiden können. Je nach Anwendung kann die konkrete Definition der Kosten sehr unterschiedlich sein und zum Beispiel den zurückgelegten Weg oder die Nähe zu Hindernissen enthalten.
Solche diskretisierten Suchräume lassen sich häufig durch *gewichtete Graphen* darstellen, welche im folgenden Abschnitt 2.1 eingeführt werden. Für die Lösung des korrespondierenden Pfadplanungsproblems gibt es verschiedene algorithmische Ansätze mit entsprechend unterschiedlichen Eigenschaften. Um diese differenziert beschreiben zu können, werden in Abschnitt 2.2 einige Grundbegriffe eingeführt. Darauf aufbauend stellt Abschnitt 2.3 mit dem Algorithmus von Dijkstra ein Verfahren vor, welches die Optimalität der gefunden Lösung bezüglich der Kosten garantieren kann. In vielen realen Anwendungen kann zudem eine *Heuristik* bereitgestellt werden, welche die Entwicklung der Knotenkosten abschätzt. Für diesen Fall kann das Dijkstra-Verfahren zum A*-Algorithmus verallgemeinert werden. Dieses kann die Informationen einer geeigneten Heuristik ausnutzen, um eine ebenfalls optimale Lösung bei deutlicher Reduktion des Suchaufwands zu berechnen, vergleiche Abschnitt 2.4. Eine Spezialisierung stellt schließlich der Hybrid A*-Algorithmus dar, welcher zusätzlich in der Lage ist, Einschränkungen an die Manövrierfähigkeit (zum Beispiel eines Fahrzeuges) direkt im Suchverlauf zu berücksichtigen, vergleiche Abbildung 2.1b. Dieses

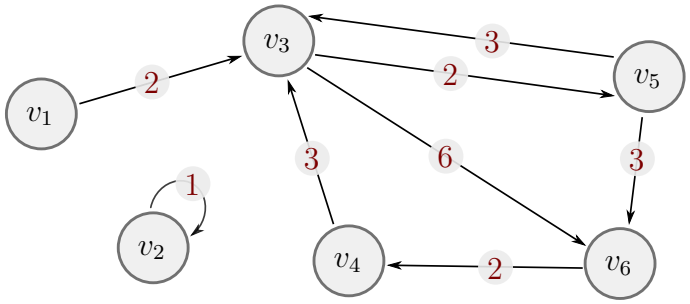


Abbildung 2.2: Beispiel für einen endlichen, gerichteten und gewichteten Graphen. Kanten zwischen den Knoten v_i sind durch Pfeile und Kantengewichte in rot dargestellt.

Verfahren wird in Abschnitt 2.5 präsentiert und entspricht der Grundlage für den in weiteren Verlauf der Arbeit entwickelten generischen Hybrid A*-Algorithmus.

Die Notation orientiert sich im Folgenden an [70]. Einen erweiterten Einblick in den Themenkomplex geben Standardwerke wie [20, 86].

2.1 Grundlagen von Graphen

Es wird zunächst allgemein der Begriff des Graphen eingeführt und anschließend spezielle Typen von Graphen beschrieben. Als visuelle Referenz für die folgenden Definitionen dient dabei Abbildung 2.2.

Definition 2.1 (Graph)

Sei $\mathcal{V}$ eine beliebige Menge und $\mathcal{E}_\mathcal{V} := \{(v_i, v_j) \mid v_i, v_j \in \mathcal{V}\}$ die Menge aller geordneter Tupel von zwei Elementen von $\mathcal{V}$. Dann heißt

$$G := (\mathcal{V}, E) \quad \text{mit} \quad E \subseteq \mathcal{E}_\mathcal{V}$$

Graph auf der *Knotenmenge* $\mathcal{V}$. Die *Knoten* $v \in \mathcal{V}$ sind dabei durch die *Kanten* $e \in E$ aus der *Kantenmenge* miteinander verbunden.

Generell ist zu beachten, dass das Tupel zur Beschreibung einer Kante geordnet ist und damit eine gerichtete Verbindung zwischen den beteiligten Knoten beschreibt. Man spricht in diesem Zusammenhang auch von einem *gerichteten Graphen*. Einen Gegenentwurf stellen die *ungerichteten Graphen* dar, bei welchen die Ordnung der Tupel einer Kante nicht beachtet wird. Da sich eine ungerichtete Verbindung durch zwei entgegengesetzte gerichtete Kanten darstellen lässt, wird für Suchalgorithmen häufig in der Praxis – und auch im Folgenden – von gerichteten Graphen ausgegangen.

Im Kontext einer gegebenen Anwendung stellen Knoten in der Regel die Elemente des (diskretisierten) Suchraumes dar, also zum Beispiel Zustände oder Konfigurationen. Kanten wiederum beschreiben die Verbindungen dieser Elemente miteinander. Für die weiteren Betrachtungen im Rahmen dieser Arbeit genügt es, endliche Graphen zu berücksichtigen; dies wird a posteriori durch Abschnitt 3.4 gerechtfertigt.

Definition 2.2 (Endlicher Graph)

Ist die Knotenmenge endlich, also $n_{\mathcal{V}} \coloneqq |\mathcal{V}| < \infty$, so überträgt sich dies auch auf jede Kantenmenge E mit $n_E \coloneqq |E| < \infty$. Das Tupel $G = (\mathcal{V}, E)$ heißt dann *endlicher Graph*.

Anmerkung 2.3 Für einen endlichen Graphen $G = (\mathcal{V}, E)$ existiert eine Indexmenge $\mathcal{I}_G \subset \mathbb{N}$, sodass sich alle Knoten $v_i, v_j \in \mathcal{V}$ mit $i, j \in \mathcal{I}_G$ eindeutig indizieren lassen, also

$$v_i = v_j \iff i = j.$$

gilt. Eine Kante von Knoten v_i zu Knoten v_j wird in diesem Fall mit $e_{i,j} \coloneqq (v_i, v_j) \in E$ beschrieben.

Knoten die sich über eine (gerichtete) Kante von einem Element aus erreichen lassen heißen Nachfolger. Die zu einem Element führenden Knoten sind seine Vorgänger.

Definition 2.4 (Nachfolger)

Sei $G = (\mathcal{V}, E)$ ein Graph und

$$N_G^{\rightarrow} \colon \mathcal{V} \to \mathcal{V}, \quad v_i \mapsto \{v_j \in \mathcal{V} \,|\, e_{i,j} \in E\} \subseteq \mathcal{V},$$

dann heißt und $v_j \in N_G^{\rightarrow}(v_i)$ *Nachfolger* von v_i.

Definition 2.5 (Vorgänger)

Sei $G = (\mathcal{V}, E)$ ein Graph und sei

$$N_G^{\leftarrow} \colon \mathcal{V} \to \mathcal{V}, \quad v_j \coloneqq \{v_i \in \mathcal{V} \,|\, e_{i,j} \in E\} \subseteq \mathcal{V}.$$

Dann heißt $v_i \in N_G^{\leftarrow}(v_j)$ *Vorgänger* von v_j. Eine Abbildung

$$\mathcal{B} \colon \mathcal{V}_{\mathcal{B}} \subseteq \mathcal{V} \to \mathcal{V}, \quad v_j \mapsto v \in N_G^{\leftarrow}(v_j),$$

die einer Teilmenge der Knoten einen speziellen Vorgänger zuordnet, wird *Vorgängerfunktion* genannt. Die k-fache Anwendung einer Vorgängerfunktion auf einen Knoten $v \in \mathcal{V}_{\mathcal{B}}$ ist definiert als

$$\mathcal{B}^{(k)}(v) \coloneqq \underbrace{\mathcal{B} \circ \ldots \circ \mathcal{B}}_{k\text{-fach}}(v), \quad \text{falls} \quad \mathcal{B}^{(l)}(v)_{|l=1,\ldots,k-1} \in \mathcal{V}_{\mathcal{B}}.$$

Anmerkung 2.6 Für einen gegebenen Knoten beschreibt eine Vorgängerfunktion eine eindeutige und geordnete Folge von Vorgängern. Um in diesem Kontext gezielte Aussagen über die enthaltenen Elemente treffen zu können, bezieht sich die Notation $(\cdot)^{(i)}$ im weiteren Verlaufe auf das i-te Element einer Folge. Damit kann sich zum Beispiel der i-te Knoten $v^{(i)}$ der betrachteten Folge von v_i, dem i-ten Knoten des unterliegenden Graphen, unterscheiden.

Eine Folge von einzigartigen Knoten wird im Folgenden als Pfad bezeichnet.

Definition 2.7 (Pfad)

Auf einem Graphen $G = (\mathcal{V}, E)$ betrachte eine geordnete Folge von Knoten und Kanten

$$P \coloneqq \left(v^{(1)}, e^{(1,2)}, \ldots, e^{(n_P-1,n_P)}, v^{(n_P)}\right)$$

mit ${v^{(i)}}_{|\, i=1,\ldots,n_P} \in \mathcal{V}$ und ${e^{(i,i+1)}}_{|\, i=1,\ldots,n_P-1} \in E$. Gilt die Einzigartigkeit der enthaltenen Elemente, also $i \neq j \implies v^{(i)} \neq v^{(j)}$ für $1 \leq i, j \leq n_P$, dann wird P als *Pfad der Länge* $|P| = n_P$ bezeichnet. Da ein Pfad durch die geordnete Angabe seiner enthaltenen Knoten eindeutig beschrieben ist, wird er auch durch

$$P \coloneqq \left\langle v^{(1)}, \ldots, v^{(n_P)} \right\rangle \quad \text{für} \quad {v^{(i)}}_{|\, i=1,\ldots,n_P} \in \mathcal{V}.$$

definiert. Die Menge aller Pfade zwischen zwei Knoten v_i, $v_j \in \mathcal{V}$ wird mit

$$\mathcal{P}(v_i, v_j) \coloneqq \{P = \langle v_i, \ \ldots, \ v_j \rangle \mid P \text{ ist Pfad von } v_i \text{ nach } v_j\}$$

bezeichnet.

Anmerkung 2.8 Auf abzählbar unendlichen Graphen lassen sich entsprechend Pfade unendlicher Länge definieren.

Basierend auf einer Vorgängerfunktion kann dann durch mehrfache Anwendung ein Pfad definiert werden.

Definition 2.9 (Pfad aus Vorgängerfunktion)

Sind für eine gegebene Vorgängerfunktion $\mathcal{B}$ und einen Knoten $v \in \mathcal{V}_\mathcal{B}$ alle Vorgänger paarweise verschieden, also $\mathcal{B}^{(i)}(v) \neq \mathcal{B}^{(j)}(v)$ für $i \neq j$ und $\mathcal{B}^{(i)}(v)$, $\mathcal{B}^{(j)}(v) \in \mathcal{V}_\mathcal{B}$, dann wird durch das Tupel $(\mathcal{B}, v)$ implizit ein Pfad definiert. Er ist durch

$$P(\mathcal{B}, v) \coloneqq \left\langle \mathcal{B}^{(k)}(v), \ldots, \mathcal{B}^{(1)}(v), v \right\rangle \quad \text{mit} \quad \mathcal{B}^{(k)}(v) \notin \mathcal{V}_\mathcal{B}$$

gegeben, wobei dessen Länge $n_P = k + 1$ vom Definitionsbereich von $\mathcal{B}$ abhängt.

Pfade und Vorgängerfunktionen sind besonders hilfreich für die Beschreibung von Suchalgorithmen auf Graphen. Dabei sind wiederum vor allem gewichtete Graphen interessant, bei denen jede Kante $e_{i,j}$ mit einem Kostenwert $d_{i,j}$ assoziiert wird.

Definition 2.10 (Gewichteter Graph)

Für einen Graphen $G = (\mathcal{V}, E)$ betrachte

$$\mathcal{D}\colon E \to \mathbb{R}, \quad e_{i,j} \mapsto d_{i,j}.$$

Dann heißt

$$G_{\mathcal{D}} \coloneqq (G, \mathcal{D}) \triangleq (\mathcal{V}, E, \mathcal{D})$$

gewichteter Graph und $\mathcal{D}$ wird als *Kostenfunktion* bezeichnet.

Daraus ergeben sich unmittelbar auch die Kosten eines ganzen Pfades als Summe von dessen Teilstücken.

Definition 2.11 (Kosten eines Pfades)

Sei $G = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph und P ein Pfad auf diesem, dann heißt

$$\mathcal{D}(P) \coloneqq \begin{cases} \infty, & \text{wenn } P = \langle\rangle \text{ leerer Pfad,} \\ \sum\limits_{e_{i,j} \in P} \mathcal{D}(e_{i,j}), & \text{sonst,} \end{cases}$$

Kosten des Pfades, wobei ein leerer Pfad unendlichen Kosten gleichgesetzt wird.

Anmerkung 2.12 Für nicht-gewichtete Graphen können die Kosten eines Pfades mit der Anzahl dessen enthaltener Kanten $n_P - 1$ gleichgestellt werden. Dies ist äquivalent dazu, jeder Kante einen Kostenwert von 1 zuzuordnen. Vor diesem Hintergrund können im weiteren Verlauf ohne Beschränkung der Allgemeinheit gewichtete Graphen betrachtet werden.

Suchalgorithmen für gewichtete Graphen stehen im Fokus der nächsten Abschnitte.

2.2 Suchalgorithmen und ihre Eigenschaften

Dieser Abschnitt stellt die Notation, Bezeichnungen und Eigenschaften vor, um Suchalgorithmen auf Graphen zu beschreiben. Dabei führt Abschnitt 2.2.1 die Grundlage zur Charakterisierung von asymptotischen Laufzeiten allgemeiner Algorithmen ein. Darauf aufbauend definiert Abschnitt 2.2.2 das im weiteren Verlauf betrachtete graphenbasierte Suchproblem und nennt wesentliche Eigenschaften von Verfahren zu dessen Lösung.

2.2.1 Asymptotische Komplexität von Algorithmen

In den folgenden Betrachtungen werden verschiedene Algorithmen unter anderem bezüglich ihrer Laufzeit miteinander verglichen. Da die konkret benötigte Rechenzeit stark von der Definition der Aufgabe, den Eingabedaten oder auch der verwendeten Rechnerhardware abhängt, wird in diesem Kontext, soweit möglich, die asymptotische Laufzeit basierend auf der Landau-Notation betrachtet, vergleiche auch [49, 76].

Definition 2.13 (Asymptotische Laufzeit)

Für zwei Funktionen $f, g\colon \mathbb{N} \to \mathbb{R}$ definieren wir

$$f(n) = \mathcal{O}(g(n)) \iff \exists c \in \mathbb{R}_{>0}\, \exists n_0 \in \mathbb{N}\, \forall \tilde{n} > n_0 : |f(\tilde{n})| \leq c \cdot |g(\tilde{n})|$$

sowie

$$f(n) = o(g(n)) \iff \forall c \in \mathbb{R}_{>0}\, \exists n_0 \in \mathbb{N}\, \forall \tilde{n} > n_0 : |f(\tilde{n})| < c \cdot |g(\tilde{n})|.$$

Gilt $f(n) = \mathcal{O}(g(n))$, so gilt, dass f *asymptotisch nicht schneller wächst als* g. Für $f(n) = o(g(n))$ *wächst* g *asymptotisch schneller als* f.

Anmerkung 2.14 Die Zeichen $\mathcal{O}$ und o werden auch *Landau-Symbole* genannt. Die Notation $f(n) = \mathcal{O}(g(n))$ ist dabei nur symbolisch gemeint; es gilt keine klassische Gleichheitsaussage.

Anmerkung 2.15 Im Kontext von Definition 2.13 ist die Aussage zum asymptotischen Wachstum über $f(n) = o(g(n))$ stärker, es gilt

$$f(n) = o(g(n)) \implies f(n) = \mathcal{O}(g(n)).$$

Mithilfe der Landau-Notation kann der *maximale* Aufwand eines Algorithmus abgeschätzt und zum Beispiel abhängig von der Größe seiner Eingabedaten $n \in \mathbb{N}$ klassifiziert werden. Die verwendete Vergleichsfunktion sollte dabei eine möglichst exakte Schranke darstellen. Es finden sich häufig die folgenden Ansätze:

$f(n) = \mathcal{O}(1)$	$\implies$	f hat *konstante* Laufzeit.
$f(n) = \mathcal{O}(\log(n))$	$\implies$	f hat *logarithmische* Laufzeit.
$f(n) = \mathcal{O}(n)$	$\implies$	f hat *lineare* Laufzeit.
$f(n) = \mathcal{O}(n \log(n))$	$\implies$	f hat *loglineare* Laufzeit.
$f(n) = \mathcal{O}(n^2)$	$\implies$	f hat *quadratische* Laufzeit.

Die gewählte Reihenfolge spiegelt die Aufwandskategorie wieder, wobei $\mathcal{O}(n^2)$ in diese Auflistung die aufwändigste Klasse darstellt. Besteht ein Verfahren additiv aus mehreren Teilen mit jeweils unterschiedlichem asymptotischen Verhalten, so kann die Klassifizierung zugunsten der aufwändigeren Repräsentation vereinfacht werden;

es gilt zum Beispiel

$$\mathcal{O}(\log(n) + 1) = \mathcal{O}(\log(n)).$$

Allgemein ist es möglich, den Aufwand auch abhängig von verschiedenen Variablen n, $m \in \mathbb{N}$ auszudrücken, was zum Beispiel durch $\mathcal{O}(n \cdot m)$ umgesetzt werden kann. Teilweise kann auch eine *amortisierte* Analyse durchgeführt werden, bei welcher der Durchschnitt der Summen der maximalen Laufzeiten einer Sequenz von Eingaben betrachtet wird. Dies macht unter anderem dann Sinn, wenn nur vereinzelte Ausführungen des betreffenden Verfahrens zu deutlich größerem Zeitaufwand führen als die übrigen.

Zu beachten ist bei den Betrachtungen zum asymptotischen Verhalten, dass alle getroffenen Aussagen sich stets auf *hinreichend große* n beziehen. Konkrete Laufzeiten eines betrachteten Algorithmus für Eingabedaten kleiner Größe können entsprechend sehr niedrig ausfallen, auch wenn sie einer hohen Aufwandskategorie zugeordnet sind, und umgekehrt.

2.2.2 Die einfache Pfadsuche

Der weitere Verlauf dieses Kapitels fokussiert sich auf Lösungsmethoden für das spezielle Suchproblem auf einem Graphen: die einfache Pfadsuche.

Definition 2.16 (Einfache Pfadsuche)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph, dann heißt die Aufgabe, einen Pfad P zwischen jeweils einem ausgezeichneten Startknoten $v^{(\text{start})} \in \mathcal{V}$ und Zielknoten $v^{(\text{end})} \in \mathcal{V}$ zu finden, *einfache Pfadsuche*.

Im Rahmen dieser Arbeit wird allgemein von *Suchalgorithmus* gesprochen, wenn Methoden zur Lösung der einfachen Pfadsuche gemeint sind. Generell lassen sich die Pfadsuche und deren Lösungsansätze auch auf mehrere Start- und Zielknoten erweitern. Zudem gibt es spezielle Methoden, um den kürzesten Pfad für alle möglichen Verbindungen zwischen Knoten eines Graphen zu finden, vergleiche [20].

Neben der zu lösenden Aufgabe selbst, lassen sich Suchalgorithmen auch anhand ihrer Lösungseigenschaften kategorisieren. Zwei wesentliche Merkmale dabei sind die *Vollständigkeit* und *Optimalität*.

Definition 2.17 (Vollständigkeit)

Ein Algorithmus heißt *vollständig*, wenn

1. eine Lösung des betrachteten Problems in endlicher Zeit gefunden wird, falls diese existiert, und
2. es berichtet wird, falls keine Lösung existiert.

Besonders für Suchalgorithmen auf Graphen ist die Vollständigkeit eine wichtige Eigenschaft, da nicht zwingend alle Knoten von einem gegebenen Startknoten aus erreichbar sind. Dies gilt zum Beispiel für die Knoten v_1 oder v_2 in Abbildung 2.2, zu welchen jeweils keine Verbindung von anderen Elementen aus besteht. Umgekehrt zeigt das Beispiel anhand der Knoten v_1 und v_6 ebenfalls, dass es auch mehrere mögliche Pfade zwischen zwei bestimmten Knoten geben kann, welche sich aber gegebenenfalls in ihren Kosten unterscheiden. Die Optimalitätseigenschaft eines Suchalgorithmus stellt sicher, dass ein gefundener Pfad immer minimale Kosten besitzt.

Definition 2.18 (Optimalität)

Sei $\mathcal{G}$ eine Menge von gewichteten Graphen. Ein Suchalgorithmus heißt *optimal auf* $\mathcal{G}$, wenn dessen Lösungspfad P (falls dieser existiert) für alle $G_\mathcal{D} = (\mathcal{V}, E, \mathcal{D}) \in \mathcal{G}$ und zwischen zwei beliebigen Knoten $v^{(\text{start})}, v^{(\text{end})} \in \mathcal{V}$ stets minimale Kosten hat, also

$$P = \underset{P_\diamond \in \mathcal{P}(v^{(\text{start})}, v^{(\text{end})})}{\arg\min} \mathcal{D}(P_\diamond)$$

gilt.

Gewisse Suchalgorithmen setzen die Nicht-Negativität aller Kantenkosten des betrachteten Graphen voraus. Für diese Methoden können negative Kostenwerte generell das Finden einer Lösung oder auch deren Optimalität verhindern. Eine Ausnahme hiervon stellt das Verfahren von *Bellman und Ford* dar [11, 34], welches ein bekanntes Beispiel für einen vollständigen und optimalen Suchalgorithmus ist, welcher explizit auch für negative Kantenkosten eingesetzt werden kann. Sind $n_\mathcal{V}$ die Anzahl der Knoten und n_E die Anzahl der Kanten, dann hat dieser die asymptotische Laufzeitkomplexität

$$\mathcal{O}(n_\mathcal{V} \cdot n_E).$$

Für viele reale Anwendungen kann jedoch ohne Einschränkung die Nicht-Negativität aller Kantenkosten eines Graphen angenommen werden. Dies gilt auch für den in dieser Arbeit entwickelten Bewegungsplaner, vergleiche Abschnitt 3.4. Mit einer solchen Einschränkung können vollständige und optimale Suchalgorithmen mit höherer Effizienz konstruiert werden, wie die nächsten Abschnitte zeigen.

2.3 Der Dijkstra-Algorithmus

Das Verfahren von *Dijkstra* [24] ist einer der bekanntesten Ansätze zur Lösung von Suchproblemen auf einem gewichteten Graphen. Dessen Grundprinzip beruht darauf, die Nachfolger von bereits bekannten Knoten zu identifizieren und so neue

bekannte Knoten zu erschließen – dieser Schritt wird im weiteren Verlauf als *Expansion* bezeichnet. Ein solches Verfahren kann zum Beispiel in einem Startknoten $v^{(\text{start})}$ initialisiert werden, wobei die Expansion dann solange erfolgt, bis ein definiertes Ziel $v^{(\text{end})}$ erreicht wird. In diesem Fall wird auch von einer *Vorwärtssuche* gesprochen. Das Verfahren lässt sich analog auch vom Zielknoten $v^{(\text{end})}$ aus umsetzen, was dann als *Rückwärtssuche* bezeichnet wird. Alternativ kann auch *bidirektional* gesucht werden, wobei dann ausgehend von $v^{(\text{start})}$ und $v^{(\text{end})}$ expandiert wird, bis eine durchgehende Verbindung zwischen diesen gefunden ist.
Die weiteren Ausführungen beziehen sich exemplarisch auf die Vorwärtssuche, in deren Kontext besonders die *minimalen Gesamtkosten* zum Erreichen eines gegebenen Knotens wichtig sind.

Definition 2.19 (Minimale Gesamtkosten)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph dann bezeichnet

$$\delta \colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}, \quad (v_i, v_j) \mapsto \min_{P_\diamond \in \mathcal{P}(v_i, v_j)} \mathcal{D}(P_\diamond)$$

die *minimalen Gesamtkosten* zwischen den Knoten $v_i, v_j \in \mathcal{V}$. Für den Startknoten einer Vorwärtssuche $v^{(\text{start})}$ stellt dann

$$g_i \coloneqq \delta(v^{(\text{start})}, v_i)$$

die minimalen Gesamtkosten zum Erreichen des Knoten v_i dar.

Ist der Pfad minimaler Gesamtkosten zu einem gegebenen Knoten bekannt, so überträgt sich diese Optimalitätseigenschaft auch auf dessen Teilpfade; dies wird auch als *Optimalitätsprinzip von Bellman* bezeichnet.

Lemma 2.20 (Optimalitätsprinzip von Bellman)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph und $P = \langle v^{(1)}, \ldots, v^{(n)} \rangle$ ein Pfad minimaler Gesamtkosten $\delta(v^{(1)}, v^{(n)})$ zwischen den Knoten $v^{(1)}$ und $v^{(n)}$. Dann ist $\widetilde{P} = \langle v^{(i)}, \ldots, v^{(j)} \rangle$ für $1 \leq i \leq j \leq n$ ein Pfad minimaler Gesamtkosten $\delta(v^{(i)}, v^{(j)})$ zwischen den Knoten $v^{(i)}$ und $v^{(j)}$.

Beweis: Die Aussage folgt aus dem Beweis von Lemma 24.1 in [20]. □

Auch wenn während einer Vorwärtssuche nur lokale Informationen über die Kosten zwischen verbundenen Knoten vorliegen, so sind die Gesamtkosten ausgehend von $v^{(\text{start})}$ trotzdem iterativ durch

$$g_j = \min_{i \in \mathcal{I}_G} \{d_{i,j} + g_i\} \tag{2.1}$$

berechenbar. Dabei kann das Optimalitätsprinzip von Bellman genutzt werden, was den Kerngedanken des Verfahrens von Dijkstra darstellt, vergleiche Algorithmus 2.1.

Algorithmus 2.1 : Dijkstra-Algorithmus zur Vorwärtssuche auf einem gewichteten Graphen

Eingabe : Gewichteter Graph $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$, Start- und Endknoten $v^{(\text{start})}, v^{(\text{end})} \in \mathcal{V}$

Ausgabe : Pfad von $v^{(\text{start})}$ zu $v^{(\text{end})}$ im Erfolgsfall, sonst leerer Pfad $\langle\rangle$

Initialisierung : Markiere $v^{(\text{start})}$ als offen, alle anderen Knoten als unbekannt, setze Vorgängerfunktion $\mathcal{B}$ mit $\mathcal{V}_{\mathcal{B}} \leftarrow \emptyset$, geschätzte Gesamtkosten $\tilde{g}_i \leftarrow \infty$ für $i \in \mathcal{I}_G$ und $\tilde{g}^{(\text{start})} \leftarrow 0$

```
1  while Es existiert ein offener Knoten do
2  |  Wähle zu expandierendes Element v_i mit i ← arg min_{k∈I_G}{g̃_k} aus der
   |   Menge der offenen Knoten;                                  // Kostenminimal
3  |  if v_i = v^(end) then
4  |  |  return Pfad P(B, v_i);
5  |  end
6  |  foreach Nachfolger v_j ∈ N_G^→(v_i) do
7  |  |  if v_j abgeschlossen then
8  |  |  |  continue
9  |  |  end
10 |  |  if v_j unbekannt or g̃_j > g̃_i + d_{i,j} then        // Kostenrelaxierung
11 |  |  |  Markiere v_j als offen;
12 |  |  |  Setze g̃_j ← g̃_i + d_{i,j};
13 |  |  |  Setze V_B ← V_B ∪ {v_j} und Vorgänger B(v_j) ← v_i;
14 |  |  end
15 |  end
16 |  Markiere v_i als abgeschlossen;
17 end
18 return ⟨⟩;                                                   // Fehlschlag
```

In diesem werden zunächst alle Knoten außer $v^{(\text{start})}$ als *unbekannt* definiert. Der Startknoten dagegen wird als *offen* markiert und stellt den Ausgangspunkt des Verfahrens da. Aus der Menge der offnen Knoten wird jeweils ein Element zur Expansion ausgewählt; dessen Nachfolger sind dann Kandidaten für weitere offene Knoten.

Die Auswahl des jeweils aktuell zu expandierenden Elementes erfolgt im Dijkstra-Algorithmus basierend auf einer Abschätzung der Gesamtkosten $\tilde{g}$. Dabei wird stets der Minimierer aus der Menge aller offenen Knoten selektiert, siehe Zeile 2. Existieren hier mehrere Kandidaten mit gleichen Werten, dann kann zum Beispiel zufällig oder chronologisch ausgewählt werden. Ist ein Nachfolger des derzeit expandierten Knotens unbekannt, wird dieser nun als offen markiert, die geschätzten Gesamtkosten berechnet sowie der Vorgängerknoten gespeichert, vergleiche Zeilen 11 bis 13. Ist ein Nachfolger bereits offen, hat aber ausgehend vom derzeit expandierenden Knoten geringere erwartete Gesamtkosten als bisher (Zeile 10), dann wurde ein kürzerer Pfad entdeckt. Mit Blick auf (2.1) erfolgt dann die sogenannte *Relaxierung*, bei der Kosten und Vorgängerfunktion des Nachfolgers ebenfalls und analog aktualisiert werden.
Das folgende Lemma beschreibt das Verhältnis zwischen geschätzten und tatsächlichen minimalen Gesamtkosten.

Lemma 2.21

Im Dijkstra-Algorithmus auf dem gewichten Graphen $G_{\mathcal{D}}$ gilt

$$g_i \leq \tilde{g}_i \quad \forall i \in \mathcal{I}_G. \tag{2.2}$$

Beweis: Die Bedingung ist für den Startknoten trivial erfüllt. Sei $v^{(i)}$ nun der erste offene Knoten, für den dies nicht gilt und $v^{(i-1)}$ sein Vorgänger, also $\tilde{g}^{(i)} = \tilde{g}^{(i-1)} + d^{(i-1,i)}$ nach Zeile 12. Dann folgt

$$\tilde{g}^{(i)} \overset{\text{Annahme}}{<} g^{(i)} \overset{\text{Lemma 2.20}}{=} g^{(i-1)} + \delta(v^{(i-1)}, v^{(i)}) \overset{\text{Konstruktion}}{\leq} \tilde{g}^{(i-1)} + d^{(i-1,i)}$$

und damit ein Widerspruch. □

Die geschätzten Gesamtkosten bleiben also grundsätzlich oberhalb der tatsächlichen Werte, wobei sie im Laufe des Verfahrens konvergieren, wie die nächste Aussage zeigt.

Lemma 2.22

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph mit nicht-negativen Kantenkosten sowie v_i ein expandierter Knoten im Dijkstra-Algorithmus, dann sind die erwarteten minimalen Gesamtkosten korrekt, also $\tilde{g}_i = g_i$.

Beweis: Für den Startknoten gilt die Aussage trivial. Sei $v_\diamond$ nun der erste offene Knoten, der nicht mit minimalen Kosten expandiert wird, also $g_\diamond < \tilde{g}_\diamond$ wegen Lemma 2.21. Dann existiert ein minimaler Pfad $P_\diamond = \langle v^{(\text{start})}, \ldots, v^{(i-1)}, v^{(i)}, \ldots, v_\diamond \rangle$ zu

$v_\diamond$ über einen offenen Knoten $v^{(i)} \neq v_\diamond$. Aus der Relaxierungsbedingung in Zeile 10 folgt zunächst

$$\tilde{g}^{(i)} \leq \tilde{g}^{(i-1)} + d^{(i-1,i)}.$$

Da $v^{(i-1)}$ per Konstruktion optimal expandiert wurde, gilt $\tilde{g}^{(i-1)} = g^{(i-1)}$. Zudem liegt $v^{(i)}$ auf einem optimalen Pfad, woraus zusätzlich $d^{(i-1,i)} = \delta(v^{(i-1)}, v^{(i)})$ folgt. Insgesamt ergibt sich damit

$$\begin{aligned} \tilde{g}^{(i)} &\overset{}{\leq} && g^{(i-1)} + \delta(v^{(i-1)}, v^{(i)}) \\ &\overset{\text{Nicht-Negativität}}{\leq} && g^{(i-1)} + \delta(v^{(i-1)}, v^{(i)}) + \delta(v^{(i)}, v_\diamond) \\ &\overset{\text{Lemma 2.20}}{=} && g_\diamond \\ &\overset{\text{Annahme}}{<} && \tilde{g}_\diamond. \end{aligned}$$

Dies widerspricht aber der Voraussetzung, dass $v_\diamond$ expandiert wurde, wonach $\tilde{g}_\diamond \leq \tilde{g}^{(i)}$ gelten muss, vergleiche Zeile 2. □

Ein bereits expandierter Knoten wird als *abgeschlossen* markiert und muss im weiteren Verlauf des Algorithmus also nicht mehr betrachtet werden, vergleiche Zeilen 7 und 16. Entspricht ein zu expandierender Knoten dem Ziel $v^{(\text{end})}$, dann endet die Suche und es wird ein Pfad ausgehend von $v^{(\text{start})}$ anhand der Vorgängerfunktion rekonstruiert (Zeile 4). Da jeder Knoten nur einmal als offen markiert und expandiert wird, sichert die Vorgängerfunktion $\mathcal{B}$ in diesem Fall die Einzigartigkeit der enthaltenen Elemente, vergleiche auch Definition 2.9. Ist kein Knoten mehr offen, dann terminiert das Verfahren ohne Ergebnis, also mit einem leeren Pfad (Zeile 18).

Satz 2.23 (Vollständigkeit und Optimalität des Dijkstra-Algorithmus)

> Der Dijkstra-Algorithmus ist auf der Menge aller gerichteter endlicher Graphen mit nicht-negativer Kostenfunktion vollständig und optimal.

Beweis: Da der Algorithmus iterativ alle Nachfolger von bereits bekannten Knoten durchsucht, werden auf endlichen Graphen perspektivisch alle von $v^{(\text{start})}$ aus erreichbaren Knoten als offen markiert. Entsprechend terminiert der Algorithmus ohne Ergebnis genau dann, wenn es keine Verbindung gibt, was die Vollständigkeit zeigt. Die Optimalitätseigenschaft folgt damit direkt aus Lemma 2.22. □

Anmerkung 2.24 Die Optimalität und Vollständigkeit gilt auch auf unendlichen Graphen, solange diese *lokal endlich* sind – also die Anzahl der ausgehenden Kanten pro Knoten beschränkt ist – mit strikt positiven Kosten für fast alle Kanten, vergleiche Abschnitt 4.1 in [86].

Die folgenden beiden Abschnitte 2.3.1 und 2.3.2 betrachten kurz mögliche Probleme des Algorithmus beim Auftreten von negativen Kantenkosten in einem Graphen sowie dessen Effizienz bei der Suche nach einem Pfad zum Zielknoten.

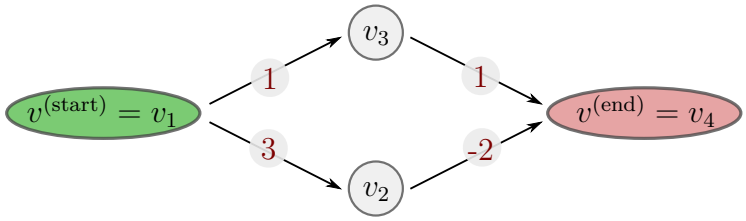


Abbildung 2.3: Beispielgraph mit negativen Kantenkosten, welcher in einem nicht-optimalen Ergebnis des Dijkstra-Algorithmus für die Pfadsuche von $v^{(\text{start})}$ nach $v^{(\text{end})}$ resultieren würde

2.3.1 Verhalten bei negativen Kantenkosten

Eine Kernvoraussetzung für die Optimalität des Dijkstra-Algorithmus ist, dass der betrachtete Graph keine negativen Kantenkosten enthält, vergleiche Satz 2.23. Abbildung 2.3 veranschaulicht exemplarisch, warum diese Anforderung wichtig ist. Hier gibt es insgesamt zwei mögliche Direktverbindungen zwischen $v^{(\text{start})}$ und $v^{(\text{end})}$. Dabei ist der Pfad über v_2 optimal; dieser hat Gesamtkosten von 1 während die Alternative über v_3 Gesamtkosten von 2 impliziert. Nach der Expansion des Startelementes sind beide Koten v_2 und v_3 als offen markiert. Aufgrund der kostenminimalen Auswahl des nächsten Knotens in Zeile 2 wird nun v_3 expandiert, dessen einziger Nachbar $v^{(\text{end})}$ ist. Dessen Gesamtkosten sind weiterhin niedriger als der Teilpfad zu v_2, weshalb das Verfahren mit einer nicht-optimalen Lösung in Zeile 4 terminiert. Ohne negative Kantenkosten sind die Gesamtkosten eines Pfades immer mindestens so hoch wie diejenigen eines beliebigen Teilpfades, weshalb das eben beschriebene Verhalten nicht auftreten kann.

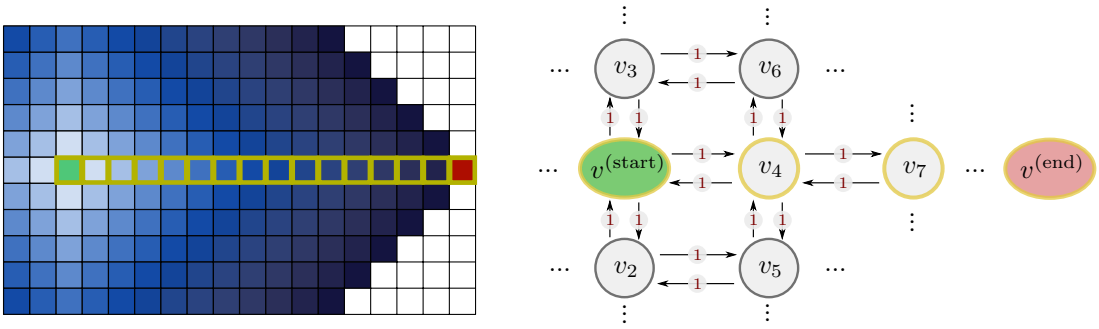
Als Suchalgorithmus auf Graphen mit negativer Kostenfunktion eignet sich zum Beispiel das Verfahren von Bellman und Ford [11, 34], welcher aber geringere Effizienz aufweist als der Ansatz von Dijkstra, vergleiche auch die Abschnitte 2.2.2 und 2.3.2.

2.3.2 Effizienzbetrachtungen

Die zusätzliche Einschränkung nicht-negativer Kantenkosten im Vergleich zum Verfahren von Bellman und Ford nutzt der Dijkstra-Algorithmus aus, um eine höhere Effizienz beim Durchsuchen des betrachteten Graphen zu ermöglichen. Es kann so eine asymptotische Zeitkomplexität von

$$\mathcal{O}(n_\mathcal{V} \cdot \log(n_\mathcal{V}) + n_E)$$

erreicht werden, vergleiche [20] und Abschnitt 2.2. Unabhängig davon hat das Verfahren jedoch Schwächen bei der Expansion zur Suche eines Pfades zu einem vorgegebenen Knoten, wie Abbildung 2.4 zeigt. Dort wird die Suche in einem offenen und hindernisfreien zweidimensionalen Raum mit Einheitskantenkosten dargestellt. Obwohl in diesem Fall der direkte Weg zum Ziel als Lösung offensichtlich ist, muss die



(a) Die Kosten der expandierten Zellen sind durch die Tiefe des Blautones und unexpandierte Zellen in weiß dargestellt

(b) Die Kanten des zugehörigen Graphen verbinden dessen Zellen horizontal und vertikal und haben jeweils Gewicht 1

Abbildung 2.4: Beispiel für die einfach Pfadsuche des Dijkstra-Algorithmus in einem zweidimensionalen Suchraum mit dem Start in grün, dem Ziel in rot und dem optimalen Pfad in gelb

Methode fast alle Knoten durchsuchen um diese zu identifizieren. Dies ist vor allem dadurch bedingt, dass bei der Wahl des expandierten Knoten keine Informationen über die relative Lage zum Zielknoten vorliegt und dieser daher lediglich auf Basis der bisherigen Gesamtkosten gewählt wird. Diese Schwäche kann der im Folgenden vorgestellte A*-Algorithmus gezielt beheben.

2.4 Der A*-Algorithmus

Das Problem der einfachen Pfadsuche wird durch den Dijkstra-Algorithmus nur ineffektiv gelöst, da Informationen über den gegebenen Zielknoten bei der Auswahl des expandierenden Knotens nicht berücksichtigt werden, vergleiche Abschnitt 2.3.2. Dieses Problem vermeidet der A*-Algorithmus [43], indem neben den bisherigen Knotenkosten g auch die verbleibenden Kosten $\delta(\cdot, v^{(\text{end})})$ zum Zielknoten $v^{(\text{end})}$ bei der Expansion berücksichtigt werden. Da letztere im Allgemeinen nicht bekannt sind, werden sie durch eine *Heuristikfunktion* abgeschätzt.

Definition 2.25 (Heuristikfunktion)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph und $v^{(\text{end})}$ der Zielknoten einer einfachen Pfadsuche, und betrachte eine Funktion

$$h\colon \mathcal{V} \to \mathbb{R}, \quad \text{mit} \quad h(v^{(\text{end})}) = 0$$

zur Approximation von $\delta(\cdot, v^{(\text{end})})$, dann heißt h *Heuristikfunktion*.

Da durch das Erreichen des Zielknotens keine weiteren Kosten entstehen, ist jede Heuristik h hier mit einem Wert von 0 exakt. Mit einer Heuristikfunktion lassen

sich schließlich die *erwarteten minimalen Gesamtkosten* zum Erreichen von $v^{(\text{end})}$ von allen Knoten des Graphen aus beschreiben.

Definition 2.26 (Erwartete minimale Gesamtkosten)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph und $v^{(\text{start})}$, $v^{(\text{end})}$ beschreiben ein einfaches Suchproblem. Für eine Heuristikfunktion h bezeichnet

$$f_i := g_i + h(v_i)$$

die *erwarteten minimalen Gesamtkosten* zwischen $v^{(\text{start})}$ und $v^{(\text{end})}$ basierend auf dem Knoten $v_i \in \mathcal{V}$.

Der A*-Algorithmus nutzt zur Expansion eine Schätzung der Gesamtkosten f und wird damit – im Gegensatz zum Dijkstra-Verfahren – zur Gruppe der *informierten Suchalgorithmen* gezählt. Die Prozedur ist in Algorithmus 2.2 dargestellt. Insgesamt gibt es große Ähnlichkeiten zu Algorithmus 2.1. Der wesentliche Unterschied betrifft die Auswahl des zu expandierenden Knoten in Zeile 2, welche anhand der Schätzung der erwarteten minimalen Gesamtkosten $\tilde{f}$ erfolgt. Diese wiederum ergibt sich direkt aus der Schätzung der Knotenkosten, vergleiche Zeile 13. Mit einer trivialen Heuristik $h_0 \equiv 0$ entspricht der A*-Algorithmus der Methode von Dijkstra und kann damit als direkte Verallgemeinerung aufgefasst werden. Unter bestimmten Annahmen an die Heuristik gilt sogar eine Äquivalenzeigenschaft, wie die folgenden Ausführungen zeigen.

Definition 2.27 (Eigenschaften von Heuristiken)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph und h eine Heuristikfunktion für eine einfache Pfadsuche mit Zielknoten $v^{(\text{end})}$, dann heißt h

1. *zulässig*, wenn gilt

$$h(v) \leq \delta(v, v^{(\text{end})}), \quad \forall\, v \in \mathcal{V};$$

2. h *monoton*, wenn gilt

$$h(v_i) \leq d_{i,j} + h(v_j), \quad \forall\, v_i \in \mathcal{V},\, v_j \in N_G^{\rightarrow}(v_i).$$

Die Zulässigkeit einer Heuristik wird auch als *optimistische Schätzung* bezeichnet. Durch sie wird sichergestellt, dass ein optimaler Pfad nicht fälschlicherweise als zu schlecht beurteilt und damit das Finden der optimalen Lösung verhindert wird. Die Eigenschaft der Monotonie wird auch als *Konsistenz* bezeichnet – sie etabliert eine Dreiecksungleichung zwischen Kostenschätzung der Heuristik und den tatsächlichen Kantenkosten und ist stärker als die Forderung der Zulässigkeit.

Algorithmus 2.2 : A*-Algorithmus zur Vorwärtssuche auf einem gewichteten Graphen

Eingabe : Gewichteter Graph $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$, Start- und Endknoten $v^{(\text{start})}, v^{(\text{end})} \in \mathcal{V}$, Heuristikfunktion h

Ausgabe : Pfad von $v^{(\text{start})}$ zu $v^{(\text{end})}$ im Erfolgsfall, sonst leerer Pfad $\langle\rangle$

Initialisierung : Markiere $v^{(\text{start})}$ als offen, alle anderen Knoten als unbekannt, setze Vorgängerfunktion $\mathcal{B}$ mit $\mathcal{V}_{\mathcal{B}} \leftarrow \emptyset$, geschätzte Knotenkosten und Gesamtkosten $\tilde{g}_i \leftarrow \infty$, $\tilde{f}_i \leftarrow \infty$ für $i \in \mathcal{I}_G$, $\tilde{g}^{(\text{start})} \leftarrow 0$ sowie $\tilde{f}^{(\text{start})} \leftarrow h(v^{(\text{start})})$

```
1  while Es existiert ein offener Knoten do
2      Wähle zu expandierendes Element v_i mit i ← arg min_{k∈I_G}{f̃_k} aus der
        Menge der offenen Knoten;                          // Gesamtkostenminimal
3      if v_i = v^(end) then
4          return Pfad P(B, v_i);
5      end
6      foreach Nachfolger v_j ∈ N_G^→(v_i) do
7          if v_j abgeschlossen then
8              continue
9          end
10         if v_j unbekannt or g̃_j > g̃_i + d_{i,j} then        // Kostenrelaxierung
11             Markiere v_j als offen;
12             Setze g̃_j ← g̃_i + d_{i,j};
13             Setze f̃_j ← g̃_j + h(v_j);                    // Gesamtkostenschätzung
14             Setze V_B ← V_B ∪ {v_j} und Vorgänger B(v_j) ← v_i;
15         end
16     end
17     Markiere v_i als abgeschlossen;
18 end
19 return ⟨⟩;                                                // Fehlschlag
```

Proposition 2.28

Die Monotonieeigenschaft einer Heuristik impliziert deren Zulässigkeit.

Beweis: Sei h eine monotone Heuristik, $v^{(1)} \in \mathcal{V}$ ein beliebiger Knoten und $P = \langle v^{(1)}, v^{(2)} \ldots, v^{(n)}, v^{(\text{end})} \rangle$ ein optimaler Pfad zum Zielknoten, dann gilt

$$h(v^{(1)}) \leq d^{(1,2)} + h(v^{(2)}) \leq \ldots \leq \underbrace{d^{(1,2)} + \ldots + d^{(n,\text{end})}}_{= \delta(v^{(1)}, v^{(\text{end})})} + \underbrace{h(v^{(\text{end})})}_{=0}$$

und damit die Zulässigkeit von h. □

Für jede monotone Heuristik eines einfachen Suchproblems, existiert ein modifizierter Graph, sodass auf diesem der Dijkstra-Algorithmus äquivalent zum A*-Algorithmus auf der ursprünglichen Aufgabe ist. Daraus lässt sich schließlich die Optimalität des letzteren herleiten. Durch die vorausschauende Eigenschaft der Heuristik selbst, sind für den A*-Algorithmus dabei insbesondere auch negative Kantenkosten erlaubt, vergleiche Abschnitt 2.3.1.

Satz 2.29 (Vollständigkeit und Optimalität des A*-Algorithmus)

Der A*-Algorithmus mit einer monotonen Heuristik h ist auf der Menge aller gerichteter endlicher Graphen vollständig und optimal.

Beweis: Seien $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph, $v^{(\text{start})}$ und $v^{(\text{end})}$ die Start- und Zielknoten einer einfach Pfadsuche sowie h die monotone Heuristikfunktion eines A*-Algorithmus. Betrachte dann den Graphen $G_{\mathcal{D}_\diamond}$ mit den modifizierten Kosten

$$\mathcal{D}_\diamond \colon E \to \mathbb{R}, \quad e_{i,j} \mapsto d_{i,j} + h(v_j) - h(v_i) =: d_{\diamond i,j}. \tag{2.3}$$

Aus der Monotonieeigenschaft von h folgt $d_{\diamond i,j} \geq 0$ (unabhängig von $d_{i,j}$ selbst), sodass der Dijkstra-Algorithmus auf dem modifizierten Graphen $G_{\mathcal{D}_\diamond}$ (mit entsprechenden Knotenkosten $g_\diamond$) optimal und vollständig ist. Dies überträgt sich auf den A*-Algorithmus des ursprünglichen Problems, wenn beide Verfahren äquivalent sind. Dazu genügt es zu zeigen, dass

1. zu jedem Zeitpunkt dieselben Knoten expandiert werden und
2. ein kürzester Pfad zwischen $v^{(\text{start})}$ und $v^{(\text{end})}$ auf $G_{\mathcal{D}_\diamond}$ ebenfalls auf $G_{\mathcal{D}}$ minimal ist.

Es wird im Folgenden ohne Beschränkung der Allgemeinheit angenommen, dass ein Gleichstand bei der Wahl des zu expandierenden Knotens von beiden Verfahren identisch aufgelöst wird.
Zu 1: Initial expandieren beide Prozeduren trivialerweise den Startknoten $v^{(\text{start})}$. Betrachtet wird nun die erste Iteration beider Algorithmen in der zwei nicht identische Knoten $v^{(n)}$ und $v_\diamond^{(n)}$ expandiert werden. Bis zu diesem Zeitpunkt sind die

Mengen der offenen Knoten per Konstruktion identisch. Seien nun $\tilde{g}_\diamond$ die geschätzten minimalen Knotenkosten im modifizierten Dijkstra-Verfahren, dann gilt

$$\begin{aligned}\tilde{g}_\diamond^{(n)} &\overset{(2.3)}{=} \sum_{i=1}^{n-1} d^{(i,i+1)} + h(v_\diamond^{(i+1)}) - h(v_\diamond^{(i)}) \\ &= \underbrace{h(v_\diamond^{(n)}) + \sum_{i=1}^{n-1} d^{(i,i+1)}}_{=:\tilde{f}_\diamond^{(n)}} - h(v^{(\text{start})}).\end{aligned}$$

Da $\tilde{g}_\diamond$ minimal unter den offenen Knoten war, folgt

$$\tilde{f}_\diamond^{(n)} \leq h(v^{(n)}) + \sum_{i=1}^{n-1} d^{(i,i+1)} = \tilde{f}^{(n)},$$

was aber bei identischer Auflösung von Kostengleichheit einen Widerspruch zu $v_\diamond^{(n)} \neq v^{(n)}$ darstellt.
Zu 2: Sei $P_\diamond = \langle v^{(\text{start})}, \ldots, v^{(\text{end})} \rangle =: \langle v^{(1)}, \ldots, v^{(n)} \rangle$ ein minimaler Pfad auf $G_{\mathcal{D}_\diamond}$ mit minimalen Knotenkosten $\delta_\diamond$, dann folgt analog zu Punkt 1

$$\delta_\diamond(v^{(\text{start})}, v^{(\text{end})}) = h(v^{(\text{end})}) + \sum_{i=1}^{n-1} d^{(i,i+1)} - h(v^{(\text{start})}).$$

und daraus die Minimalität der rechten Seite durch den A*-Algorithmus, also

$$\delta(v^{(\text{start})}, v^{(\text{end})}) = \sum_{i=1}^{n-1} d^{(i,i+1)}.$$

Dies impliziert die geforderte Eigenschaft und schließt den Beweis ab. □

Anmerkung 2.30 Für die Vollständigkeit und Optimalität des A*-Algorithmus genügt auch die Zulässigkeit der Heuristik h, wobei dann allerdings abgeschlossene Knoten nicht zwingend mit minimalen Kosten expandiert wurden und die Überprüfung in Zeile 7 nicht angewendet werden kann, vergleiche [86]. Zu berücksichtigen, dass jeder abgeschlossene Knoten dann entsprechend wieder geöffnet werden könnte, führt in der Regel zu einer spürbaren Erhöhung der Laufzeit des Verfahrens.

Mit dem Wissen, dass auch der A*-Algorithmus das Finden einer optimalen Lösung – wenn existent – garantiert, beschäftigen sich die folgenden beiden Abschnitte 2.4.1 und 2.4.2 mit der Effizienz des Verfahrens. Dabei werden sowohl theoretische Eigenschaften als auch Details bei dessen Umsetzung analysiert.

2.4.1 Effizienzbetrachtungen und Heuristiken

Der Beweis von Satz 2.29 zeigt die Äquivalenz zwischen A*-Algorithmen und einem Dijkstra-Algorithmus auf einem modifizierten Graphen. Im Allgemeinen stimmt da-

mit auch die asymptotische Laufzeitkomplexität beider Verfahren mit

$$\mathcal{O}(n_{\mathcal{V}} \cdot \log(n_{\mathcal{V}}) + n_E)$$

überein, hängt also von der Gesamtzahl der Knoten und Kanten im gegebenen Graphen ab, vergleiche auch Abschnitt 2.3.2. Tatsächlich kann die einfache Pfadsuche durch den A*-Algorithmus häufig jedoch wesentlich effizienter gelöst werden, da durch eine geeignete Heuristik deutlich weniger Knotenexpansionen notwendig sind. Insbesondere erlaubt die Auswahl der Heuristik ein Abwägen zwischen der Optimalität der Lösung und der Geschwindigkeit des Suchverfahrens, wie im Folgenden dargestellt. Dazu werden zunächst exemplarische Heuristikfunktionen eingeführt, die sich besonders für Probleme eignen, in denen die Knoten eines Graphen mit Koordinaten assoziiert sind und die Kosten der Kanten sich aus der entsprechenden Distanz zwischen benachbarten Knoten ergeben. Dann lassen sich Heuristiken aus den Metriken des betrachteten Raumes ableiten.

Definition 2.31 (Beispielheuristiken)

Sei $G_{\mathcal{D}} = (\mathcal{V}, E, \mathcal{D})$ ein gewichteter Graph mit $\mathcal{V} \subset \mathbb{R}^n$, sodass $v = \left(v^{[1]}, \ldots, v^{[n]}\right)^\top \in \mathcal{V}$ ist. Dann definiere für ein gegebenes $v^{(\text{end})} \in \mathcal{V}$:

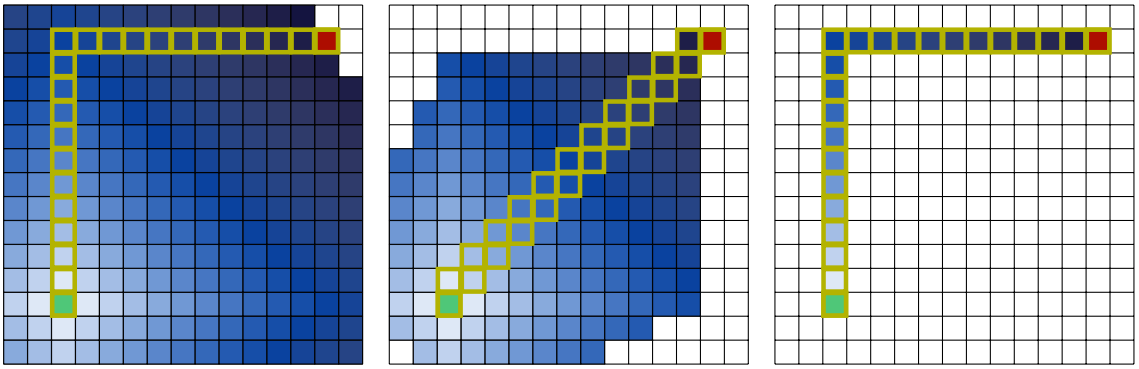
1. die *Manhattan-Heuristik* h_{manh} als Metrik induziert durch die $\|\cdot\|_1$-Norm, also

$$\begin{aligned} h_{\text{manh}} &\colon \mathcal{V} \to \mathbb{R}, \\ v &\mapsto \left|v^{[1]} - v^{(\text{end})[1]}\right| + \ldots + \left|v^{[n]} - v^{(\text{end})[n]}\right| = \sum_{l=1}^{n} \left|v^{[l]} - v^{(\text{end})[l]}\right|, \end{aligned}$$

2. die *euklidische Heuristik* h_{eukl} als Metrik induziert durch die $\|\cdot\|_2$-Norm, also

$$\begin{aligned} h_{\text{eukl}} &\colon \mathcal{V} \to \mathbb{R}, \\ v &\mapsto \left(\left(v^{[1]} - v^{(\text{end})[1]}\right)^2 + \ldots + \left(v^{[n]} - v^{(\text{end})[n]}\right)^2\right)^{1/2} \\ &= \left(\sum_{l=1}^{n} \left(v^{[l]} - v^{(\text{end})[l]}\right)^2\right)^{1/2}. \end{aligned}$$

Die Anwendung von h_{eukl}, h_{manh} und $h_0 \coloneqq 0$ ist exemplarisch in Abbildung 2.5 dargestellt. Das gezeigte Problem basiert auf dem Graphen aus Abbildung 2.4, für den diese Heuristiken monoton und damit auch zulässig sind – entsprechend finden alle Verfahren einen Pfad minimaler Länge. Allerdings ist das gezeigte Problem offensichtlich sehr symmetrisch, weshalb keine eindeutige Lösung existiert. Welchen Pfad



(a) $h = h_0$, $\tilde{g}^{(\text{end})} = 22$, # Expandiert: 221

(b) $h = h_{\text{eukl}}$, $\tilde{g}^{(\text{end})} = 22$, # Expandiert: 158

(c) $h = h_{\text{manh}}$, $\tilde{g}^{(\text{end})} = 22$, # Expandiert: 23

Abbildung 2.5: Beispiele für die einfache Pfadsuche des A*-Algorithmus mit verschiedenen Heuristiken auf dem Graphen aus Abbildung 2.4. Der Start ist in grün, das Ziel in rot, der optimale Pfad in gelb und die Kosten der expandierten Zellen durch die Tiefe des Blautones sowie unexpandierte Zellen in weiß dargestellt. Gleichstände in den geschätzten erwarteten Gesamtkosten werden zugunsten kleinerer Werte in der Heuristik und dann chronologisch aufgelöst. Die optimale Pfadlänge ist $\delta(v^{(\text{start})}, v^{(\text{end})}) = 22$.

ein A*-Algorithmus konkret identifiziert, hängt dann maßgeblich von der gewählten Heuristik aber auch von dem Vorgehen ab, mit dem Gleichstände in den erwarteten Gesamtkosten bei der Wahl des zu expandierenden Knotens aufgelöst werden.
Insbesondere die Wahl der Heuristik bestimmt schließlich auch, wie effizient der entsprechende A*-Algorithmus das gegebene Problem lösen kann. Mit der konstanten Funktion $h_0 \equiv 0$ vereinfacht sich das Verfahren zum Dijkstra-Algorithmus, sodass die Suche – wie in Abschnitt 2.3.2 gezeigt – ungezielt und damit aufwändig ist. Durch h_{eukl} und h_{manh} müssen jeweils deutlich weniger Knoten expandiert werden, wobei nur h_{manh} das Ziel *perfekt* erreicht, also unnötige Expansion vermeidet. Da im betrachteten Beispiel nur horizontale und vertikale Verbindungen existieren, ist die Manhattan-Heuristik stets exakt, also

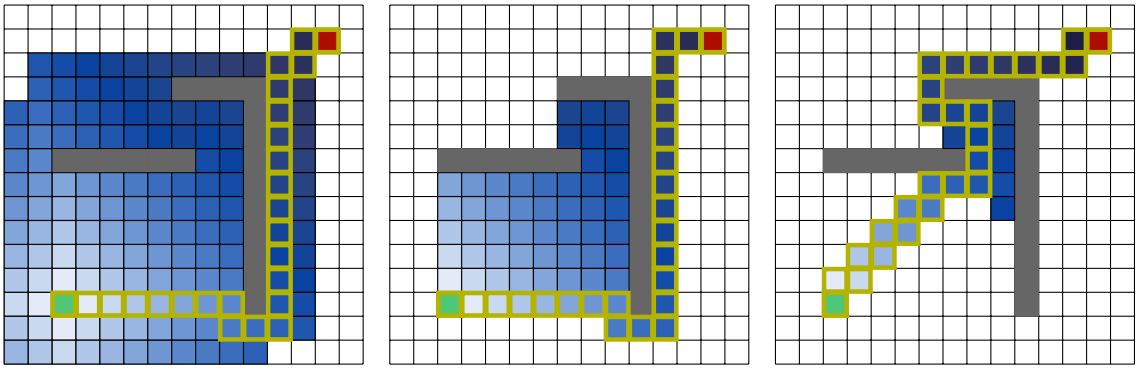
$$h_{\text{manh}}(v) = \delta(v, v^{(\text{end})}) \quad \forall v \in \mathcal{V}.$$

Mit Blick auf Algorithmus 2.2 zeigt sich, dass dann entlang aller optimalen Pfade insbesondere auch

$$\tilde{f} = f = \delta(v^{(\text{start})}, v^{(\text{end})}) = c \in \mathbb{R}_{\geq 0}$$

konstant ist, und damit niemals ein nicht-optimaler (und damit *teurerer*) Knoten expandiert wird. Die euklidische Heuristik dagegen *unterschätzt* die tatsächlichen Kosten tendenziell, es gilt also

$$h_{\text{eukl}}(v) \leq \delta(v, v^{(\text{end})}) \quad \forall v \in \mathcal{V}.$$



(a) $h = h_{\text{eukl}}$, $\tilde{g}^{(\text{end})} = 24$, # Expandiert: 148

(b) $h = h_{\text{manh}}$, $\tilde{g}^{(\text{end})} = 24$, # Expandiert: 73

(c) $h = 5 \cdot h_{\text{eukl}}$, $\tilde{g}^{(\text{end})} = 26$, # Expandiert: 33

Abbildung 2.6: Beispiele für die einfache Pfadsuche des A*-Algorithmus mit unter- und überschätzenden Heuristiken. Hindernisse sind in grau dargestellt und entsprechen nicht expandierbaren Zellen. Das restliche Setting entspricht Abbildung 2.5. Die optimale Pfadlänge ist $\delta(v^{(\text{start})}, v^{(\text{end})}) = 24$.

Insbesondere ist $\tilde{f}$ dann nicht konstant und es kann vorkommen, dass auch nicht-optimale Knoten zur Expansion gewählt werden. Im gezeigten Beispiel zeigt sich dies vor allem an den expandierten Elementen unten links, welche sich vom Ziel wegbewegen. Insgesamt passiert dies umso häufiger, je stärker die Heuristik die tatsächlichen Kosten unterschätzt.

Die Berechnung einer perfekten Heuristik ist in komplizierteren Problemen mit komplexen Kostenstrukturen oder unregelmäßigen Graphen häufig nicht möglich oder zu aufwändig. Abbildung 2.6 zeigt ein Beispiel, in welchem eine Teilmenge der Knoten als Hindernisse markiert sind. Diese werden dann nicht als Teil des betrachteten Graphen berücksichtigt und können entsprechend nicht expandiert werden. In diesem Beispiel ist auch h_{manh} unterschätzend, da kein direkter Weg zum Ziel mehr passierbar ist, sodass wesentlich mehr Knoten expandiert werden müssen als im vorigen Problem. Ist die Struktur der Hindernisse im Vorfeld nicht bekannt – wie in realen Anwendungen häufig der Fall – dann können diese auch nicht als Vorwissen durch eine Heuristikfunktion berücksichtigt werden. Um trotzdem weniger Knoten zu expandieren und damit eine Reduktion der Laufzeit zu erreichen, kann eine *überschätzende* Heuristik gewählt werden, vergleiche Abbildung 2.6c. Im Beispiel wird ein Vielfaches der Euklidischen Heuristik verwendet und damit deutlich am schnellsten das Ziel erreicht. Grundsätzlich führt ein Überschätzen der Kantenkosten dazu, dass dessen Einfluss auf die Expansion sinkt und der A*-Algorithmus vor allem versucht, möglichst kleine Werte in der Heuristik zu erreichen (was im diesem Fall der Luftlinie zwischen Start und Ziel entspricht). Dies kann einerseits eine Reduktion in der Anzahl der zu expandierenden Knoten bewirken, führt allerdings andererseits auch zu einem nicht-optimalen Verfahren, da weder Zulässigkeit noch Monotonie der Heuristikfunktion gegeben sind. Letztlich kann es dann passieren, dass – wie im

Beispiel – auch nicht-minimale Pfade als Lösungen gefunden werden.
Das Verwenden einer skalierten Heuristikfunktion wird zum Beispiel im Rahmen des *Anytime Repairing A** ausgenutzt, um zunächst möglichst schnell eine zulässige Lösung zu finden. Durch iterative Reduktion der Skalierung wird die Optimalität dann inkrementell a posteriori hergestellt [63].

Insgesamt erlaubt die Gestaltung der Heuristikfunktion also ein Abwägen zwischen der Genauigkeit des Verfahrens und dessen Laufzeit. Ist die Wahl einer perfekten Heuristik möglich, wie im Beispiel von Abbildung 2.5, so ermöglicht diese sogar das Finden einer optimalen Lösung mit minimalem Expansionaufwand. In diesem Sinne stellt sie eine Repräsentation des Vorwissens über das Problem dar: Je genauer eine Heuristik die tatsächlichen Kosten abbildet, desto effizienter wird der A*-Algorithmus eine optimale Lösung identifizieren.

Anmerkung 2.32 (Optimale Effizienz) Für eine gegebene Wissensrepräsentation über ein Problem der einfachen Pfadsuche durch eine monotone Heuristik findet der A*-Algorithmus einen minimalen Pfad mit *optimaler Effizienz*. Das heißt, kein anderes vom Startknoten aus expandierendes Verfahren kann mit weniger untersuchten Knoten terminieren (für *nicht-pathologische* Beispiele). Für eine exakte Formulierung des Satzes und den Beweis wird auf [22] verwiesen.

2.4.2 Details zur numerischen Umsetzung

Neben der (gegebenenfalls aufwändigen) Auswertung der Heuristikfunktion ist eine der wesentlichen Komponenten des A*-Algorithmus die Organisation der Knoten. Da in gezielten Suchen die meisten Knoten häufig unbekannt bleiben, ist es in der Regel am effizientesten, nur die jeweils offenen und geschlossenen Elemente in disjunkten Mengen explizit zu berücksichtigen. Diese Datenstrukturen werden dann entsprechend als *offene Menge* **O** beziehungsweise *geschlossene Menge* **C** bezeichnet – alle nicht enthaltenen Knoten sind dann implizit als unbekannt gekennzeichnet.
Mit Blick auf Algorithmus 2.2 zeigt sich, dass auf **O** und **C** regelmäßig aber mit unterschiedlicher Frequenz eine Reihe von Operationen durchgeführt werden müssen. Zum einen werden in beide Strukturen häufig Elemente hinzugefügt und später auf Existenz getestet (*Insert* und *Contains*). Zum anderen wird in jeder Iteration das jeweils beste Element aus der offenen Menge bereit gestellt (*Pop-Best*). Zudem müssen im Rahmen der Kostenrelaxierung Knoten- und Gesamtkosten sowie der entsprechende Vorgänger angepasst werden, falls ein besseres offenes Element gefunden wird (*Update*). Vor allem aufgrund der letzten beiden Operationen macht es Sinn, die Informationen über Kosten und Vorgänger zusammen mit den Knoten abzuspeichern, um gegebenenfalls Vorteile durch eine bestimmte Organisation der Daten ausnutzen zu können. Eine Zusammenfassung dieser Anforderungen ist in Tabelle 2.1 dargestellt.

Tabelle 2.1: Operationen auf der offenen Menge **O** und geschlossenen Menge **C** im A*-Algorithmus. Die Zeilennummern verweisen jeweils auf die entsprechende Stelle in Algorithmus 2.2.

Operation	Beschreibung	Verweis
Insert	Fügt einen Knoten hinzu	**O**: Zeile 11 **C**: Zeile 17
Contains	Testet, ob ein gegebener Knoten Teil der Menge ist	**O**: Zeile 10 **C**: Zeile 7
Pop-Best	Gibt den besten Knoten zurück und entfernt diesen aus der Menge	**O**: Zeile 2
Update	Ändert die Gesamtkostenschätzung und den Vorgänger eines Knotens aus der Menge	**O**: Zeile 12 bis Zeile 14

Die konkrete Umsetzung dieser Operationen hat einen hohen Einfluss auf die Laufzeit des A*-Algorithmus. Sie lassen sich durch verschiedene Datenstrukturen implementieren, welche alle jeweils bestimmte Stärken und Schwächen aufweisen. Im Folgenden werden dazu einige Ansätze vorgestellt, wobei diese basierend auf den asymptotischen Laufzeiten für die oben genannten Operationen analysiert werden. Dabei beschreibt K die Anzahl der jeweils gespeicherten Elemente. Auch wenn recht unterschiedliche Ausgangspunkte für die Umsetzung gewählt werden, erhebt die nachfolgende Auflistung nicht den Anspruch abgeschlossen zu sein. Für ausführlichere Darstellungen zu Funktionsweise und Implementierungsdetails einzelner Datenstrukturen wird zum Beispiel auf [20, 71] verwiesen.

Array Eine der einfachsten Datenstrukturen ist das *Array* (oder auch *Feld*). Bei diesem werden die Daten hintereinander im Speicher abgelegt, wodurch eine besonders kompakte Darstellung und damit auch schnelle Iteration ermöglicht wird. Der Zugriff auf ein bestimmtes Element durch einen Index kann zudem in konstanter Zeit mit $\mathcal{O}(1)$ erfolgen. Dies gilt ebenfalls für das Einfügen eines Elementes am Ende. Um das Vorhandensein eines gegebenen Datenpunktes zu prüfen muss das gesamte Array durchsucht werden, was eine Komplexität von $\mathcal{O}(K)$ bedeutet. Das Entfernen des besten Elementes kann durch Verschiebung an das Ende ebenso in konstanter Zeit erfolgen wie das Update in der Kostenrelaxierung. Für Ersteres muss der betroffene Knoten allerdings vorher gefunden werden was wieder eine Suche im Array mit $\mathcal{O}(K)$ impliziert.

Verkettete Liste Wie das Array sind *verkettete Listen* ebenfalls linear geordnete Strukturen, wobei die enthaltenen Elemente allerdings beliebig im Speicher liegen und dafür mit einem Zeiger auf das jeweils nächste Element versehen sind. Dies vereinfacht insbesondere das Entfernen von Elementen an beliebiger

Stelle (es müssen nur die Zeiger entsprechend angepasst werden), sodass auch ohne eine Verschiebung an den Rand eine konstante Zeit $\mathcal{O}(1)$ möglich ist. Alle anderen Operationen werden analog zum Array umgesetzt; insbesondere muss zum Finden eines bestimmten Elementes immer über die gesamte Struktur mit $\mathcal{O}(K)$ iteriert werden.

Hashtabelle Ein deutlich einfacherer Zugriff auf einen bestimmten Knoten kann durch eine *Hashtabelle* erreicht werden. Hier wird der Speicherort eines gegebenen Elementes durch eine Hashfunktion aus dem Element selber bestimmt. Ist die Auswertung dieser in konstanter Zeit möglich, dann kann ebenfalls mit $\mathcal{O}(1)$ eingefügt, geupdatet und auf Existenz geprüft werden. Zum Finden des besten Elementes muss allerdings auch hier die gesamte Datenstruktur mit $\mathcal{O}(K)$ durchsucht werden. Ist die Hashfunktion so ungeeignet gewählt, dass häufig Elemente auf den gleichen Hashwert abgebildet werden (dies wird *Kollision* genannt), so können sich alle Operationen auf $\mathcal{O}(K)$ verschlechtern.

Geordnetes Array Um den Zugriff auf das beste Element in der offenen Menge möglichst Effizient zu gestalten, kann zum Beispiel ein Array geordnet gehalten werden. Wird vom Ende aus sortiert, ist der Zugriff auf das beste Element und dessen Entfernen in $\mathcal{O}(1)$ möglich. Für den Existenztest müssen weiterhin alle Elemente in $\mathcal{O}(K)$ überprüft werden. Zum Einfügen muss zunächst die richtige Stelle gefunden werden, was durch eine Binärsuche in $\mathcal{O}(\log(K))$ möglich ist. Um dann den entsprechenden Speicherplatz frei zu bekommen, müssen allerdings noch alle Elemente verschoben werden, was eine Komplexität von $\mathcal{O}(K)$ bedeutet. Dies gilt analog für das Update.

Heap Eine häufig gewählte Alternative zur Konstruktion von Prioritätswarteschlangen – hier bezüglich der Knotenkosten – sind *Heaps*. Sie stellen baumartigen Datenstrukturen dar, welche, je nach konkretem Aufbau, unterschiedliche Operationen ermöglichen und verschieden effizient sind. Allen gemeinsam ist, dass das Finden des besten Elementes sehr einfach in $\mathcal{O}(1)$ möglich ist. Wird dieses entfernt, muss der Heap allerdings neu angeordnet werden, um die Sortierung aufrecht zu erhalten, was in logarithmischer Zeit $\mathcal{O}(\log(K))$ möglich ist. Zudem muss für den Test auf Existenz immer die gesamte Struktur durchsucht werden, was eine asymptotische Laufzeit von $\mathcal{O}(K)$ bedeutet. Wird speziell der *Binärheap* betrachtet, so benötigt sowohl das Einfügen als auch die Änderung der Priorität eines Elementes durch das Update ebenfalls $\mathcal{O}(\log(K))$, um die Priorisierung aufrecht zu erhalten. Wird ein *Binomialheap* [102] verwendet, dann kann mit konstanter Laufzeit $\mathcal{O}(1)$ eingefügt werden. Mit einem *Fibonacci-Heap* [38] wird zusätzlich auch der Update-Schritt in konstanter Zeit $\mathcal{O}(1)$ erfolgen. Als in der Praxis häufig sehr effizient gilt der *Pairing-Heap* [37], obwohl er im Gegensatz zum Fibonacci-Heap zur Prioritätsänderung eine asymptotische Laufzeit von $o(\log(K))$ benötigt [74].

Tabelle 2.2: Asymptotische Laufzeiten der Operationen aus Tabelle 2.1 für verschiedene Datenstrukturen

Datenstruktur	Insert	Contains	Pop-Best	Update
Array	$\mathcal{O}(1)$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$
Verkettete Liste	$\mathcal{O}(1)$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$
Hashtabelle	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$
Geordnetes Array	$\mathcal{O}(K)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	$\mathcal{O}(K)$
Binärheap	$\mathcal{O}(\log(K))$	$\mathcal{O}(K)$	$\mathcal{O}(\log(K))$	$\mathcal{O}(\log(K))$
Binomialheap	$\mathcal{O}(1)^*$	$\mathcal{O}(K)$	$\mathcal{O}(\log(K))$	$\mathcal{O}(\log(K))$
Fibonacci-Heap	$\mathcal{O}(1)$	$\mathcal{O}(K)$	$\mathcal{O}(\log(K))^*$	$\mathcal{O}(1)^*$
Pariring-Heap	$\mathcal{O}(1)$	$\mathcal{O}(K)$	$\mathcal{O}(\log(K))^*$	$o(\log(K))^*$
Hashtabelle +				
Geordnetes Array	$\mathcal{O}(K)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(K)$
Binärheap	$\mathcal{O}(\log(K))$	$\mathcal{O}(1)$	$\mathcal{O}(\log(K))$	$\mathcal{O}(\log(K))$
Binomialheap	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(K))$	$\mathcal{O}(\log(K))$
Fibonacci-Heap	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(K))$	$\mathcal{O}(1)$
Pairing-Heap	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(K))$	$o(\log(K))$

Hybride Struktur Heaps und geordnete Arrays ermöglichen einen effizienten Zugriff auf den jeweils nächsten zu expandierenden Knoten. Da der Existenz-Test jedoch jeweils recht aufwändig ist, sind beide nicht uneingeschränkt als Datenstruktur für die offene Menge zu empfehlen. Dies kann zum Beispiel durch eine hybride Struktur verbessert werden, bei der alle Elemente zusätzlich in einer Hashtabelle gespeichert werden. In diese kann vergleichsweise einfach eingefügt und dann sehr effizient auf Existenz geprüft werden. Der Zugriff auf das beste Elemente oder die Neu-Sortierung nach einem Update erfolgt entsprechend unabhängig von der Hashtabelle nur in der geordneten Struktur. In diese müssen entsprechend ebenfalls neue Elemente eingefügt werden, wodurch im Wesentlichen der Aufwand dieser Operation vorgegeben wird. Existenz-Tests können dagegen durch die Hashtabelle stets in konstanter Zeit $\mathcal{O}(1)$ umgesetzt werden. Auf diese Weise können die Stärken von beiden Datenstrukturen ausgenutzt werden, wobei jedoch ebenfalls der benötigte Speicheraufwand verdoppelt wird.

Die genannten Datenstrukturen sind gemeinsam mit den asymptotischen Laufzeiten für die betrachteten Operationen in Tabelle 2.2 zusammengefasst. Eine Aussage über die ideale Repräsentation für beispielsweise die offene Menge lässt sich daraus jedoch nicht ableiten, da es keine Struktur gibt, die bezüglich aller Operationen her-

*Amortisierte Laufzeit, vergleiche Abschnitt 2.2.1

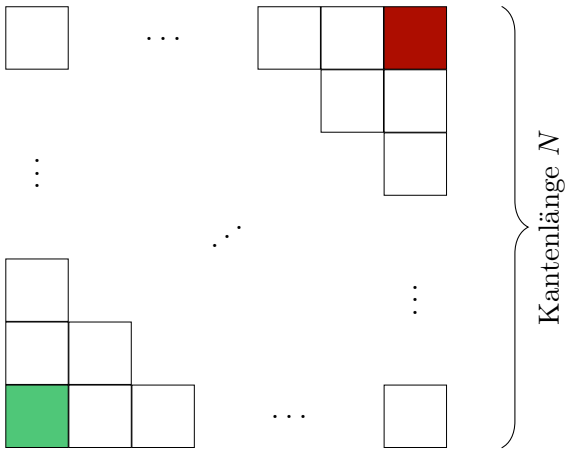


Abbildung 2.7: Generische Beispielsituation zur Evaluierung verschiedener Implementierungen des A*-Algorithmus, orientiert an Abbildung 2.5. Die Szenarien sind jeweils quadratisch mit N Feldern entlang einer Kante.

vorsticht. Die beste Implementierung kann daher stark vom konkret betrachteten Problem abhängen: Die Anzahl der Nachbarn eines Knotens bestimmt die relative Häufigkeit des Einfügens und Existenz-Tests. Die Wahrscheinlichkeit, dass Knoten mehrfach eingefügt werden, bestimmt die Anzahl der nötigen Updates in der Relaxierung. Aussagen über die asymptotischen Laufzeiten gelten hier zudem nur für eine große Zahl von Knoten, weshalb auch die Anzahl der notwendigen Expansionen selbst ein Faktor ist. Ab wann ein Problem für eine gegebene Datenstruktur dabei entsprechend als *groß* gilt, kann zudem nicht ohne weiteres klar beziffert werden. Hinzu kommen Effekte durch die teilweise recht unterschiedliche Organisation der Elemente im Speicher. So sind beispielsweise häufige Zugriffe und Iterationen bei einem kontinuierlichen Speicherblock wie dem Array in der Regel wesentlich schneller als bei einer unzusammenhängenden verketteten Liste.

Auch wenn die Implementierung der offenen Menge also problemspezifisch validiert werden sollte, wird im Folgenden ein kurzer und exemplarischer Eindruck von der Leistungsfähigkeit der vorgenannten Beispiele vermittelt. Dazu wird eine Verallgemeinerung der Aufgabe aus Abbildung 2.5 mit variabler Problemgröße betrachtet, vergleiche auch Abbildung 2.7. Je nach Kantenlänge N ergeben sich insgesamt N^2-1 zu expandierende Felder, wobei die Anzahl der tatsächlichen Expansionen bekanntermaßen von der Heuristik abhängt. Für die Darstellung der geschlossenen Menge wurde in dieser Auswertung durchweg die Hashtabelle verwendet, da beide notwendigen Operationen in konstanter Laufzeit möglich sind. Die Implementierung der gezeigten Datenstrukturen basiert weitestgehend auf der *Standard Template Library* der Programmiersprache C++ [93]. Die Grundlage der Heap-basierten Ansätze bildet dagegen die Softwarebibliothek *Boost* [13].

Abbildung 2.8 zeigt die erzielten Rechenzeiten für die euklidische Heuristik abhängig von der Problemgröße. Die Ergebnisse zeigen, dass sowohl die rein Heap-basierten

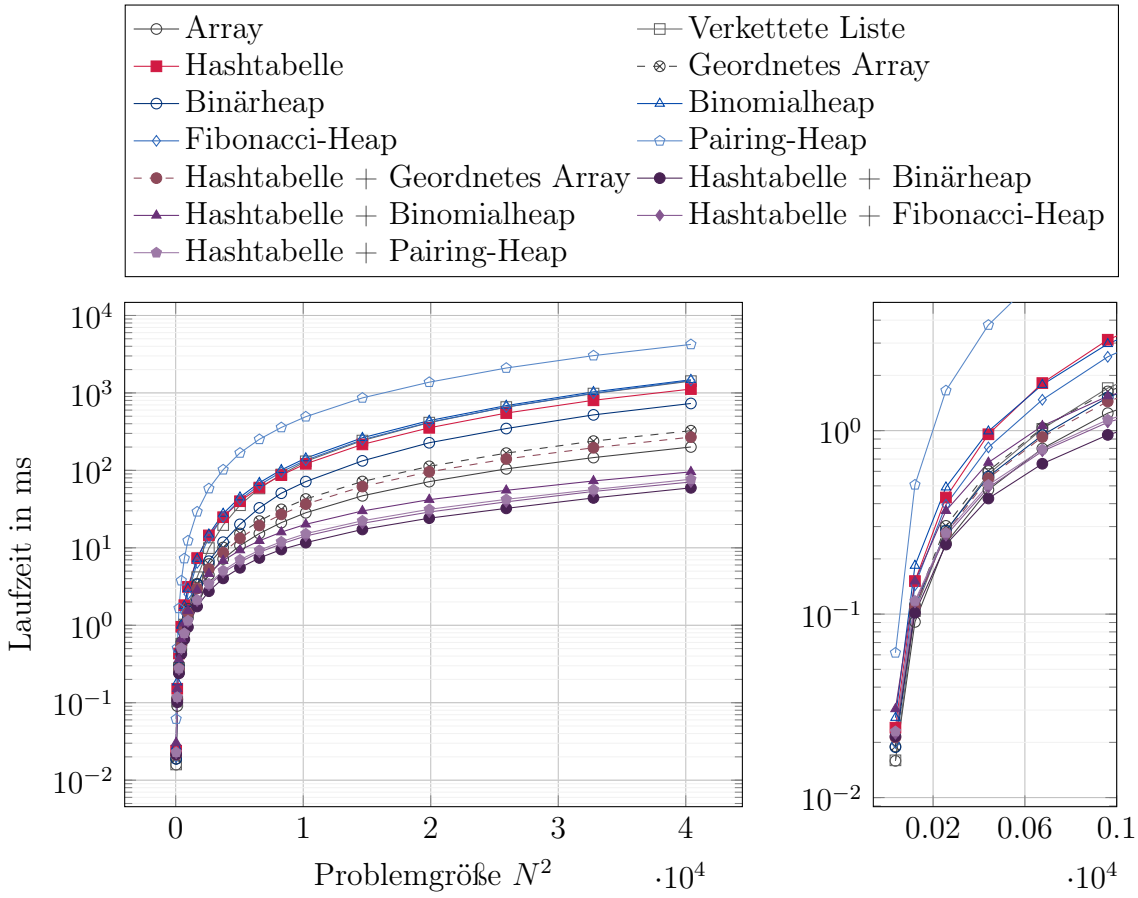


Abbildung 2.8: Mittlere Laufzeit des A*-Algorithmus zur Lösung des generischen Problems aus Abbildung 2.7 mit der **euklidischen Heuristik**. Auf der linken Seite ist die gesamte Auswertung dargestellt; die rechte Seite zeigt einen Ausschnitt für kleine Problemgrößen.

Ansätze als auch die Hashtabelle als offene Menge in diesem Fallbeispiel jeweils vergleichsweise schlecht abschneiden. Werden beide in einer hybriden Implementierung kombiniert ergeben sich hingegen die deutlich schnellsten Laufzeiten. Mittlere Geschwindigkeiten werden mit den Umsetzungen basierend auf Arrays und verketteten Listen erreicht. Insgesamt zeigt sich ein enormes Gefälle in der Laufzeit abhängig von der Darstellung der offenen Menge. Während für das größte Problem mit dem Pairing-Heap über 4000 ms benötigt werden, sind dies bei der hybriden Variante basierend auf dem Binärheap nur ca. 60 ms. Insgesamt ist diese für alle Probleme mit mehr als 200 Knoten am effizientesten.

Ein grundsätzliches ähnliches Gesamtbild ergibt sich auch durch den Einsatz der Manhattan-Heuristik in Abbildung 2.9. Da mit dieser aber insgesamt deutlich weniger Knoten expandiert werden müssen (vergleiche Abbildung 2.5), ergeben sich hier geringere Rechenzeiten und gegebenenfalls eine Verschiebung in den relativen

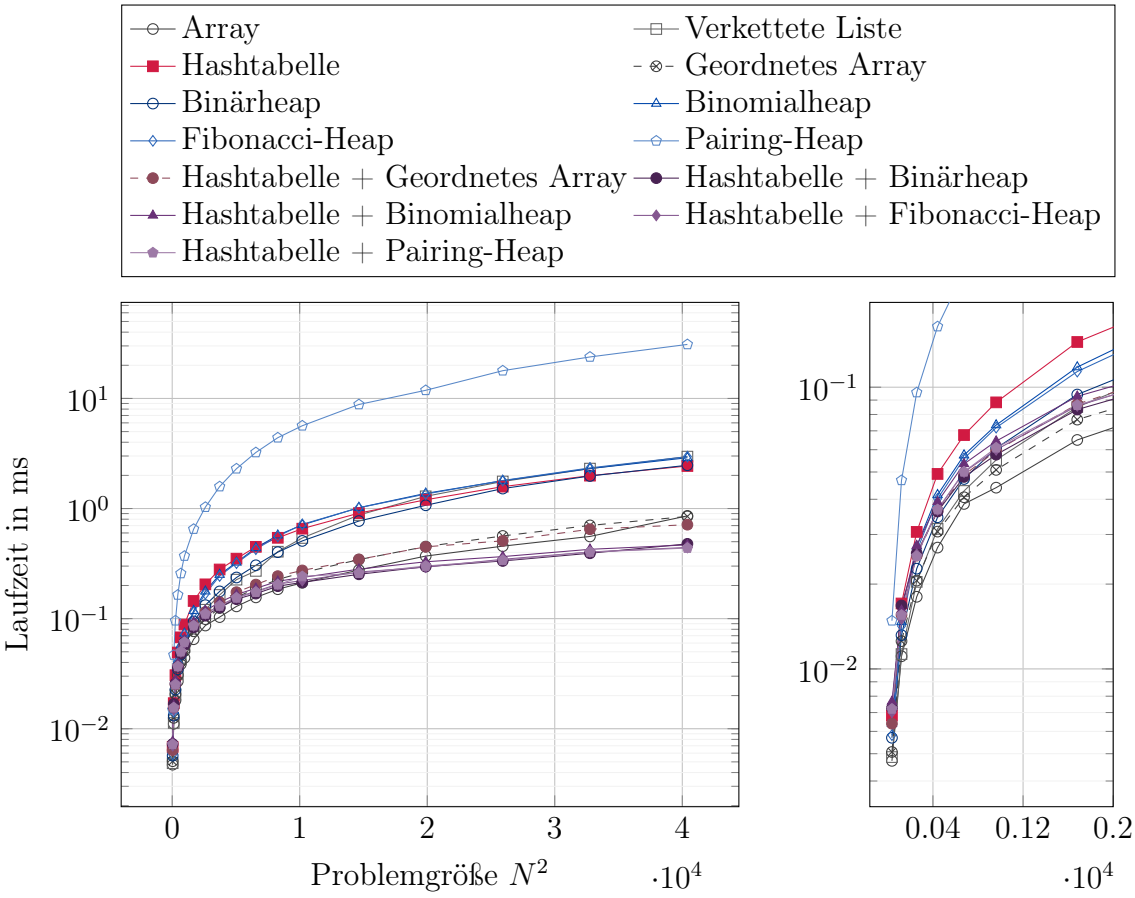


Abbildung 2.9: Mittlere Laufzeit des A*-Algorithmus zur Lösung des generischen Problems aus Abbildung 2.7 mit der **Manhattan-Heuristik**. Auf der linken Seite ist die gesamte Auswertung dargestellt; die rechte Seite zeigt einen Ausschnitt für kleine Problemgrößen.

Häufigkeiten der Operationen aus Tabelle 2.1. Hervorzuheben ist, dass für Probleme mit weniger als 15 000 Knoten die Implementierung basierend auf einem einfachen Array die besten Rechenzeiten erzielt. Auch das geordnete Array und die verkettete Liste sind hier schneller als die hybriden Heap-Varianten, die erst bei höheren Problemgrößen wieder dominieren.

Zusammenfassend lässt sich damit festhalten, dass die Implementierung der offenen Menge einen großen Einfluss auf die Performance des A*-Algorithmus hat, aber eine problemunabhängig beste Wahl aus den hier vorgestellten Ergebnissen nicht hervorgeht. Sollte der zusätzliche Speicheraufwand tolerabel sein, scheint aber insbesondere die Kombinationen einer Hashtabelle mit einem Heap konstant gute Ergebnisse zu liefern (vor allem für viele zu expandierende Knoten). Es sei allerdings noch vermerkt, dass die hier vorgestellten und untersuchten Umsetzungen keine abschließende Auflistung an Alternativen darstellen. Zur besseren Übersicht wurde

zudem die Implementierung der geschlossenen Menge hier nicht numerisch sondern nur analytisch untersucht. Ähnlich wie für die offene Menge ist aber auch hier eine problemspezifische Auswahl sinnvoll.

2.5 Hybride Graphensuche

Die in den vorigen Abschnitten eingeführten Suchalgorithmen erlauben die Generierung von Pfaden zwischen einem gegebenen Start- und Zielpunkt und ermöglichen damit, eine grundlegende Planungsebene für zum Beispiel ein autonomes Fahrzeug umzusetzen. In Kapitel 3 dieser Arbeit wird gezeigt, wie diese Basis zu einer intelligenten und automatischen Bewegungsplanung erweitert werden kann. Mit Blick auf Abschnitt 1.1.3 müssen dazu vor allem die beiden folgenden Funktionalitäten gegeben sein:

1. **Fahrzeugdynamik:** Das Ergebnis der Bewegungsplanung entspricht einer – bis zu einem gewissen Modellierungsgrad – tatsächlich befahrbaren Trajektorie des betrachteten Fahrzeugs. Die Beispiele aus Abschnitt 2.4.1 zeigen jedoch, dass das Ergebnis eines A*-Algorithmus sehr starke Richtungsänderungen beinhalten kann, durch welche diese Anforderung für einen normalen Pkw nicht erfüllt wäre. Um einer gegebenen Fahrzeugdynamik zu genügen, müsste diese also bei der Definition des Suchraums und insbesondere bei der Expansionsregel zur Generierung von Nachfolgerknoten berücksichtigt werden.

2. **Zulässigkeit:** Wie in Abschnitt 2.4.1 gezeigt ist es einfach möglich, im A*-Algorithmus bestimmte Bereiche als unpassierbar zu definieren, indem zugehörige Knoten zum Beispiel nicht im betrachteten Graphen dargestellt werden. Für ein autonomes Fahrzeug können solche Bereiche unter anderem anhand von dessen Sensorwahrnehmung erkannt werden. Um eine zulässige und damit kollisionsfreie Trajektorie zu erhalten, muss dabei jedoch auch die exakte räumliche Ausdehnung des betrachteten Fahrzeuges berücksichtigt werden. Im Falle von dynamischen Objekten (wie anderen Verkehrsteilnehmern) verändert sich die Passierbarkeit von einzelnen Knoten insbesondere über die Zeit. Für Anwendungen im Straßenverkehr kann es zudem Sinn machen, tatsächlich blockierte Bereiche (aufgrund von Hindernissen) von befahrbaren aber unerwünschten Bereichen (zum Beispiel eine andere Fahrspur) zu unterscheiden.

Einen ersten Ansatz, um diese Aspekte zumindest partiell zu erfüllen, liefert der Hybrid A*-Algorithmus, welcher innerhalb der DARPA Urban Challenge 2007 als Teil des Pfadplanungssystems für das Fahrzeug *Junior* entwickelt wurde. Die Ergebnisse dieses Verfahrens stellen eine einfache, kinematisch zulässige und kollisionsfreie

Bahn eines Fahrzeuges zwischen zwei gegebenen Konfigurationen in einem bekannten Straßennetz dar. Diese wurde dann zum einen als Entscheidungsgrundlage zur Festlegung des Fahrzeugkurses genutzt. Zum anderen diente sie als Ausgangspunkt für ein Optimierungsverfahren zur Bestimmung der tatsächlich auszuführenden Trajektorie. Für diesen Schritt wurde unter anderem das Voronoi-Potentialfeld zur Minimierung der Kollisionswahrscheinlichkeit eingeführt [25].

Die ursprüngliche Präsentation des Hybrid A*-Algorithmus ist stark an eine konkrete Anwendung sowie ein gegebenes Fahrzeug gekoppelt [25]. Im weiteren Verlauf dieses Abschnitts wird daher gezeigt, wie sich dieser Ansatz formalisieren und damit grundsätzlich auch auf allgemeine Systeme übertragen und schließlich auch für die Bewegungsplanung einsetzen lässt. Dafür wird im folgenden Abschnitt 2.5.1 zunächst das Konzept eines hybriden Graphen eingeführt. Mit dieser Notation stellt Abschnitt 2.5.2 dann eine abstrahierte Darstellung des Hybrid A*-Algorithmus vor. Insbesondere wird gezeigt, unter welchen Bedingungen sich die Optimalitätseigenschaften des ordinären A* aus Abschnitt 2.4 übertragen. Abschließend wird in Abschnitt 2.5.3 das einfache Voronoi-Potential definiert, welches eine zentrale Rolle im weiteren Verlauf dieser Arbeit spielen wird.

2.5.1 Einführung in hybride Graphen

Die Kernidee des Hybrid A*-Algorithmus ist, die graphenbasierten Suchtechniken aus Abschnitt 2.4 zu nutzen, um eine optimale Trajektorie für (zum Beispiel) ein autonomes Fahrzeug zu berechnen. Dazu sollen Steuerungen in der Regel mit dem Ziel gefunden werden, den Zustand des Fahrzeuges in einen Endzustand zu überführen. Der Zusammenhang zwischen Zuständen und Steuerungen wird durch eine modellierte Systemdynamik hergestellt. Die dabei beteiligten Funktionen unterliegen verschiedenen Anforderung an deren Stetigkeit und Differenzierbarkeit. In diesem Zusammenhang beschreibt

$$
\begin{aligned}
&\mathcal{C}^0(X, Y) && \text{die Menge der stetigen,}\\
&\mathcal{C}_p^0(X, Y) && \text{die Menge der stückweise stetigen und}\\
&\mathcal{C}^1(X, Y) && \text{die Menge der differenzierbaren Funktionen}
\end{aligned}
$$

von X nach Y. Als Teil des Urbildes X wird im Folgenden stets ein Zeitintervall

$$T := [t_0, t_f] \subset \mathbb{R} \quad \text{mit} \quad t_0 > -\infty$$

betrachtet.

Definition 2.33 (Zustände, Steuerung und Systemdynamik)

Für $t \in T$ betrachte $z(t) \in \mathcal{Z} \coloneqq \mathbb{R}^{d_z}$ zur Beschreibung des *Zustands* $z(t)$ im *Zustandsraum* $\mathcal{Z}$ eines betrachteten Systems, sowie die *Steuerung* $u(t) \in U \subseteq \mathbb{R}^{d_u}$ im *Steuerraum* U, durch welche ein Einfluss auf den Verlauf des Zustands modelliert wird. Seien die Zustände $z \in \mathcal{C}^1(T, \mathcal{Z})$ stetig differenzierbar und die Steuerungen $u \in \mathcal{C}_p^0(T, U)$ stückweise stetig, und gelte für $F \in \mathcal{C}^0(\mathcal{Z} \times U \times T, \mathbb{R}^{d_z})$ die Gleichung

$$\dot{z}(t) = F(z(t),\, u(t),\, t),$$

dann beschreibt dies ein Differentialgleichungssystem, welches als *Systemdynamik* bezeichnet wird.

Für die Berücksichtigung der Systemdynamik in einem graphenbasierten Suchalgorithmus, können deren kontinuierliche Zustände z sowie die entsprechende Zeit t mit den Knoten einer gegebenen Knotenmenge $\mathcal{V}$ zusammengefasst werden. Die daraus resultierenden hybriden Knoten stellen die Grundlage des Hybrid A*-Algorithmus dar und verleihen ihm seinen Namen.

Definition 2.34 (Hybride Knotenmenge)

Für einen Zustandsraum $\mathcal{Z}$, das Zeitintervall T und eine Knotenmenge $\mathcal{V}$ heißt

$$\mathcal{V}^{\mathcal{H}} \coloneqq T \times \mathcal{Z} \times \mathcal{V}$$

hybride Knotenmenge. Für $v^{\mathcal{H}} \in \mathcal{V}^{\mathcal{H}}$ beschreibt $v^{\mathcal{H}}_{|T} \in T$ die Zeitkomponente des Knotens; analog wird $v^{\mathcal{H}}_{|\mathcal{V}} \in \mathcal{V}$ als *diskrete* und $v^{\mathcal{H}}_{|\mathcal{Z}} \in \mathcal{Z}$ als *kontinuierliche* Knotenrepräsentation bezeichnet.

Die Idee des Hybrid A*-Algorithmus basiert darauf, die diskrete Komponente wie zuvor zur Berechnung der Heuristik eines Knotens sowie zu dessen Organisation in der offenen und geschlossenen Menge zu nutzen. Die Bestimmung von Nachfolgern in der Knotenexpansion basiert dagegen auf der kontinuierlichen Darstellung in Kombination mit der modellierten Systemdynamik. Dazu wird eine (kleine) diskrete Teilmenge des Steuerraumes, die Steuermenge

$$\overline{U} \coloneqq \{\bar{u}_1, \ldots, \bar{u}_{n_{\bar{u}}}\} \subsetneq U, \quad \text{mit} \quad n_{\bar{u}} \ll \infty$$

verwendet: für jedes enthaltene Element $\bar{u}$ kann die Systemdynamik jeweils durch ein numerisches Integrationsverfahren mit einem gegebenen Zeitschritt Δt gelöst werden, um ein Element $z \in \mathcal{Z}$ in einen neuen kontinuierlichen Zustand $z_{\bar{u}} \in \mathcal{Z}$ zu überführen. In dieser Arbeit wird dazu das Euler-Einschrittverfahren mit

$$z_{\bar{u}} \coloneqq z + F(z, \bar{u}, t) \cdot \Delta t \tag{2.4}$$

für $z \in \mathcal{Z}$ und $\bar{u} \in \overline{U}$ verwendet. Mithilfe einer geeigneten Diskretisierungsfunktion kann daraus ebenfalls ein neuer diskreter Knoten berechnet werden.

Definition 2.35 (Knotendiskretisierung)

Sei $\mathcal{Z}$ ein Zustandsraum und $\mathcal{V}$ eine endliche Menge von Knoten. Ein Operator

$$\lceil \cdot \rfloor : \mathcal{Z} \to \mathcal{V}$$

heißt *Knotendiskretisierung*.

Wenn in $\mathcal{V}$ und $\mathcal{Z}$ die gleiche Art von Koordinaten beschrieben werden (zum Beispiel Zustandsvektoren in $\mathbb{R}^{d_z}$), dann stellt die elementweise Anwendung der kaufmännischen Rundung

$$\lceil \cdot \rfloor_{\text{round}} : \mathbb{R} \to \mathbb{N}, \quad x \mapsto \lfloor |x| + 1/2 \rfloor \cdot \operatorname{sgn}(x)$$

ein einfaches Beispiel für eine Diskretisierungsfunktion dar, wobei $\lfloor \cdot \rfloor$ einer *Abrundung* und $\operatorname{sgn}(\cdot)$ der Vorzeichenfunktion entspricht. Basierend auf numerischer Integration und anschließender Diskretisierung lassen sich schließlich die Nachfolger eines gegebenen hybriden Knotens berechnen.

Definition 2.36 (Hybride Nachfolger)

Sei $\mathcal{V}^{\mathcal{H}}$ eine hybride Knotenmenge mit Systemdynamik F und Steuermenge $\overline{U}$. Für eine gegebene Diskretisierungsfunktion $\lceil \cdot \rfloor$ ergibt sich die Menge der *hybriden Nachfolger* von $v^{\mathcal{H}} \in \mathcal{V}^{\mathcal{H}}$ durch

$$N_{\overline{U}}^{\rightarrow}\left(v^{\mathcal{H}}\right) := \left\{ (t + \Delta t,\, z_{\bar{u}},\, \lceil z_{\bar{u}} \rfloor) \mid \bar{u} \in \overline{U},\, z = v^{\mathcal{H}}_{|\mathcal{Z}} \text{ und } z_{\bar{u}} \text{ aus } (2.4) \right\} \subseteq \mathcal{V}^{\mathcal{H}}.$$

Insbesondere ist $v^{\mathcal{H}}_{\bar{u}} \in N_{\overline{U}}^{\rightarrow}\left(v^{\mathcal{H}}\right)$ der entsprechende Nachfolger zur Steuerung $\bar{u} \in \overline{U}$.

Die Definition der hybriden Nachfolger ergibt sich damit analog zum Nachfolgerbegriff aus Abschnitt 2.1. Der wesentliche Unterschied besteht in der Verbindung zwischen zwei Knoten, die in diesem Fall nicht durch Kanten, sondern primär durch die Steuermenge festgelegt ist. Dies motiviert schließlich die Charakterisierung von hybriden Kanten und Graphen.

Definition 2.37 (Hybride Kante)

Sei $\mathcal{V}^{\mathcal{H}}$ eine hybride Knotenmenge mit Steuermenge $\overline{U}$, dann sei die Menge aller *hybriden Kanten* definiert durch

$$U^{\mathcal{H}} := \left\{ \left(v^{\mathcal{H}},\, \bar{u},\, v^{\mathcal{H}}_{\bar{u}}\right) \in \mathcal{V}^{\mathcal{H}} \times \overline{U} \times \mathcal{V}^{\mathcal{H}} \mid v^{\mathcal{H}}_{\bar{u}} \in N_{\overline{U}}^{\rightarrow}\left(v^{\mathcal{H}}\right) \right\}.$$

Definition 2.38 (Hybrider Graph)

Das Tupel $G^{\mathcal{H}} := (\mathcal{V}^{\mathcal{H}}, U^{\mathcal{H}})$ heißt *hybrider Graph*. Ist die Knotenmenge $\mathcal{V}$ endlich, so wird $G^{\mathcal{H}}$ *endlicher hybrider Graph* genannt. Mit einer Gewichtsfunktion $\mathcal{D}^{\mathcal{H}} \colon U^{\mathcal{H}} \to \mathbb{R}$ wird

$$G_{\mathcal{D}}^{\mathcal{H}} := (G^{\mathcal{H}}, \mathcal{D}^{\mathcal{H}}) \triangleq (\mathcal{V}^{\mathcal{H}}, U^{\mathcal{H}}, \mathcal{D}^{\mathcal{H}})$$

als *gewichteter hybrider Graph* definiert.

Begriffe aus Abschnitt 2.1 wie Pfade, Vorgänger oder Kosten lassen sich analog auf das hier betrachtete hybride Setting übertragen, sodass $G^{\mathcal{H}}$ als Spezialfall eines Graphen angesehen werden kann. Insbesondere lässt sich darauf basierend auch eine besondere Variante des A*-Suchalgorithmus definieren – der Hybrid A*, welcher im nächsten Abschnitt 2.5.2 eingeführt wird.

2.5.2 Der Hybrid A*-Algorithmus

Der vorige Abschnitt 2.5.1 hat bereits Kernideen und -begriffe des Hybrid A* eingeführt. Auf dieser Grundlage wird das komplette Verfahren zum Finden einer Trajektorie zwischen einem Startzustand $z^{(\text{start})}$ und Endzustand $z^{(\text{end})}$ in Algorithmus 2.3 dargestellt. Im Grundsatz orientiert sich dessen Aufbau am herkömmlichen A*-Algorithmus aus Abschnitt 2.4, übertragen auf hybride Knoten und Trajektorien. Zur Vereinfachung der Notation werden hier die Operationen der offenen und geschlossenen Menge aus Abschnitt 2.4.2 als Funktionen genutzt. Dabei werden insbesondere auch direkt die Kosten, Heuristiken und Vorgängerknoten in diesen organisiert. Zur Bezeichnung eines speziellen Heuristikwertes wird die Variable $\tilde{h} \in \mathbb{R}$ verwendet. Eine genaue Erläuterung ist in Tabelle 2.3 dargestellt.

Wie in den graphenbasierten Verfahren der vorigen Abschnitte, werden in jeder Iteration des Hybrid A*-Algorithmus der jeweils beste Knoten aus der offenen Menge entfernt und entsprechend die (hybriden) Nachfolger expandiert. Bereits abgeschlossene Knoten müssen dabei wie zuvor nicht erneut betrachtet werden (Zeile 7). Um die Zulässigkeit einer möglichen Lösung zu garantieren, wird zudem eine (zeitabhängige) Kollisionsüberprüfung durchgeführt (Zeile 10). Anschließend werden die Kosten des neuen Knotens iterativ berechnet und der Nachbarknoten – inklusive Knotenkosten, Heuristik und Vorgänger – in die offene Menge **O** eingefügt (Zeile 15). Falls notwendig wird eine Relaxierung durchgeführt (Zeile 17). Der aktuell betrachtete Knoten wird schließlich zusammen mit seinem Vorgänger, den Knotenkosten und dem Wert der Heuristik als abgeschlossen gespeichert (Zeile 20). Erfüllt ein expandierter Knoten ein gegebenes Terminierungskriterium (abhängig vom Zielzustand $z^{(\text{end})}$), so kann aus den gespeicherten Vorgängern die Lösungstrajektorie rekonstruiert werden (Zeile 4). Diese ist analog zur Pfaddefinition aus Abschnitt 2.1

Tabelle 2.3: Spezifikation der Funktionen von offener und geschlossener Menge im Hybrid A*-Algorithmus 2.3. Es gelten die (implementierungsabhängigen) Eigenschaften der Operationen aus Abschnitt 2.4.2.

Funktion	**Beschreibung**
insert($v^{\mathcal{H}}$, $\tilde{g}$, $\tilde{h}$, $v^{\mathcal{H}}_{\leftarrow}$)	Fügt einen hybriden Knoten $v^{\mathcal{H}} \in \mathcal{V}^{\mathcal{H}}$, dessen geschätzte Knotenkosten $\tilde{g} \in \mathbb{R}$, den Wert der Heuristik $\tilde{h} \in \mathbb{R}$ sowie den Vorgänger $v^{\mathcal{H}}_{\leftarrow} \in \mathcal{V}^{\mathcal{H}}$ als Tupel der Menge hinzu. Es erfolgt keine Rückgabe.
contains($v^{\mathcal{H}}_{\lvert\mathcal{V}}$)	Testet, ob ein hybrider Knoten mit diskreter Repräsentation $v^{\mathcal{H}}_{\lvert\mathcal{V}} \in \mathcal{V}$ teil der Menge ist. Die Rückgabe ist ein Boolean.
pop_best()	Gibt das Tupel $(v^{\mathcal{H}}, \tilde{g}, \tilde{h}, v^{\mathcal{H}}_{\leftarrow}) \in \mathcal{V}^{\mathcal{H}} \times \mathbb{R} \times \mathbb{R} \times \mathcal{V}^{\mathcal{H}}$ mit geringsten geschätzten Gesamtkosten $\tilde{g} + \tilde{h}$ aller enthaltenen Elemente zurück.
update($v^{\mathcal{H}}$, $\tilde{g}$, $\tilde{h}$, $v^{\mathcal{H}}_{\leftarrow}$)	Ist ein Element $(v^{\mathcal{H}}_{\diamond}, \tilde{g}_{\diamond}, \tilde{h}_{\diamond}, v^{\mathcal{H}}_{\diamond\leftarrow}) \in \mathcal{V}^{\mathcal{H}} \times \mathbb{R} \times \mathbb{R} \times \mathcal{V}^{\mathcal{H}}$ in der Menge enthalten und es gilt $v^{\mathcal{H}}_{\lvert\mathcal{V}} = v^{\mathcal{H}}_{\diamond\lvert\mathcal{V}}$ sowie $\tilde{g} < \tilde{g}_{\diamond}$, dann ersetzt die Eingabe $(v^{\mathcal{H}}, \tilde{g}, \tilde{h}, v^{\mathcal{H}}_{\leftarrow}) \in \mathcal{V}^{\mathcal{H}} \times \mathbb{R} \times \mathbb{R} \times \mathcal{V}^{\mathcal{H}}$ dieses bereits vorhandene Element. Es erfolgt keine Rückgabe.

durch

$$P^{\mathcal{H}}(v^{\mathcal{H}}, v^{\mathcal{H}}_{\leftarrow}, \mathbf{C}) := \left\langle v^{\mathcal{H}(n)}_{\lvert\mathcal{Z}}, \ldots, v^{\mathcal{H}(1)}_{\lvert\mathcal{Z}}, v^{\mathcal{H}}_{\lvert\mathcal{Z}} \right\rangle \quad \text{mit} \quad v^{\mathcal{H}(n)} \neq v^{\mathcal{H}(n-1)}$$

gegeben, wobei sich der erste Vorgänger durch $v^{\mathcal{H}(1)} = v^{\mathcal{H}}_{\leftarrow}$ ergibt und alle weiteren rekursiv aus der geschlossenen Menge $\mathbf{C}$ folgen. Der Vorgänger des Startknotens wird als der Knoten selbst initialisiert; dies kann schließlich als Abbruchkriterium der Rekursion benutzt werden.

Die so berechnete Trajektorie ist per Konstruktion frei von Kollisionen und genügt im Rahmen der Integrationsgenauigkeit der gegebenen Systemdynamik aus Definition 2.33. Die resultierende Verbesserung der Ausführbarkeit der Lösung im Vergleich zum klassischen A*-Algorithmus wird durch Abbildung 2.10 exemplarisch dargestellt. Zudem demonstriert sie die spezielle Expansionsstrategie des hybriden Verfahrens: Auf der einen Seite ergeben sich die kontinuierlichen Knotenrepräsentationen der hybriden Knoten frei von der unterliegenden Diskretisierung. Auf der anderen Seite kann aber nur der jeweils *beste* Knoten zu einer diskreten Repräsentation expandiert werden. In Algorithmus 2.3 wird dies durch die Zeilen 7 und 17 garantiert. Diese Eigenschaft beschreibt den Kerngedanken der Verfahren aus Kapitel 2 und sichert die Übertragung deren theoretischer Resultate.

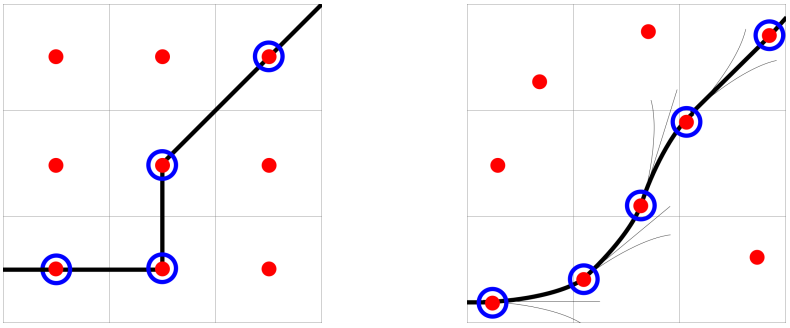
Algorithmus 2.3 : Hybrid A*-Algorithmus auf einem gewichteten hybriden Graphen

Eingabe : Gewichteter hybrider Graph $G_{\mathcal{D}}^{\mathcal{H}} = (\mathcal{V}^{\mathcal{H}}, U^{\mathcal{H}}, \mathcal{D}^{\mathcal{H}})$, Anfangs- und Endzustände $z^{(\text{start})}, z^{(\text{end})} \in \mathcal{Z}$, Heuristikfunktion h, Startzeit $t_0 \in \mathbb{R}$, Knotendiskretisierung $\lceil\rfloor$, Terminierungskriterium `IsTerminating`, Kollisionsüberprüfung `IsCollision`

Ausgabe : Trajektorie von $z^{(\text{start})}$ zu $z^{(\text{end})}$ im Erfolgsfall, sonst leere Trajektorie $\langle\rangle$

Initialisierung : Initialisiere hybriden Knoten $v^{\mathcal{H}} \leftarrow (t_0, z^{(\text{start})}, \lceil z^{(\text{start})} \rfloor)$, hybriden Vorgänger $v_{\leftarrow}^{\mathcal{H}} \leftarrow v^{\mathcal{H}}$, geschätzte Knotenkosten $\tilde{g} \leftarrow 0$, geschlossene Menge $\mathbf{C} \leftarrow \emptyset$ und offene Menge $\mathbf{O} \leftarrow \{(v^{\mathcal{H}}, \tilde{g}, h(v_{|\mathcal{V}}^{\mathcal{H}}), v_{\leftarrow}^{\mathcal{H}})\}$

```
1  while O ≠ ∅ do
2      Wähle zu expandierendes Element (v^H, g̃, h̃, v←^H) ← O.pop_best();
3      if IsTerminating(v^H, z^(end)) then   // Allgemeine Terminierungsprüfung
4          return Trajektorie P^H(v^H, v←^H, C);
5      end
6      foreach Nachfolger v_ū^H ∈ N_U→(v^H) do
7          if C.contains(v_ū|V^H) then                    // Bereits expandiert
8              continue
9          end
10         if IsCollision(v_ū^H) then                 // Allgemeine Hindernisprüfung
11             continue
12         end
13         g̃_ū ← g̃ + D^H(v^H, ū, v_ū^H);                      // Neue Knotenkosten
14         if not O.contains(v_ū|V^H) then          // Nachbar bisher unbekannt
15             O.insert(v_ū^H, g̃_ū, h(v_ū|V^H), v^H);
16         else
17             O.update(v_ū^H, g̃_ū, h(v_ū|V^H), v^H);          // Relaxierung
18         end
19     end
20     C.insert(v^H, g̃, h̃, v←^H);                  // Markiere als abgeschlossen
21 end
22 return ⟨⟩;                                                  // Fehlschlag
```



(a) A*-Algorithmus **(b)** Hybrid A*-Algorithmus

Abbildung 2.10: Exemplarischer Vergleich des A*-Algorithmus mit dem Hybrid A*-Algorithmus, aus [25]. Das Gitter beschreibt die (identische) 2D-Diskretisierung, rote Punkte stellen expandierte und abgeschlossene Knoten dar, blaue Umrandungen markieren die Lösung des Problems. Die entsprechende Trajektorie dazu ist in schwarz eingezeichnet. Die dünnen schwarzen Linien im hybriden Algorithmus beschreiben die Expansion durch numerische Integration der hier betrachteten drei möglichen Steuerungen *links*, *rechts* oder *geradeaus*, vergleiche auch Abbildung 2.1b.

Satz 2.39 (Vollständigkeit und Optimalität des Hybrid A*-Algorithmus)

> Der Hybrid A*-Algorithmus mit einer monotonen Heuristik h ist auf der Menge aller endlicher gewichteter hybrider Graphen vollständig und optimal.

Beweis: Der Hybrid A*-Algorithmus unterscheidet sich von einem herkömmlichen A*-Verfahren durch die Verwendung einer Systemmodellierung und der damit verbundenen kontinuierlichen Zustandsrepräsentation. Diese wird jeweils verwendet für

1. Terminierungs- und Kollisionsüberprüfung (Zeilen 3 und 10).
2. Berechnung von Nachfolgeknoten (Zeile 6).
3. Berechnung von Knotenkosten (Zeile 13).

Alle drei Punkte legen implizit fest, welche diskreten Knotenkomponenten durch den Suchalgorithmus erreichbar sind und expandiert werden. Diese deterministische Relation lässt sich äquivalent durch einen Graphen im Sinne von Kapitel 2 ausdrücken. Dessen Endlichkeit ergibt sich direkt aus der Endlichkeit der Knotenmenge $\mathcal{V}$ und Steuermenge $\overline{U}$. Der sonstige Verlauf des Algorithmus, zum Beispiel die Berechnung des Heuristikwertes sowie die Ergebnisse der Operationen von offener und geschlossener Menge, hängen dann nicht mehr von der kontinuierlichen Darstellung ab. Damit kann der Hybrid A*-Algorithmus als eine spezielle Variante des herkömmlichen A*-Verfahrens interpretiert werden, sodass die Aussage direkt aus dem Beweis von Satz 2.29 folgt. □

Anmerkung 2.40 Als Spezialfall eines A*-Algorithmus übertragen sich auch die Aussagen zur Laufzeitkomplexität

$$\mathcal{O}(n_{\mathcal{V}} \cdot \log(n_{\mathcal{V}}) + n_E)$$

und Effizienz aus Abschnitt 2.4.1 auf Algorithmus 2.3. Die Anzahl der Knoten und Kanten ergibt sich dabei aus dem implizit definierten Graphen des vorstehenden Beweises.

Anmerkung 2.41 Je nach Anwendung kann es sinnvoll sein, zum Beispiel die Steuermenge $\overline{U}$ beziehungsweise den Zeitschritt Δt basierend auf der vorliegenden Situation zu wählen, um so eine erhöhte Genauigkeit oder einen speziellen Fokus in der Verbindung von Knoten zu erreichen. Eine einfache Bewertungsgrundlage dafür kann der aktuelle diskrete Zustand sein, woraus sich

$$\overline{U} = \overline{U}\left(v^{\mathcal{H}}_{|\mathcal{V}}\right) \quad \text{und} \quad \Delta t = \Delta t\left(v^{\mathcal{H}}_{|\mathcal{V}}\right)$$

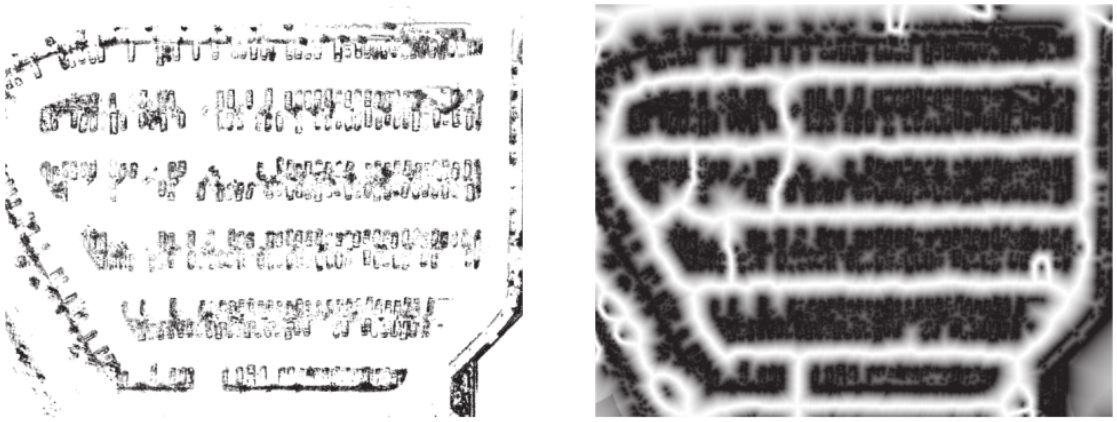
ergibt. Solche Abhängigkeiten verändern nicht die Endlichkeitseigenschaft des implizit gegebenen Graphen und sind somit verträglich mit Satz 2.39 umsetzbar.

Anmerkung 2.42 In Abschnitt 3.2 werden Freibereichspolygone eingeführt, welche die räumliche Ausdehnung des Suchbereichs einschränken. Dies wird in Abschnitt 3.4 eine der wesentlichen Komponenten zur Definition einer diskreten Knotenmenge $\mathcal{V}$ als Voraussetzung von Satz 2.39 sein.

Auch für den Hybrid A*-Algorithmus ist also sichergestellt, dass effizient eine optimale – also kostenminimale – Trajektorie als Lösung berechnet wird, wenn diese existiert. Die Frage der Existenz selbst ist dabei abhängig von der betrachteten Anwendung und kann durch die konkrete Vorgabe der Systemdynamik, Zustandsräume und Steuerungen sowie der bisher nicht weiter spezifizierten Funktionen `IsCollision` und `IsTerminating` beeinflusst werden. Die tatsächliche Umsetzung dieser Punkte im Kontext dieser Arbeit wird in Abschnitt 3.4 detailliert dargestellt. Dies gilt ebenso für die Definition der Heuristik sowie der Knotenkosten. Als wesentlicher Bestandteil von letzteren wird eine dynamische Verallgemeinerung des Voronoi-Potentials verwendet werden, welches in seiner grundlegenden Form im Folgenden vorgestellt wird.

2.5.3 Das Voronoi-Potential

In der Einführung dieses Abschnittes wurde die *Zulässigkeit* der Lösungstrajektorie als Teil der Minimalanforderungen an die Bewegungsplanung eingeführt. Dies wird im Hybrid A*-Algorithmus 2.3 direkt durch die Kollisionsprüfung sichergestellt. Effektiv werden dadurch alle unzulässigen Knoten aus dem Suchraum entfernt. In Abschnitt 2.4.1 wurden entsprechende Beispiele für die Pfadsuche mit einer einfachen A*-Suche bereits betrachtet, bei denen die Minimierung der Pfadlänge das Ziel



(a) Beispieldaten **(b)** Voronoi-Potentialfeld

Abbildung 2.11: Statisches Voronoi-Potentialfeld für eine exemplarische Punktwolke, aus [25]. Weiße Bereiche implizieren ein niedriges, schwarze Bereiche ein hohes Potential.

war. Liegen blockierte Knoten zwischen Start und Ziel, dann führt dies in der Regel dazu, dass diese sehr eng passiert werden, wie in Abbildung 2.6 dargestellt. Wird im hybriden Verfahren vor allem die Länge der resultierenden Trajektorie minimiert, überträgt sich diese Beobachtung, sodass vorhandene Hindernisse sehr eng umfahren werden. Insbesondere im Kontext von autonomen Fahrzeugen (aber auch anderen autonomen oder automatisierten Systemen) führt dies mindestens zu einem starken Gefühl der Gefährdung beim möglichen Fahrgast oder sogar zu einer erhöhten Kollisionswahrscheinlichkeit durch Unsicherheiten im Wissen über die Umgebung.

Ein in dieser Hinsicht verbessertes Verhalten ergibt sich durch die Maximierung des Abstandes zu Hindernissen soweit es das Umfeld zulässt. Im ursprünglichen Hybrid A*-Ansatz wird dies in einer nachgelagerten Optimierung durch eine *konjugierte Gradienten*-Methode bezüglich des speziell konstruierten Voronoi-Potentialfeldes realisiert [25]. Dieses ist für Beispieldaten eines komplett vermessenen Parkplatzes in Abbildung 2.11 dargestellt. Es ist gut zu erkennen, dass der Wert des Potentialfeldes umso niedriger wird, desto zentraler sich die betrachtete Position auf einer freien Fahrspur befindet. In der Nähe von Hindernisse werden besonders hohe Potentiale erreicht.

Im Folgenden wird kurz die Definition des Potentialfeldes vorgestellt, welche im Kern auf der Berechnung eines Voronoi-Diagramms für gegebene Daten als Punktwolke $D \subset \mathbb{R}^2$ basiert. Für diese werden die Daten in zusammenhängende Strukturen gruppiert und der Raum anschließend in Zellen eingeteilt, sodass sich alle Elemente in einer Zelle ein gemeinsames Cluster von Datenpunkten teilen, zu dem sie den kürzesten Abstand haben, vergleiche Abbildung 2.12. Die Kanten zwischen den Zellen des Voronoi-Diagramms entsprechen per Konstruktion denjenigen Orten, welche die (lokal) größten Abstände zu den Eingabedaten aufweisen. Werden diese als Hin-

Abbildung 2.12: Voronoi-Diagramm für die Punktwolke aus Abbildung 2.11, nach [25]. Die gegebenen Datenpunkte sind jeweils in weiß und die Zellen des Voronoi-Diagramms durch die verschieden gefärbten Bereiche gekennzeichnet.

dernisse interpretiert, so ermöglicht eine Navigation entlang der *Voronoi-Kanten* $\mathcal{R}$ entsprechend eine Maximierung des Sicherheitsabstandes, was einem Kerngedanken bei der Konstruktion des Voronoi-Potentialfeldes $\overline{\Phi}_V$ entspricht. Für eine gegebene Position $p \in \mathbb{R}^2$ in der Ebene berechnet sich dann dessen Wert nach [25] durch

$$\begin{aligned} &\overline{\Phi}_V \colon \mathbb{R}^2 \to \mathbb{R}, \\ &p \mapsto \frac{\alpha}{\alpha + \mathrm{d}(p,\, D)} \cdot \frac{\mathrm{d}(p,\, \mathcal{R})}{\mathrm{d}(p,\, D) + \mathrm{d}(p,\, \mathcal{R})} \cdot \left(\frac{\mathrm{d}(p,\, D) - \mathrm{d}^{\max}}{\mathrm{d}^{\max}} \right)^2 . \end{aligned} \tag{2.5}$$

Dieses hängt im Wesentlichen von den jeweils kürzesten Distanzen des Punktes p zu den Voronoi-Kanten $\mathrm{d}(p,\, \mathcal{R})$ sowie zu den Datenpunkten $\mathrm{d}(p,\, D)$ ab. Dabei folgt im Fall einer *Kollision* mit den Datenpunkten, also $\mathrm{d}(p,\, D) = 0$, ein Potential von $\overline{\Phi}_V(x, y) = 1$. Liegt der getestete Punkt hingegen genau auf einer Voronoi-Kante, also $\mathrm{d}(p,\, \mathcal{R}) = 0$, so ergibt sich $\overline{\Phi}_V(x, y) = 0$. Per Konstruktion der Voronoi-Kanten gilt dabei, dass nur eine der beiden Distanzen tatsächlich den Wert 0 annehmen kann, sodass $\overline{\Phi}_V$ stets wohldefiniert ist. Der Parameter $\alpha \in \mathbb{R}_{>0}$ steuert den Anstieg des Potentialfeldes zwischen diesen beiden Grenzfällen, wobei höhere Werte von α zu einem schnelleren Wachstum des Potentials in Richtung der Datenpunkte führen. Zuletzt erfolgt eine Dämpfung durch die Berücksichtigung der gegebenen maximalen Distanz $\mathrm{d}^{\max} \in \mathbb{R}_{\geq 0}$. Entspricht die Entfernung zu den Datenpunkten genau diesem Wert, also $\mathrm{d}(p,\, D) = \mathrm{d}^{\max}$, dann verschwindet das Potential auch hier. Damit dies auch für noch größere Distanzen $\mathrm{d}(p,\, D) > \mathrm{d}^{\max}$ gilt, wird das beschränkte Voronoi-

Potential

$$\begin{aligned} &\Phi_V : \mathbb{R}^2 \to [0,1] \subset \mathbb{R}, \\ &p \mapsto \begin{cases} \overline{\Phi}_V(p), & \text{für } \mathrm{d}(p,\, D) < \mathrm{d}^{\max}, \\ 0, & \text{sonst}, \end{cases} \end{aligned} \tag{2.6}$$

definiert.

Die Einführung dieses speziellen Ansatzes wird dabei durch drei wesentliche Eigenschaften motiviert [25]:

1. **Räumliche Adaptivität:** Der Anstieg des Potentials passt sich an den lokal verfügbaren Platz an. Durch das Vorhandensein einer Voronoi-Kante zwischen zwei Hindernissen, ist es dabei stets möglich, eine potentialfreie Lösung zu finden; dies gilt explizit auch in engen Bereichen, wenn diese ohne Kollision möglich ist. Potentiale, die alternativ nur auf Basis der Hindernisdistanz definiert werden, führen in schmalen Passagen stets zu hohen Kosten und erschweren hier entsprechend die Navigation.

2. **Effizienz durch Vereinfachung:** Zur Auswertung des Voronoi-Potentials sind vor allem die Randpunkte eines jeweiligen Clusters von Daten relevant; alle innenliegenden Elemente können dagegen vernachlässigt werden. Durch eine geeignete Darstellung der Eingabedaten kann die Berechnung des Potentialwertes damit sehr effizient umgesetzt werden.

3. **Differenzierbarkeit:** Das Voronoi-Potential ist fast überall differenzierbar (bis auf die Positionen der Datenpunkte und Voronoi-Kanten) und eignet sich damit zur numerischen Nutzung in der eingangs beschriebenen Methode der konjugierten Gradienten.

Im folgenden Kapitel 3 wird der generische Hybrid A*-Algorithmus als Erweiterung des hier vorgestellten Verfahrens eingeführt. Dieser wird am Beispiel des autonomen Fahrens mit dem Ziel entwickelt, eine vollständige Bewegungsplanung zu ermöglichen. Insbesondere motiviert durch die beiden Punkte 1 und 2 wird eine Erweiterung des Voronoi-Potentials dabei eine zentrale Rolle spielen. In diesem Zusammenhang wird auch eine genauere Einführung in Voronoi-Diagramme und die spezifische Definition der Voronoi-Kanten $\mathcal{R}$ gegeben, vergleiche Abschnitt 3.3.2. Zudem werden die benötigten Distanzbegriffe zur Berechnung des Potentialwertes im Detail vorgestellt.

Kapitel 3

Generische Bewegungsplanung

Der im vorigen Abschnitt 2.5 eingeführte Hybrid A*-Algorithmus wird in diesem Kapitel zu einer vollständigen Bewegungsplanung für ein autonomes Fahrzeug erweitert. Ein wesentlicher Aspekt dabei stellt die Umsetzung der in Algorithmus 2.3 noch

offen gebliebenen Kollisionsüberprüfung dar. Insbesondere dieser spezielle Aspekt weist bei der originalen Vorstellung des Verfahrens in [25] mehrere Schwächen auf:

1. **Fehlende Zeitkomponente:** In der Systemdynamik des Fahrzeugs wurden keine Geschwindigkeitskomponenten und in der Darstellung von Hindernissen in dessen Umgebung keine Bewegungsinformationen integriert. Die Planung von sicheren Trajektorien hängt aber – vor allem in urbanen Szenarien – auch stark vom (erwarteten) Verhalten anderer Verkehrsteilnehmer ab. Um dies zu berücksichtigen muss nicht nur ein zeitabhängiges Modell der Fahrzeugbewegung verwendet werden, sondern auch eine entsprechend zeitliche Auswertung der Kollisionsüberprüfung erfolgen.

2. **Spezialisierung auf geometrischen Beschränkungen:** Kollisionsfreie Trajektorien sind als einziges Kriterium von sinnvollen Fahrmanövern in der Regel nicht ausreichend. Häufig müssen einer vorgegebenen Straßenführung gefolgt oder gegebenenfalls längerfristige, strategische Ziele berücksichtigt werden. Im originalen Ansatz wird dies zum einen durch die Modifikation der Kosten sowie Heuristiken und zum anderen durch eine Erweiterung der Knotenexploration durch spezielle Makro-Aktionen, jeweils bezüglich des (bekannten) Straßennetzes, erreicht. Andere Beschränkungen, wie zum Beispiel Fahrbahnbegrenzungen (unter anderem aus Kamerabildern) ohne ein zugehöriges Straßennetz können nicht einfach integriert werden.

3. **Nachgelagerte Optimierung:** Um ein möglichst sicher ausführbares Planungsergebnis zu erreichen wird die resultierende Trajektorie anhand des Voronoi-Potentials aus Abschnitt 2.5.3 nach-optimiert. Dabei werden die tatsächliche Fahrzeuggeometrie vernachlässigt und ebenfalls lediglich statische Hindernisse betrachtet.

Insgesamt werden im ordinären Hybrid A*-Algorithmus also wichtige Aspekte der Bewegungsplanung gar nicht oder nur durch sehr spezielle Anpassungen beziehungsweise nachgelagerte Verfahren berücksichtigt. In diesem Kapitel wird vorgestellt, wie diese Nachteile durch eine generalisierte Modellierung der zuvor genannten Punkte verbessert werden können. Dazu wird in Abschnitt 3.1 zunächst mit dem Einspurmodell eine kinematische Systemdynamik für ein Fahrzeug eingeführt, welche insbesondere Geschwindigkeitsinformationen als Teil des Zustandsraumes beinhaltet. Anschließend wird eine allgemeine, abstrakte und effiziente Beschreibung von räumlichen Beschränkungen vorgestellt. Dies basiert im Kern auf der Repräsentation von sämtlichen statischen Komponenten der Umgebung durch Freibereichspolygone. Das entsprechende Konzept sowie der zugehörige Ansatz zur Berücksichtigung dynamischer Hindernisse wird in Abschnitt 3.2 betrachtet. Es wird sich zeigen, dass die Umfelddarstellung durch Polygone eine ideale Ausgangslage zur Erstellung von

Voronoi-Potentialen ist. Abschnitt 3.3 wird auf dieser Tatsache aufbauen, um eine Verallgemeinerung der Potentialdefinition aus (2.6) einzuführen, welche ebenfalls dynamische Informationen und die komplette räumliche Ausdehnung von Objekten beschreibt. Dabei werden insbesondere umfängliche Berechnungen von Distanzen zu diesen Objekten durchgeführt. Damit kann die Kollisionsüberprüfung auch als Teil der Berechnung des Potentialfeldes aufgefasst werden, was dessen direkten Einsatz im Hybrid A*-Algorithmus als Teil der Knotenkosten motiviert. Darüber hinaus stellt Abschnitt 3.3.5 einen speziellen Algorithmus zur effizienten Konstruktion der Potentialfunktion vor, welcher insbesondere für komplexe dynamische Szenarien und bei eingeschränkten Rechenkapazitäten von Vorteil ist.

Mit den im Folgenden präsentierten Methoden können, im Vergleich zu [25], nicht nur allgemeinere (dynamische) Probleme gelöst, sondern auch beliebige Arten von Hindernissen berücksichtigt werden. Durch die direkte Auswertung des Potentials während des Suchverfahrens entstehen nicht nur Synergieeffekte bezüglich der Kollisionsüberprüfung, sondern es ergibt sich auch direkt eine sehr sichere Lösung ohne die Notwendigkeit einer Nachbearbeitung. Das auf diese Weise verbesserte Verfahren wird in dieser Arbeit als *generischer Hybrid A*-Algorithmus* bezeichnet und in Abschnitt 3.4 detailliert eingeführt. Dabei werden im weiteren Verlauf – wie im ursprüngliche Hybrid A*-Algorithmus auch - die Betrachtungen auf das Lösen von Planungsproblemen in der Ebene reduziert. Zudem dient ebenfalls das autonome Fahren als Anwendungsfall; der dabei im Fokus stehende Pkw aus Abschnitt 1.2 wird im Folgenden als *Ego-Fahrzeug* bezeichnet. Grundsätzlich lassen sich die präsentierten Ansätze jedoch auch auf ähnliche Probleme (z.B. autonome Schiffsfahrt oder mobile Roboter) durch die Verwendung einer entsprechenden Systemdynamik und Umgebungsmodellierung übertragen.

3.1 Systemdynamik für autonome Fahrzeuge

Ein wesentlicher Bestandteil des in Abschnitt 2.5 vorgestellten Hybrid A*-Algorithmus besteht in der Verwendung einer Systemdynamik des Ego-Fahrzeuges zur Expansion neuer, hybrider Knoten. Um im Rahmen der Kollisionsüberprüfung `IsCollision` auch mögliche Bewegungen von Hindernissen berücksichtigen zu können, muss diese mindestens Geschwindigkeitskomponenten als zeitliche Änderung der Lageinformation enthalten. Diese Anforderung kann für den Anwendungsfall des autonomen Fahrens durch ein kinematisches *Einspurmodell* erfüllt werden, was im folgenden Abschnitt 3.1.1 vorgestellt wird. Die Vorteile dieses Modellierungsansatzes liegen vor allem in seiner Einfachheit, da zum Beispiel nur wenige, einfache Gleichungen und keine schwer messbaren Parameter enthalten sind. Zudem können optimale Pfade zwischen zwei Fahrzeugkonfigurationen unter bestimmten Bedingungen exakt berechnet werden, wie Abschnitt 3.1.2 zeigt. Dies ermöglicht insbesondere die Berechnung einer unteren Abschätzung für die Länge der Lösungstrajektorie des

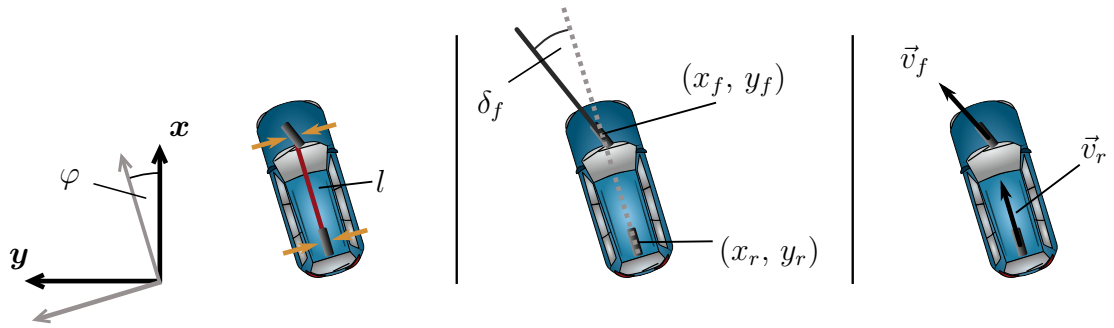


Abbildung 3.1: Modellierungsansatz des in dieser Arbeit betrachteten kinematischen Einspurmodells, nach [30]

Hybrid A*-Algorithmus und kann damit im Rahmen der benötigten Heuristik verwendet werden, vergleiche Abschnitt 3.4.
Mit der Einfachheit des kinematischen Einspurmodells ergeben sich allerdings auch Einschränkungen in Bezug auf dessen Einsatzbereich. Da keine Komponenten einer dynamischen Bewegungsmodellierung enthalten sind, wie beispielsweise durch die Berücksichtigung der auf die Reifen wirkenden Kräfte, ist dieser Modellierungsansatz allgemein nur für geringe laterale Beschleunigungen hinreichend genau [75]. Da in dieser Arbeit urbane Szenarien betrachtet werden, kann dies jedoch unter der Annahme von insgesamt niedrigen Kurvengeschwindigkeit vorausgesetzt werden.

3.1.1 Kinematisches Einspurmodell

Zu seinem Namen kommt das Einspurmodell aufgrund des speziellen Modellierungsansatzes, bei dem anstelle der zwei üblichen Reifen pro Vorder- und Hinterachse nur jeweils ein zentral verankertes Rad angenommen wird, wie in Abbildung 3.1 skizziert. Daraus lassen sich die in dieser Arbeit verwendeten Koordinaten des (zeitabhängigen) Zustandes des Ego-Fahrzeuges ableiten, welcher durch das 4-Tupel

$$z_{\text{Ego}}(t) := \Big(x_r(t),\, y_r(t),\, \varphi(t),\, v_r(t)\Big)^\top$$

repräsentiert wird. Diese beziehen sich auf ein gegebenes Koordinatensystem $(\boldsymbol{x}, \boldsymbol{y})$. Die Position des Fahrzeuges wird dabei durch einen Referenzpunkt (x_r, y_r) angegeben, welcher dem zentral gelegenen Reifen auf der Hinterachse entspricht. Dessen zeitliche Änderung wird durch den Geschwindigkeitsvektor $\vec{v}_r := \big(\dot{x}_r, \dot{y}_r\big)^\top$ beschrieben, aus welchem sich die absolute Fahrzeuggeschwindigkeit $v_r := \|\vec{v}_r\|_2$ ergibt. Die Orientierung bezüglich des gegebenen Referenzkoordinatensystems wird schließlich mit φ bezeichnet. Der Abstand zwischen Vorder- und Hinterachse heißt im Allgemeinen *Radstand* und wird durch die Konstante $l > 0$ repräsentiert. Als Einflussfaktoren

auf den Verlauf des Fahrzeugzustandes werden im Folgenden die Steuerungen

$$u_{\text{Ego}}(t) \coloneqq \big(a_r(t),\, \delta_f(t)\big)^\top$$

betrachtet, wobei $a_r \coloneqq \dot{v}_r$ die Beschleunigung der Hinterachse und δ_f den Einschlagwinkel der Vorderreifen repräsentieren. Um ein reales physikalisches System abzubilden, werden zusätzlich die Steuergrenzen $a_r^{\min} \in \mathbb{R}_{<0}$, $a_r^{\max} \in \mathbb{R}_{>0}$ sowie $\delta_f^{\max} \in \mathbb{R}_{>0}$ gemäß

$$a_r \in [a_r^{\min},\, a_r^{\max}] \quad \text{sowie} \quad \delta_f \in [-\delta_f^{\max},\, \delta_f^{\max}]$$

betrachtet, wobei die Beschränkung des Lenkwinkels seiner Natur entsprechend als symmetrisch angenommen wird.

Zur Beschreibung der Systemdynamik als System gewöhnlicher Differentialgleichungen für $\dot{z}_{\text{Ego}}$ wird im weiteren Verlauf ein wohlbekannter kinematischer Ansatz verwendet, wie er zum Beispiel in [65, 82] vorgestellt wird. Bei diesem werden unter anderem sämtliche lateral auf die Reifen wirkenden Kräfte vernachlässigt, was für die Annahme insgesamt moderater Querbeschleunigungen im Rahmen einer Bewegungsplanung zu akzeptablen Ergebnissen führt [75]. Auf dieser Grundlage kann die Positionsänderung direkt aus Geschwindigkeit und aktueller Ausrichtung gemäß

$$\begin{pmatrix} \dot{x}_r \\ \dot{y}_r \end{pmatrix} = \begin{pmatrix} v_r \cos(\varphi) \\ v_r \sin(\varphi) \end{pmatrix} \tag{3.1}$$

abgeleitet werden. Für die Frontachse gilt mit der Absolutgeschwindigkeit $v_f \coloneqq \|\vec{v}_f\|_2$ analog

$$\begin{pmatrix} \dot{x}_f \\ \dot{y}_f \end{pmatrix} = \begin{pmatrix} v_f \cos(\varphi + \delta_f) \\ v_f \sin(\varphi + \delta_f) \end{pmatrix}.$$

Wird die rechte Seite mit dem hierzu orthogonalen Vektor $\big(-\dot{y}_f,\, \dot{x}_f\big)^\top$ skalarmultipliziert, so folgt

$$0 = v_f \left(\dot{x}_f \sin(\varphi + \delta_f) - \dot{y}_f \cos(\varphi + \delta_f)\right).$$

Es wird zunächst der Fall $v_f \neq 0$ eines nicht stehenden Fahrzeugs betrachtet, für welchen

$$0 = \dot{x}_f \sin(\varphi + \delta_f) - \dot{y}_f \cos(\varphi + \delta_f) \tag{3.2}$$

folgt. Für die Vorderachsenposition gilt zudem

$$\begin{pmatrix} \dot{x}_f \\ \dot{y}_f \end{pmatrix} = \begin{pmatrix} \frac{\mathrm{d}}{\mathrm{d}t}(x_r + l\cos(\varphi)) \\ \frac{\mathrm{d}}{\mathrm{d}t}(y_r + l\sin(\varphi)) \end{pmatrix} = \begin{pmatrix} \dot{x}_r - \dot{\varphi} l \sin(\varphi) \\ \dot{y}_r + \dot{\varphi} l \cos(\varphi) \end{pmatrix}, \tag{3.3}$$

was gemeinsam mit (3.2) in

$$\begin{aligned} 0 &= \dot{x}_r \sin(\varphi + \delta_f) - \dot{\varphi} l \sin(\varphi) \sin(\varphi + \delta_f) - \dot{y}_r \cos(\varphi + \delta_f) - \dot{\varphi} l \cos(\varphi) \cos(\varphi + \delta_f) \\ &= \dot{x}_r \sin(\varphi + \delta_f) - \dot{y}_r \cos(\varphi + \delta_f) - \dot{\varphi} l \left(\sin(\varphi) \sin(\varphi + \delta_f) + \cos(\varphi) \cos(\varphi + \delta_f)\right) \\ &= \dot{x}_r \sin(\varphi + \delta_f) - \dot{y}_r \cos(\varphi + \delta_f) - \dot{\varphi} l \cos(\delta_f) \end{aligned}$$

resultiert. Unter Berücksichtigung der Gleichung aus (3.1) folgt dann direkt

$$\begin{aligned} 0 &= v_r \left(\cos(\varphi) \sin(\varphi + \delta_f) - \sin(\varphi) \cos(\varphi + \delta_f)\right) - \dot{\varphi} l \cos(\delta_f) \\ &= v_r \sin(\delta_f) - \dot{\varphi} l \cos(\delta_f). \end{aligned}$$

Unter der Annahme, dass für ein reales Fahrzeug $\delta_f^{\max} < \pi/2$ gilt, also die Vorderräder nicht weiter als 90° eingeschlagen werden können, kann dies schließlich zu

$$\dot{\varphi} = \frac{v_r}{l} \tan(\delta_f)$$

umgestellt werden. Damit sind Differentialgleichungen für alle Zustandskomponenten bekannt und lassen sich als Systemdynamik

$$\dot{z}_{\text{Ego}}(t) = \begin{pmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\varphi}(t) \\ \dot{v}_r(t) \end{pmatrix} = \begin{pmatrix} v_r(t) \cos(\varphi(t)) \\ v_r(t) \sin(\varphi(t)) \\ \frac{v_r(t)}{l} \tan(\delta_f(t)) \\ a_r(t) \end{pmatrix} \tag{3.4}$$

abhängig von den Steuergrößen $u_{\text{Ego}}(t)$ zusammenfassen. Für den Spezialfall eines stehenden Fahrzeuges $v_f = v_r = 0$ folgt aus (3.1) und (3.3) zudem direkt $\dot{x}_r(t) = \dot{y}_r(t) = \dot{\varphi}(t) = 0$.

Die Zustände $z_{\text{Ego}}(t)$ genügen als Lösung dieses Differentialgleichungssystems Anforderungen an ihre Differenzierbarkeit. Für die Steuergrößen wird zur Verwendung im Hybrid A*-Algorithmus dagegen lediglich stückweise Stetigkeit gefordert, vergleiche auch Definition 2.33. Dies erlaubt in diesem Fall spontane Sprünge der Lenkwinkel und Beschleunigungen. Im Rahmen eines Bewegungsplaners kann dieses Verhalten gegebenenfalls akzeptiert werden, wenn die Steuerungen für das Ego-Fahrzeug nicht direkt auf deren Trajektorien basieren, sondern durch einen nachgelagerten Regler bereitgestellt werden, wie für das in Abschnitt 1.2 dargestellte OPA^{3}L-System. Der folgende Abschnitt zeigt allerdings, wie insbesondere die Unstetigkeit des Lenkwinkels δ_f zur Berechnung optimaler Pfade für das Einspurmodell ausgenutzt werden kann.

3.1.2 Optimale Pfade

Um die Länge einer berechneten Trajektorie zwischen einem Start- und Zielzustand $z^{(\text{start})}$ und $z^{(\text{end})}$ im Rahmen der Kosten des Hybrid A*-Algorithmus 2.3 bestrafen

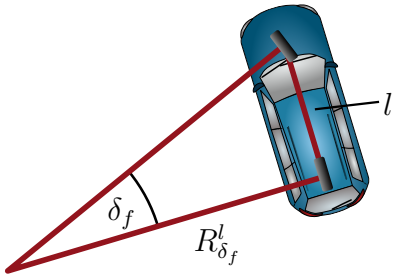


Abbildung 3.2: Momentanradius eines Fahrzeuges mit Lenkwinkel δ_f und Radstand l gemäß des Einspurmodells aus Abschnitt 3.1.1

und damit minimieren zu können, wird eine zugehörige Heuristikfunktion benötigt. Um das Finden einer optimalen Lösung zu garantieren, sollte diese die Monotonieeigenschaft aus Definition 2.27 erfüllen, und damit insbesondere auch eine untere Abschätzung der tatsächlich verbleibenden Länge von einem gegebenen Zustand $z \in \mathcal{Z}$ zum Ziel darstellen. Diese ergibt sich im Wesentlichen aus den Bewegungsmöglichkeiten, die aus der verwendeten Systemdynamik resultieren. Im weiteren Verlauf der Arbeit wird für diese das Differentialgleichungssystem aus (3.4) angenommen. Eine besonders einfache Abschätzung ist durch die euklidische Heuristik $h_{\text{eukl}}(z) = \|z^{(\text{end})} - z\|_2$ gegeben, vergleiche Abschnitt 2.4.1. Dieser sehr generische Ansatz vernachlässigt jedoch die konkreten Eigenschaften der vorliegenden Dynamik, woraus eine vergleichsweise schwache Abschätzung und damit ein aufwändigeres Suchverfahren resultiert, wie in Abschnitt 2.4.1 exemplarisch dargestellt.

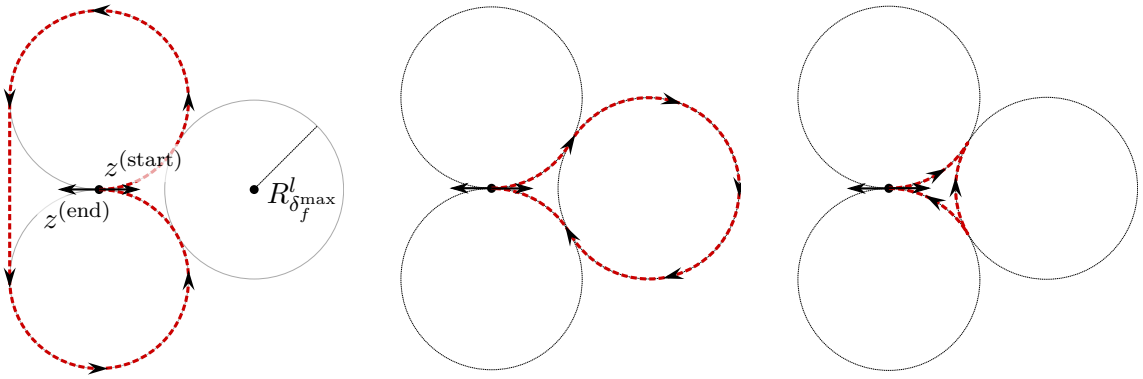
Bessere Abschätzungen, speziell basierend auf dem Einspurmodell, können mit den optimalen Pfaden von Dubins [27] oder Reeds und Shepp [79] erreicht werden. Beide Ansätze liefern Lösungen für Trajektorien mit minimaler geometrischer Länge zwischen $z^{(\text{start})}$ und $z^{(\text{end})}$, welche die Gleichungen aus (3.4) erfüllen. Diese berücksichtigen jedoch explizit keine räumlichen Einschränkungen wie etwa durch das Vorhandensein von Hindernissen. Zudem werden zeitliche Komponenten vernachlässigt, weshalb hier die Geschwindigkeit des Ego-Fahrzeugs nicht betrachtet wird. Kann die Bewegungsrichtung des Fahrzeugs schließlich auf *vorwärts* reduziert werden, dann zeigt der Ansatz von Dubins, dass der Pfad optimaler Länge sich stets als Komposition aus drei möglichen Bewegungsprimitiven ergibt:

(L): Fahren nach links mit maximalem Lenkwinkel $\delta_f = \delta_f^{\max}$.

(R): Fahren nach rechts mit maximalem Lenkwinkel $\delta_f = -\delta_f^{\max}$.

(S): Fahren ohne Lenkradeinschlag mit $\delta_f = 0$.

Diese Primitive entsprechen entweder Geraden oder Kreisabschnitten mit dem minimal möglichen fahrbaren Radius des Fahrzeugs $R^l_{\delta_f^{\max}}$ gemäß des Einspurmodells.



(a) Nicht optimaler (LSL)-Pfad nach Dubins

(b) Optimaler (LRL)-Pfad nach Dubins

(c) Optimaler Pfad nach Reeds und Shepp

Abbildung 3.3: Exemplarische Darstellung optimaler und nicht optimaler Pfade für einen Richtungswechsel auf der Stelle gemäß Dubins oder Reeds und Shepp, adaptiert aus [79]

Letzterer ergibt sich wie in Abbildung 3.2 dargestellt aus

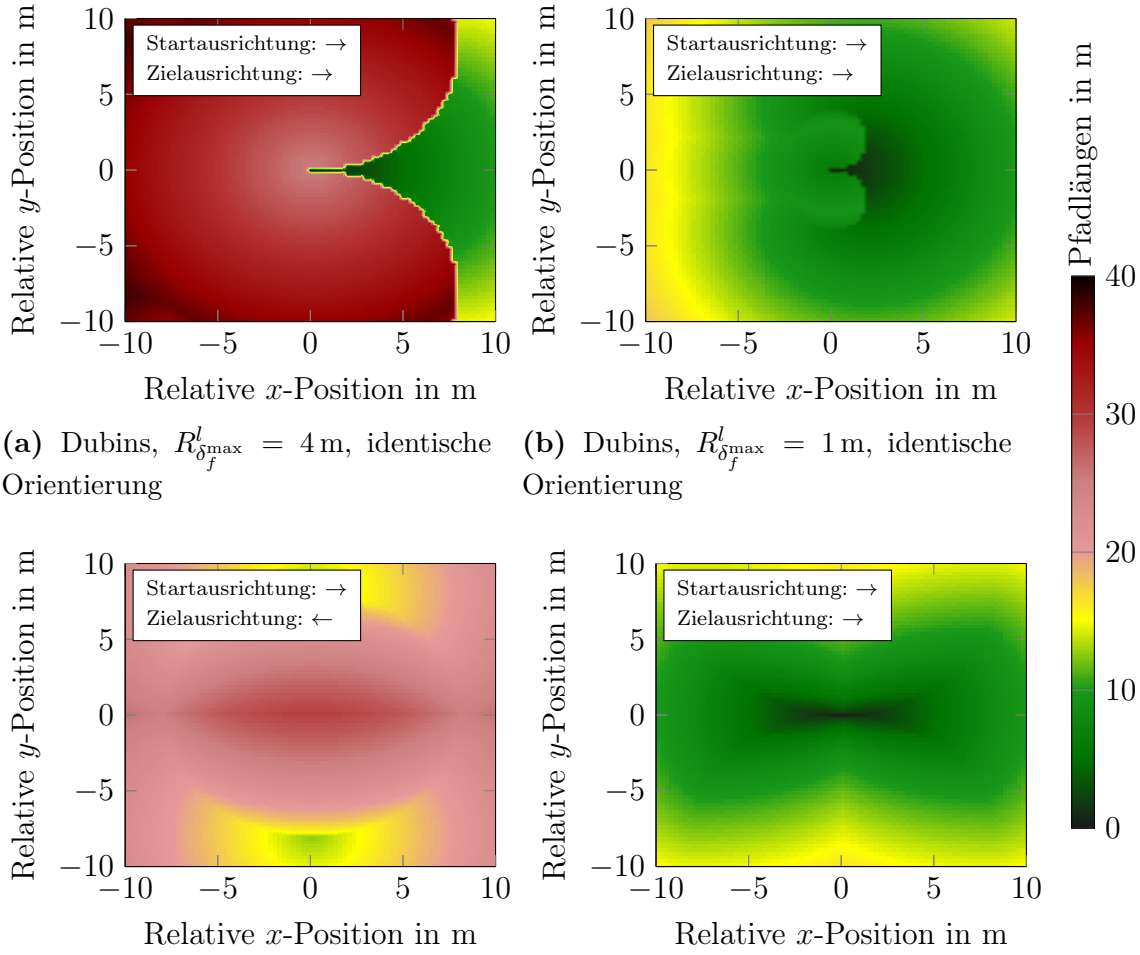
$$R^l_{\delta_f^{\max}} = \frac{l}{\tan(\delta_f^{\max})}.$$

Zwei konsekutive Abschnitte dieser Bahn repräsentieren dabei stets zwei verschiedene Primitive, da diese sonst zu einem Einzigen zusammengefasst werden könnten. Von den zehn verbleibenden Kombinationen dreier Abschnitte können die folgenden Sechs zu einem Pfad minimaler Länge führen [27]:

(LRL), (RLR), (LSL), (RSR), (LSR), (RSL).

Für jede Kombination wird die Länge der entsprechenden Abschnitte durch einfache geometrische Auswertungen bestimmt; dabei kann einzelnen Primitiven auch eine Länge von 0 zugeordnet werden. Für zwei beliebige Fahrzeugzustände kann so durch Vergleich aller sechs möglichen Pfade effizient und direkt die Lösung kürzester Länge bestimmt werden. Eine nicht optimale sowie eine optimale Kombination der Bewegungsprimitive für das Beispiel des Richtungswechsels auf der Stelle sind in den Abbildungen 3.3a und 3.3b gezeigt. Aus letzterer ergibt sich insbesondere direkt, dass die optimale Lösung nicht eindeutig gegeben sein muss: neben dem gezeigten (LRL)-Pfad hat die gespiegelte (RLR)-Lösung ebenfalls minimale Länge.

Eine Verallgemeinerung dieses Ansatzes stellt das Verfahren von Reeds und Shepp dar, welches ebenfalls optimale Pfade für das Einspurmodell berechnet, im Gegensatz zum Vorigen jedoch auch Wechsel in der Fahrrichtung zulässt. Abbildung 3.3c zeigt die entsprechende optimale Lösung für den Richtungswechsel auf der Stelle: der zusätzliche Freiheitsgrad wird dabei ausgenutzt, sodass sie im Vergleich zu Dubins-Pfad wesentlich kürzer ist. Die Berechnung reduziert sich analog auf die Analyse



(a) Dubins, $R^l_{\delta^{\max}_f} = 4\,\mathrm{m}$, identische Orientierung

(b) Dubins, $R^l_{\delta^{\max}_f} = 1\,\mathrm{m}$, identische Orientierung

(c) Dubins, $R^l_{\delta^{\max}_f} = 4\,\mathrm{m}$, entgegengesetzte Orientierung

(d) Reeds-Shepp, $R^l_{\delta^{\max}_f} = 4\,\mathrm{m}$, identische Orientierung

Abbildung 3.4: Längen der optimalen Pfade nach Dubins beziehungsweise Reeds und Shepp zu relativen Positionen des Zielzustandes, abhängig vom kleinsten fahrbaren Radius $R^l_{\delta^{\max}_f}$ und der relativen Orientierung

von Kombinationen aus Bewegungsprimitiven, welche in diesem Fall in insgesamt 48 möglichen Gruppen* resultieren. Um die Anzahl an Überprüfungen zu reduzieren und das Verfahren effizienter zu machen, können dabei für einen gegebenen relativen Zielzustand durch geometrische Auswertungen, basierend auf dem minimalen Fahrzeugradius, jeweils einige dieser möglichen Lösungskandidaten im Vorfeld ausgeschlossen werden [92].

Die Methoden von Dubins oder Reeds und Shepp werden in Abschnitt 3.4 als Teil der Definition von Heuristik und Kosten genutzt, um eine untere Abschätzung für die Entfernung einer gegebenen Fahrzeugkonfiguration zum Ziel zu berechnen. Aufgrund

*Es konnte später gezeigt werden, dass tatsächlich 46 Gruppen ausreichend sind [94].

der verwendeten Kinematik und der Berücksichtigung der jeweiligen Orientierungen stellen sie eine deutlich genauere Approximation dar als zum Beispiel die euklidische Distanz, welche lediglich eine Aussage über die örtliche relative Lage zwischen Fahrzeugkonfigurationen zulässt. Umgekehrt weisen sie allerdings auch Diskontinuitäten in ihrem Verlauf auf. Dies lässt sich gut anhand der Beispiele aus Abbildung 3.4 beobachten. Da durch den minimal fahrbaren Kurvenradius die modellierte Manövrierfähigkeit eingeschränkt ist, kann schon eine kleine Verschiebung in den betrachteten Konfigurationen zu völlig unterschiedlichen Bewegungsprimitiven und zugehörigen Pfadlängen führen. Diese Nichtstetigkeit ergibt sich für jedes $R^l_{\delta_f^{\max}} > 0$, wobei die Übergänge für kleinere Radien weicher und die Entfernungen niedriger werden, wie exemplarisch im direkten Vergleich zwischen Abbildungen 3.4a und 3.4b erkennbar ist. Wird die relative Orientierung des Zielzustandes variiert, so treten analog Diskontinuitäten auf, deren Lage sowie der gesamte Verlauf sich jedoch entsprechend verändern, vergleiche Abbildung 3.4c. Auch für die Pfade nach Reeds und Shepp treten vergleichbare Effekte durch die Beschränkung des Fahrradius auf, allerdings (ähnlich wie bei einem verringerten Lenkwinkel) mit deutlich niedrigeren Entfernungen aufgrund der zusätzlichen Freiheit, rückwärts zu fahren. In Abbildung 3.4d ist zudem deutlich zu erkennen, dass auch Ziele hinter der Startposition wesentlich kürzer erreicht werden können als durch Dubins Pfade.

3.2 Adaptive räumliche Beschränkungen

Grundlage einer präzisen und effizienten Trajektorienplanung ist eine entsprechende Darstellung aller relevanter Umgebungsinformationen, durch welche die räumlichen Bewegungsmöglichkeiten des betrachteten Systems eingeschränkt werden. Im weiteren Verlauf wird in diesem Kontext vor allem zwischen dynamischen Objekten und statischen Beschränkungen unterschieden. Dabei wird für ein dynamisches Objekt angenommen, dass Informationen über dessen erwartete Bewegung und Größe (mindestens bestehend aus geschätzter Länge und Breite) vorliegen. Statische Einschränkungen können dagegen generell in beliebiger Darstellung gegeben sein. Diese Arbeit reduziert sich jedoch darauf, zusammenhängende Strukturen durch Polygone und alle anderen Daten in einer Punktwolke $D \subset \mathbb{R}^2$ zu berücksichtigen. Alle tatsächlich relevanten Bestandteile dieser statischen Informationen werden schließlich durch ein sogenanntes Freibereichspolygon um das Fahrzeug herum approximiert. Abbildung 3.5 zeigt dies für zwei verschiedene Planungshorizonte.

Die Reduktion der vorhandenen Daten auf diese Weise hat verschiedene Vorteile. Diese haben bereits in vorigen Arbeiten die Verwendung von Freibereichspolygonen motiviert, zum Beispiel zur Definition von Optimalsteuerungsproblemen für autonome Roboter [66] oder Fahrzeuge [90]. Dabei sind vor allem die folgenden Aspekte zu nennen:

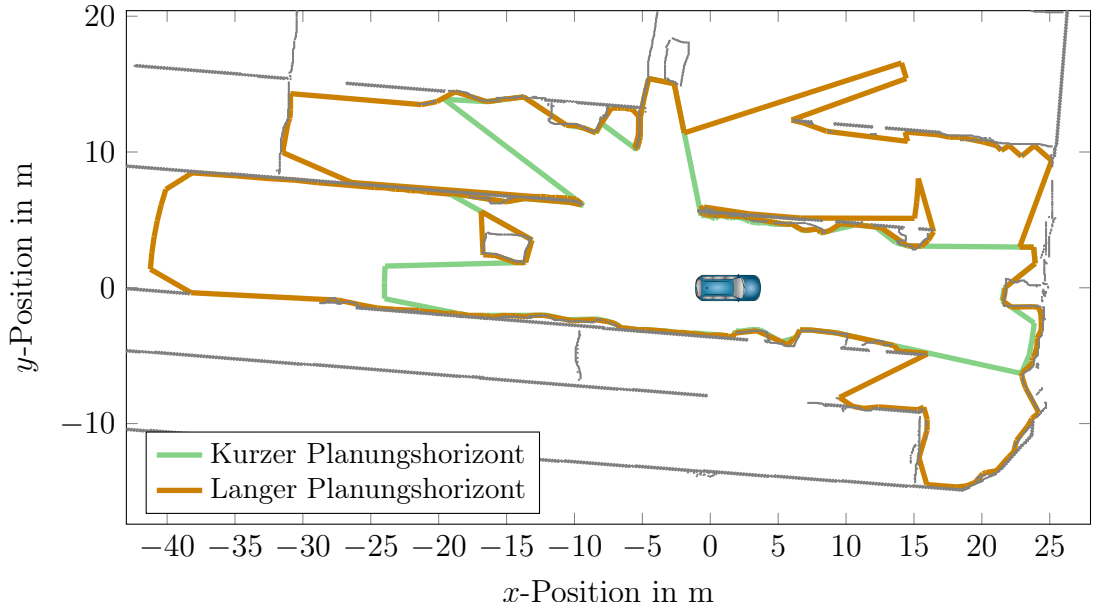


Abbildung 3.5: Exemplarische Freibereichspolygone für unterschiedliche Planungshorizonte im komplexen Umfeld des gezeigten Fahrzeuges, gegeben als schwarze Punktwolke mit 7376 Elementen

1. Insbesondere im Vergleich zu großen Punktwolken stellen Freibereichspolygone eine kleine Datenstruktur dar, die eine entsprechend effiziente Kollisionsüberprüfung ermöglicht.

2. Die Genauigkeit der Approximation durch ein Freibereichspolygon lässt sich im Rahmen von dessen Berechnung fein justieren. Je höher die Auflösung, desto aufwändiger ist jedoch auch die Erstellung und Auswertung zur Kollisionsüberprüfung.

3. Es müssen nur Beschränkungen berücksichtigt werden, die innerhalb des Planungshorizontes liegen; dies lässt sich ebenfalls während der Erstellung berücksichtigen, um so die Größe des Freibereichspolygons adaptiv anzupassen.

4. Es lassen sich beliebige Arten statischer Beschränkungen abbilden.

Speziell für den Einsatz im generischen Hybrid A*-Algorithmus sind zudem die folgenden beiden Punkte relevant.

5. Für das Innere eines Freibereichspolygons lässt sich sehr effektiv das zugehörige Voronoi-Diagramm als Grundlage des entsprechenden Potentialfeldes berechnen. Dies wird in Abschnitt 3.3.2 genauer beschrieben.

6. Ein (geschlossenes) Freibereichspolygon ermöglicht relativ kanonisch die Konstruktion einer endlichen diskreten Knotenmenge, was eine grundlegende Vor-

aussetzung für die Optimalität und Vollständigkeit des Hybrid A*-Algorithmus darstellt, vergleiche Satz 2.39 sowie Anmerkung 2.42.

Der folgende Abschnitt 3.2.1 stellt die geometrischen Grundlagen vor, unter anderem zur Definition von Polygonen allgemein sowie des Freibereichspolygons im Speziellen. In diesem Kontext wird insbesondere auch ein Distanzbegriff eingeführt, welcher die Grundlage für spätere Kollisionsberechnungen darstellt. Darauf aufbauend präsentiert Abschnitt 3.2.2 ein Verfahren, um Freibereichspolygone für beliebige Umgebungen adaptiv zu erstellen. Die Laufzeitkosten zur Ausführung dieses Algorithmus wird anschließend in Abschnitt 3.2.3 gegen die Vorteile zur Berechnung von Kollisionsüberprüfungen abgewogen. Abschließend wird in Abschnitt 3.2.4 kurz dargestellt, wie die Umgebungsbeschreibung mit Freibereichspolygonen auf allgemeine dynamische Szenarien erweitert werden kann.

3.2.1 Beschreibung von geometrischen Objekten

Das Ziel dieses Abschnittes ist die Formulierung von Grundlagen zur Beschreibung von geometrischen Strukturen, welche im Folgenden unter anderem zur Approximation von Umgebungsinformationen in Planungsaufgaben in der Ebene genutzt werden. Dazu werden aufeinander aufbauend zunächst Punkte, Liniensegmente und schließlich Polygone eingeführt. Dabei steht insbesondere die Entwicklung von Distanzfunktionen zwischen diesen Objekten im Fokus. Der Abschnitt schließt mit der Definition von Freibereichspolygonen, welche eine wesentliche Komponente aller folgenden Betrachtungen darstellen werden.

3.2.1.1 Punkte

Die grundlegendste in dieser Arbeit verwendete Struktur ist ein Punkt.

Definition 3.1 (Punkt)

> Ein Element $q \in \mathbb{R}^2$ wird als *Punkt* bezeichnet. Für diesen beschreiben q_x und q_y dessen x- beziehungsweise y-Koordinate bezüglich des zugrunde liegenden Koordinatensystems.

Distanzen zwischen zwei Punkten werden im Folgenden stets basierend auf der euklidischen Norm gemäß

$$\begin{aligned} \mathrm{d}\colon \mathbb{R}^2 \times \mathbb{R}^2 &\to \mathbb{R}_{\geq 0}, \\ (q_1,\, q_2) &\mapsto \|q_2 - q_1\|_2 \end{aligned}$$

berechnet. Um Vektoren zwischen Punkten miteinander vergleichen zu können, wird darüber hinaus im weiteren Verlauf das Standardskalarprodukt

$$a \cdot b \coloneqq a_x b_x + a_y b_y \in \mathbb{R}, \quad \text{für } a, b \in \mathbb{R}^2$$

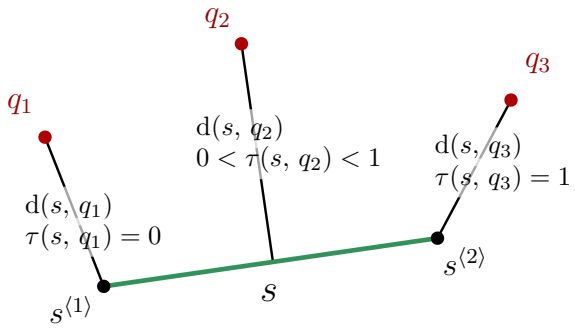


Abbildung 3.6: Beispiele zur Berechnung von Distanzen zwischen einem Liniensegment $s \in \mathcal{S}$ (grün) und verschiedenen Punkten $q_i \in \mathbb{R}^2$ (rot)

sowie das Kreuzprodukt verwendet. Für Letzteres können zweidimensionale Elemente $a, b \in \mathbb{R}^2$ durch

$$\begin{pmatrix} a_x \\ a_y \\ 0 \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ 0 \end{pmatrix} := \begin{pmatrix} 0 \\ 0 \\ a_x b_y - a_y b_x \end{pmatrix}$$

in den $\mathbb{R}^3$ eingebettet werden. In diesem Kontext wird $\times_2$ als Einschränkung auf die nicht triviale Koordinate gemäß

$$a \times_2 b := a_x b_y - a_y b_x \in \mathbb{R}, \quad \text{für } a, b \in \mathbb{R}^2$$

definiert.

3.2.1.2 Liniensegmente

Zwei Punkte lassen sich durch ein Liniensegment linear verbinden.

Definition 3.2 (Liniensegment)

Sei $s \subset \mathbb{R}^2$ und es gelte für $s^{\langle 1 \rangle}, s^{\langle 2 \rangle} \in \mathbb{R}^2$

$$s = \left\{ s^{\langle 1 \rangle} + \tau \cdot (s^{\langle 2 \rangle} - s^{\langle 1 \rangle}) \,\middle|\, \tau \in [0, 1] \right\},$$

dann heißt s *Liniensegment* und wird durch seine Endpunkte mit $(s^{\langle 1 \rangle}, s^{\langle 2 \rangle})_{\mathcal{S}} := s$ parametrisiert. Die Menge aller Liniensegmente wird mit

$$\mathcal{S} := \left\{ (s^{\langle 1 \rangle}, s^{\langle 2 \rangle})_{\mathcal{S}} \,\middle|\, s^{\langle 1 \rangle}, s^{\langle 2 \rangle} \in \mathbb{R}^2 \right\}$$

bezeichnet. Die *Potenzmenge* aller Liniensegmente wird im weiteren Verlauf $\Pi(\mathcal{S})$ genannt.

Im Folgenden wird die Kategorisierung der Lage von Liniensegmenten zueinander hilfreich sein, wobei insbesondere Parallelität und Orthogonalität zwei wichtige Spezialfälle darstellen.

Definition 3.3 (Parallelität und Orthogonalität von Liniensegmenten)

Betrachte zwei Liniensegmente $s_1 = (s_1^{\langle 1\rangle}, s_1^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{S}$ sowie $s_2 = (s_2^{\langle 1\rangle}, s_2^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{S}$.

1. Es gilt
$$s_1 \perp s_2 \iff \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right) \cdot \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right) = 0.$$
Die Segmente $s_1 \perp s_2$ heißen *orthogonal* zueinander.

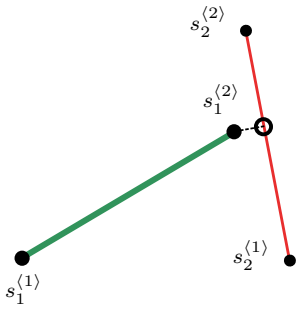
2. Es gilt
$$s_1 \parallel s_2 \iff \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right) \times_2 \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right) = 0.$$
Die Segmente $s_1 \parallel s_2$ heißen *parallel* zueinander. Gilt keine Parallelität, so wird dies durch $s_1 \nparallel s_2$ ausgedrückt.

Dabei bezeichnen $a \cdot b$ das Standardskalarprodukt und $a \times_2 b$ das eingeschränkte Kreuzprodukt für $a, b \in \mathbb{R}^2$, vergleiche Abschnitt 3.2.1.1.

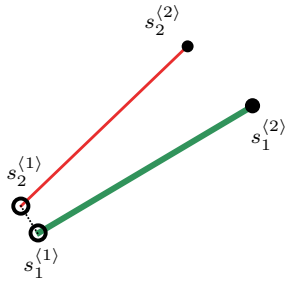
Die Berechnung von Distanzen zu einem Liniensegment $s = (s^{\langle 1\rangle}, s^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{S}$ wird zunächst ausgehend von einem Punkt $q \in \mathbb{R}^2$ betrachtet. Dazu wird die Projektion τ im Sinne von Definition 3.2 durch
$$\tau(s, q) \coloneqq \frac{(q - s^{\langle 1\rangle}) \cdot (s^{\langle 2\rangle} - s^{\langle 1\rangle})}{\mathrm{d}(s^{\langle 2\rangle} - s^{\langle 1\rangle})} \in \mathbb{R}$$
bestimmt.Wie in Abbildung 3.6 skizziert entspricht dies einer Zahl in $(0, 1)$, falls die Projektion im Inneren des Segmentes liegt. Ist dagegen $s^{\langle 1\rangle}$ der naheste Segmentpunkt, so gilt $\tau \leq 0$; für $s^{\langle 2\rangle}$ ergibt sich analog $\tau \geq 1$. Damit berechnet sich die Distanz $\mathrm{d}(s, q)$ zwischen Liniensegment s und Punkt q nach [29] durch
$$\mathrm{d}\colon \mathcal{S} \times \mathbb{R}^2 \to \mathbb{R}_{\geq 0},$$
$$\mathrm{d}(s, q) \coloneqq \begin{cases} \mathrm{d}(q - s^{\langle 1\rangle}), & \tau(s, q) \leq 0, \\ \mathrm{d}(q - s^{\langle 2\rangle}), & \tau(s, q) \geq 1, \\ \mathrm{d}(q - (s^{\langle 1\rangle} + \tau(s, q) \cdot (s^{\langle 2\rangle} - s^{\langle 1\rangle}))), & \text{sonst.} \end{cases}$$

Insbesondere in Abschnitt 3.3 wird zudem die Distanz zwischen zwei Liniensegmenten benötigt. Für den Fall von zwei bis ins Unendliche verlängerte Linien sind zur Distanzberechnung zwischen ihnen genau drei Fälle zu beachten:

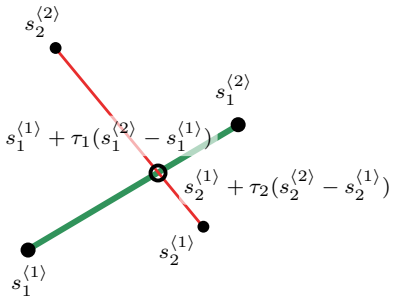
1. Sie treffen sich in genau einem Punkt,
2. sie verlaufen parallel und haben keinen Schnittpunkt oder
3. sie verlaufen parallel und liegen übereinander, sodass sie sich in jedem ihrer Punkte schneiden.



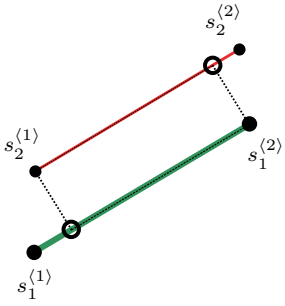
(a) Kürzeste Distanz zwischen Innerem und Randpunkt



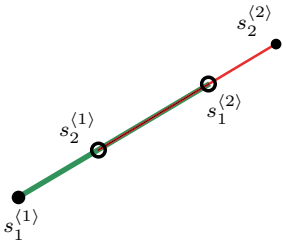
(b) Kürzeste Distanz zwischen Randpunkten



(c) Schneidend



(d) Parallel



(e) Überlappend

Abbildung 3.7: Exemplarische Darstellung verschiedener Fälle bei der Distanzberechnung zwischen zwei Liniensegmenten $s_1 \in \mathcal{S}$ (grün) und $s_2 \in \mathcal{S}$ (rot)

Diese übertragen sich auch auf Liniensegmente, vergleiche Abbildungen 3.7c bis 3.7e. Aus deren Beschränktheit ergibt sich allerdings, dass überlappende Segmente sich im Allgemeinen nur in einer Teilmenge ihrer jeweiligen Punkte decken. Zudem resultiert ein vierter Fall, in dem sie sich nicht schneiden, allerdings jeweils einen eindeutigen Punkt haben, durch welchen die Distanz zum Vergleichssegment minimiert wird. Für mindestens eines der Segmente liegt dieser Punkt stets auf dem Rand, vergleiche Abbildung 3.7a. Als Spezialfall kann die Situation in Abbildung 3.7b betrachtet werden, bei dem das Minimum auf dem Rand beider Segmente angenommen wird.

Für zwei nicht parallele Liniensegmente $s_1 \nparallel s_2$ mit $s_1 = (s_1^{\langle 1\rangle}, s_1^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{S}$ und $s_2 = (s_2^{\langle 1\rangle}, s_2^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{S}$ gibt es zwei eindeutige Skalare $\tau_1, \tau_2 \in [0, 1] \subset \mathbb{R}$, sodass

$$s_1^{\langle 1\rangle} + \tau_1 \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right) = s_2^{\langle 1\rangle} + \tau_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right)$$

gilt, siehe Abbildung 3.7c. Nach Anwendung des Kreuzprodukts $\times_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right)$ resultiert daraus

$$\left(s_1^{\langle 1\rangle} + \tau_1 \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right)\right) \times_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right) = \left(s_2^{\langle 1\rangle} + \tau_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right)\right) \times_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right),$$

womit sich mit $s \times_2 s = 0$ direkt

$$\tau_1 = \frac{\left(s_2^{\langle 1\rangle} - s_1^{\langle 1\rangle}\right) \times_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right)}{\left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right) \times_2 \left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right)} \tag{3.5}$$

ergibt. Durch analoges Vorgehen folgt darüber hinaus

$$\tau_2 = \frac{\left(s_1^{\langle 1\rangle} - s_2^{\langle 1\rangle}\right) \times_2 \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right)}{\left(s_2^{\langle 2\rangle} - s_2^{\langle 1\rangle}\right) \times_2 \left(s_1^{\langle 2\rangle} - s_1^{\langle 1\rangle}\right)}. \tag{3.6}$$

Die Bedingung $\tau_1, \tau_2 \in [0, 1]$ kann als Kriterium für den Schnitt zweier nicht paralleler Liniensegmente verwendet werden, für den sich unmittelbar ein Abstand von 0 ergibt. Liegt kein Schnittpunkt vor, so kann die Distanz zwischen zwei Liniensegmenten grundsätzlich anhand des minimalen Abstandes aller vier Randpunkte mit dem jeweils anderen Vergleichssegment bestimmt werden. Dies gilt auch für Spezialfälle wie Parallelität oder Überlappung, vergleiche Abbildungen 3.7d und 3.7e. Damit ergibt sich die Distanz zwischen zwei Liniensegmenten durch

$$\begin{aligned} &\mathrm{d}\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \\ &(s_1,\, s_2) \mapsto \begin{cases} 0, & \text{falls } s_1 \nparallel s_2 \text{ und } \tau_1,\, \tau_2 \in [0, 1], \\ \min\left(\mathrm{d}(s_1, s_2^{\langle 1\rangle}),\, \mathrm{d}(s_1, s_2^{\langle 2\rangle}),\, \mathrm{d}(s_2, s_1^{\langle 1\rangle}),\, \mathrm{d}(s_2, s_1^{\langle 2\rangle})\right), & \text{sonst,} \end{cases} \end{aligned} \tag{3.7}$$

wobei τ_1, τ_2 für $s_1 \nparallel s_2$ gemäß (3.5) und (3.6) berechnet werden.

Anmerkung 3.4 Für eine effizientere Berechnung bei sich nicht schneidenden und nicht parallelen Liniensegmenten kann anhand der Werte von τ_1 und τ_2 auf die minimierenden Elemente geschlossen werden, wie anhand der folgenden Beispiele gezeigt wird:

- Für $\tau_1 > 0$, $\tau_2 \in [0, 1]$ gilt $\mathrm{d}(s_1,\, s_2) = \mathrm{d}(s_1^{\langle 1\rangle},\, s_2)$, vergleiche Abbildung 3.7a.
- Für $\tau_1 < 0$, $\tau_2 < 0$ gilt $\mathrm{d}(s_1,\, s_2) = \mathrm{d}(s_1^{\langle 1\rangle},\, s_2^{\langle 1\rangle})$, vergleiche Abbildung 3.7b.

3.2.1.3 Polygone

Ein Polygon lässt sich als eine geschlossene Verkettung von Liniensegmenten darstellen.

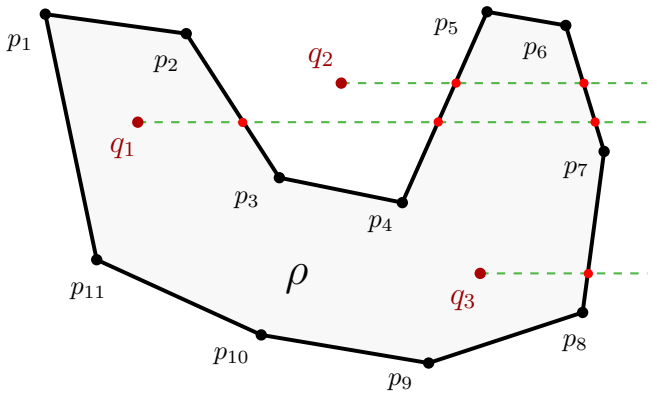


Abbildung 3.8: Anwendung des Jordanschen Kurvensatzes zur Lagebestimmung von verschiedenen Testpunkten $q_i \in \mathbb{R}^2$ (rot) bezüglich eines Polygons $\rho \in \Lambda$

Definition 3.5 (Polygonzug und Polygon)

Sei $\{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ eine Menge von Punkten, dann beschreibt

$$\rho = \{(p_i,\, p_{i+1})_{\mathcal{S}} \mid i = 1, \ldots, n\} \in \Pi(\mathcal{S})$$

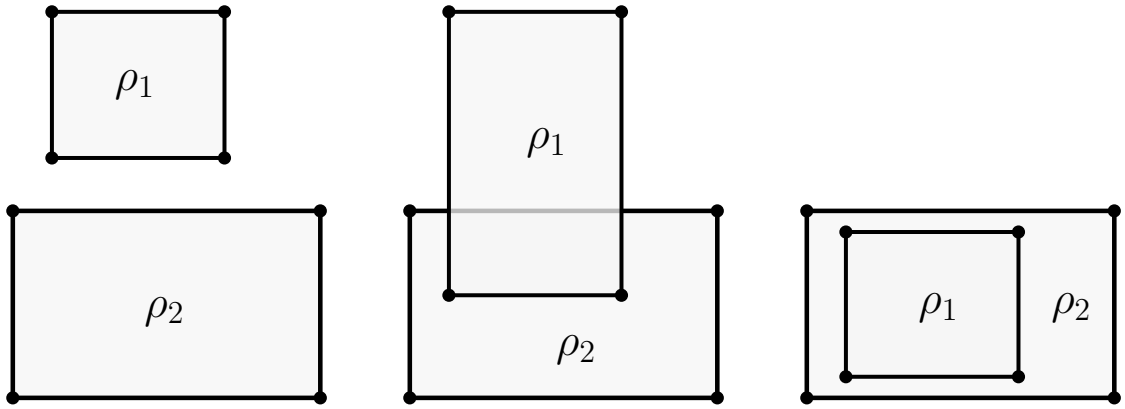
einen *Polygonzug*. Sind der erste und letzte Punkt identisch und nicht alle resultierenden Liniensegmente parallel, dann entspricht ein Polygonzug einer geschlossenen Figur. Folglich heißt dann

$$\Lambda := \left\{ \rho = \bigcup_{i=1,\ldots,n-1} (p_i,\, p_{i+1})_{\mathcal{S}} \,\middle|\, \{p_1, \ldots, p_n\} \subset \mathbb{R}^2,\, n \in \mathbb{N},\, 3 \leq n < \infty,\, p_1 = p_n, \right.$$
$$\Big(\exists i \in \{2, \ldots, n-1\} \colon (p_{i-1},\, p_i)_{\mathcal{S}} \nparallel (p_i,\, p_{i+1})_{\mathcal{S}} \text{ oder}$$
$$\left. (p_{n-1},\, p_n)_{\mathcal{S}} \nparallel (p_n,\, p_1)_{\mathcal{S}} \text{ oder } (p_n,\, p_1)_{\mathcal{S}} \nparallel (p_1,\, p_2)_{\mathcal{S}} \Big) \right\}$$

Menge aller *geschlossener Polygonzüge*. Die geometrische Figur, welche durch einen geschlossenen Polygonzug beschrieben wird, heißt *Polygon*.

Anmerkung 3.6 Grundsätzlich können gemäß vorstehender Definition Teile der Liniensegmente einander überlappen. Die vorstehende Bedingung der Existenz nichtparalleler Segmente in einem geschlossenen Polygonzug stellt jedoch sicher, dass die enthaltenen Segmente nicht ausnahmslos übereinander liegen und das resultierende Polygon tatsächlich eine zweidimensionale Figur beschreibt. Abbildung 3.8 zeigt ein Beispiel für ein solches Polygon.

Anmerkung 3.7 Der Begriff *Polygon* wird im Folgenden äquivalent zu dessen beschreibendem geschlossenen Polygonzug ρ verwendet.



(a) Disjunkt **(b)** Überlappend **(c)** Verschachtelt

Abbildung 3.9: Lagebeziehung von zwei Polygonen $\rho_1, \rho_2 \in \Lambda$ nach Definition 3.8

Polygone werden im Folgenden unter anderem zur Kollisionsüberprüfung verwendet, bei welcher Distanzberechnungen zu einzelnen Punkten in der Ebenen durchgeführt werden müssen, vergleiche Abschnitt 3.2.4. Dies kann auf die Minimierung bezüglich aller enthaltenen Liniensegmente nach

$$\begin{aligned} &\mathrm{d}\colon \Lambda \times \mathbb{R}^2 \to \mathbb{R}, \\ &(\rho,\, q) \mapsto \min_{s \in \rho} \{\mathrm{d}(s,\, q)\} \cdot \mathrm{sgn}(\rho,\, q) \end{aligned} \tag{3.8}$$

mit

$$\begin{aligned} &\mathrm{sgn}\colon \Lambda \times \mathbb{R}^2 \to \{-1, 1\}, \\ &(\rho,\, q) \mapsto \begin{cases} -1, & q \textit{ strikt außerhalb von } \rho, \\ +1, & q \textit{ innerhalb von } \rho, \end{cases} \end{aligned}$$

zurückgeführt werden. Dies wird durch die Vorzeichenfunktion $\mathrm{sgn}(\rho,\, q)$ als negativ festgelegt, wenn sich der Testpunkt q außerhalb des Polygons ρ befindet. Dadurch lassen sich einerseits zusätzlich die Lageinformationen durch $\mathrm{d}(\rho,\, q)$ ausdrücken, andererseits liegt aufgrund der möglichen Negativität aber kein Distanzbegriff im klassischen Sinne mehr vor – da dies keine Auswirkungen auf die nachfolgenden Betrachtungen hat, wird $\mathrm{d}(\rho,\, q)$ der Einfachheit halber trotzdem im weiteren Verlauf als *Distanz* bezeichnet. Das Vorzeichen selbst lässt sich numerisch effizient auf Basis des Jordanschen Kurvensatzes bestimmen [28]. Dabei wird vom betrachteten Punkt q aus überprüft, wie oft ein halb-unendlicher Strahl in eine beliebige aber feste Richtung die Segmente des Polygons überschreitet. Eine ungerade Anzahl legt ihn dabei eindeutig als *innerhalb*, eine grade Anzahl als *außerhalb* fest, vergleiche Abbildung 3.8.

Basierend auf den Lageinformationen einzelner Punkte zu einem Polygon lässt sich auch die Lagebeziehung von zwei vollständigen Polygonen beschreiben.

Definition 3.8 (Lagebeziehungen von Polygonen)

Betrachte zwei Polygone, gegeben durch die geschlossenen Polygonzüge $\rho_1, \rho_2 \in \Lambda$.

1. Gilt
$$\exists s \in \rho_2 \colon \exists q_1, q_2 \in s \colon \operatorname{sgn}(\rho_1, q_1) \neq \operatorname{sgn}(\rho_1, q_2),$$
dann heißen ρ_1 und ρ_2 *überlappend*.
2. Gilt
$$\forall s \in \rho_2 \colon \forall q \in s \colon \operatorname{sgn}(\rho_1, q) = -1,$$
dann heißen ρ_1 und ρ_2 *disjunkt*.
3. Gilt
$$\forall s \in \rho_2 \colon \forall q \in s \colon \operatorname{sgn}(\rho_1, q) = 1,$$
dann heißt ρ_2 in ρ_1 *verschachtelt*.

Die drei betrachteten Lagebeziehungen sind in Abbildung 3.9 schematisch skizziert.

3.2.1.4 Freibereichspolygone

Basierend auf Definition 3.8 können spezielle Anforderungen an die Lagebeziehung für Mengen von Polygonen formuliert werden, was schließlich die Beschreibung eines Freibereichspolygons erlaubt.

Definition 3.9 (Freibereichspolygon)

Sei $\gamma \coloneqq \{\rho_1, \dots \rho_n\} \subset \Lambda$ für $n \in \mathbb{N}$ eine Menge von Polygonen für die gilt

$$\text{Falls } \rho_j \text{ verschachtelt in } \rho_i \implies \nexists \rho_k \in \gamma \setminus \{\rho_j\} \colon \rho_k \text{ verschachtelt in } \rho_j,$$

für $i, j, k \in [0, n]$, dann heißen γ *einfach verschachtelt*.
Sei dann die Menge der *strikt einfach verschachtelten Polygone* gegeben als

$$\begin{aligned}\Gamma \coloneqq \{\gamma = \{\rho_1, \dots \rho_n\} \mid\ & \gamma \text{ einfach verschachtelt},\\ & \rho_i, \rho_j \in \gamma \text{ paarweise verschachtelt oder disjunkt},\\ & n \in \mathbb{N}\}.\end{aligned}$$

Ein Element $\gamma \in \Gamma$ heißt im weiteren Verlauf *Freibereichspolygon*.

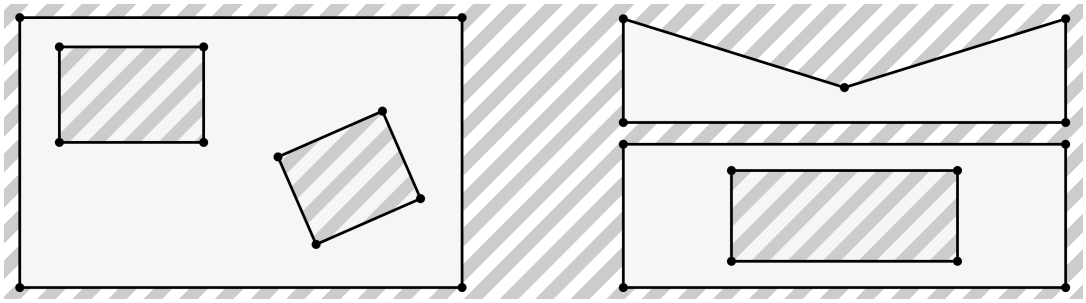


Abbildung 3.10: Beispiel eines Freibereichspolygons bestehend aus sechs Einzelpolygonen. Es liegt höchstes einfache Verschachtelung vor und nicht verschachtelte Polygone sind disjunkt. Die Schraffierung kennzeichnet, welche Bereiche nach $\operatorname{sgn}(\gamma, \cdot)$ aus (3.10) als *außerhalb* gelten.

Anmerkung 3.10 Jedes Polygon $\rho \in \Lambda$ kann als Freibereichspolygon aufgefasst werden.

Abbildung 3.10 zeigt ein Beispiel für eine strikt einfach verschachtelte Menge von Polygonen. Ein solches Freibereichspolygon verdankt seinen Namen vor allem der Interpretierbarkeit als Freibereich für Planungsalgorithmen. Dazu wird der verallgemeinerte Distanzbegriff für Polygone so erweitert, dass er für die spezielle Lagebeziehung der Elemente des Freibereichspolygons anwendbar ist:

$$\begin{aligned} &\mathrm{d}\colon \Gamma \times \mathbb{R}^2 \to \mathbb{R}, \\ &(\gamma, q) \mapsto \min_{\rho \in \gamma} \{|\mathrm{d}(\rho, q)|\} \cdot \operatorname{sgn}(\gamma, q) \end{aligned} \tag{3.9}$$

wobei sgn durch

$$\operatorname{sgn}\colon \Gamma \times \mathbb{R}^2 \to \{-1, 1\},$$

$$(\gamma, q) \mapsto \begin{cases} -1, & \operatorname{sgn}(\rho, q) = -1 \text{ für alle } \rho \in \gamma, \\ +1, & \operatorname{sgn}(\rho_i, q) = 1 \text{ für ein } \rho_i \in \gamma \text{ und} \\ & \operatorname{sgn}(\rho_k, q) = -1 \text{ für alle } \rho_k \in \gamma \setminus \{\rho_i\}, \\ -1, & \operatorname{sgn}(\rho_i, q) = \operatorname{sgn}(\rho_j, q) = 1 \text{ für zwei } \rho_i, \rho_j \in \gamma \text{ und} \\ & \operatorname{sgn}(\rho_k, q) = -1 \text{ für alle } \rho_k \in \gamma \setminus \{\rho_i, \rho_j\}, \end{cases} \tag{3.10}$$

gegeben ist. Abbildung 3.10 veranschaulicht die Definition der Lagebeschreibung mit $\operatorname{sgn}(\gamma, \cdot)$. Ein gegebener Testpunkt gilt genau dann als *innerhalb* gelegen, wenn er in nur einem einzigen Polygon enthalten ist. Dies ermöglicht Planungsalgorithmen zum einen, den Suchbereich nach außen hin abzugrenzen, aber auch innen liegende Hindernisse durch einzelne Polygon als *außerhalb* der planbaren Fläche zu beschreiben. Dabei kann es aufgrund von gegebenen Nebenbedingungen auch zu separierten Bereichen kommen. Der nächste Abschnitt stellt einen Algorithmus vor, welcher die Konstruktion von Polygonstrukturen basierend auf gegebenen Umgebungsinformationen ermöglicht, sodass genau die vorgenannten Eigenschaften erfüllt werden.

3.2.2 Automatische Generierung von Freibereichspolygonen

Es wird kurz ein Verfahren zur adaptiven Erstellung von Freibereichspolygonen vorgestellt, welches in seiner grundlegenden Form erstmals in [66] präsentiert wurde. Bei diesem wird die Umgebung des Fahrzeuges durch *lokale* Polygone approximiert. Diese werden um eine gegebene Menge von Expansionszentren $\Theta \subset \mathbb{R}^2$ herum erzeugt und anschließend zu einem Freibereichspolygon vereinigt. Ausgehend von dieser Idee wurde in [32] ein iteratives Verfahren eingeführt, durch welches die Expansionszentren adaptiv, basierend auf den Umgebungsinformationen, berechnet werden können. Dadurch werden Stück für Stück neue lokale Polygone identifiziert, bis ein vorgegebener Planungshorizont in Form einer Expansionsweite $\kappa \in \mathbb{R}_{>0}$ erreicht ist. Dabei werden sowohl räumliche Beschränkungen durch eine Punktwolke $D \subset \mathbb{R}^2$ als auch durch polygonale Strukturen $\rho^B \subset \Lambda$ berücksichtigt.
Da dieses Verfahren eine wichtige Grundlage für die weiteren Komponenten in dieser Arbeit darstellt, wird es im Folgenden kurz schematisch beschrieben. Die zugehörige Iteration ist in Algorithmus 3.1 dargestellt, wobei jeder Iterationsschritt $i \in \mathbb{N}$ grundsätzlich drei Bestandteile beinhaltet:

1. **Erzeugung lokaler Polygone:** Für jedes Element einer gegebenen Menge an *Expansionszentren* $\Theta = \{\vartheta_1, \ldots, \vartheta_{n_i}\} \subset \mathbb{R}^2$ wird jeweils ein einfaches lokales Polygon generiert, was in der Menge $\{\rho_1^{(i)}, \ldots, \rho_{n_i}^{(i)}\} \subset \Lambda$ zusammengefasst wird. Sie berücksichtigen die entsprechenden Daten zu räumlichen Beschränkungen aus der Punktwolke $D \subset \mathbb{R}^2$. Die Größe eines einzelnen Polygons wird dabei durch die lokale Expansionstiefe $\kappa_\rho \in \mathbb{R}_{>0}$ beschränkt, welche den maximalen Abstand der Polygonsegmente zum jeweiligen Expansionszentrum beschreibt.

2. **Clipping:** Die Menge aller expandierter Polygone wird in einem Freibereichspolygon γ vereinigt. Polygonale Beschränkungen ρ^B werden anschließend ausgeschnitten.

3. **Erzeugung von Expansionszentren:** Innerhalb des aktuellen Freibereichspolygons werden neue Expansionszentren Θ identifiziert, die nicht weiter als bis zur vorgegebenen Expansionsweite $\kappa \in \mathbb{R}_{>0}$ reichen. Sie werden so ausgewählt, dass sie den Abstand zu den Rändern des aktuellen Freibereichspolygons γ sowie zu allen vorigen Expansionszentren $\overline{\Theta}$ maximieren, dabei jedoch einen gegeben Minimalabstand von $\xi \in \mathbb{R}_{\geq 0}$ zueinander wahren.

Als Eingabe für das Verfahren wird (neben den räumlichen Beschränkungen) eine initiale Auswahl an Expansionszentren Θ benötigt. Diese können in der Praxis leicht aus dem aktuellen Systemzustand (des autonomen Fahrzeugs) oder bereits bekannten oder interessanten Bereichen in dessen Umgebung bestimmt werden. Der

Algorithmus 3.1 : Automatische Generierung eines Freibereichspolygons

Eingabe : Initiale Expansionszentren $\Theta \subset \mathbb{R}^2$, Beschränkungen aus Punktwolke $D \subset \mathbb{R}^2$ und Polygonen $\rho^B \subset \Lambda$, Expansionsweite $\kappa \in \mathbb{R}_{>0}$, lokale Expansionstiefe $\kappa_\rho \in \mathbb{R}_{>0}$, Expansionsabstand $\xi \in \mathbb{R}_{\geq 0}$

Ausgabe : Freibereichspolygon $\gamma \in \Gamma$

Initialisierung : Freibereichspolygon $\gamma \leftarrow \emptyset$, Bereits expandierte Zentren $\overline{\Theta} \leftarrow \emptyset$, Iterationszähler $i \leftarrow 0$

---

while $\Theta \neq \emptyset$ **do** // Leer, wenn Expansionsweite überschritten
  $i \leftarrow i + 1$;
  // Erzeuge lokale Polygone
  $n_i \leftarrow |\Theta|$;
  $\{\rho_1^{(i)}, \ldots, \rho_{n_i}^{(i)}\} \leftarrow$ `GenerateLocalPolygons`(Θ, D, κ_ρ); // Siehe [32]
  // Clipping, vergleiche Abbildung 3.12
  $\gamma \leftarrow$ `Unite` $\left(\rho_1^{(1)}, \ldots, \rho_{n_1}^{(1)}, \ldots \rho_1^{(i)}, \ldots, \rho_{n_i}^{(i)}\right)$; // Vereinigung
  $\gamma \leftarrow$ `Intersect` $\left(\gamma, \rho^B\right)$; // Schnitt
  // Erzeuge neue Expansionszentren
  $\overline{\Theta} \leftarrow \overline{\Theta} \bigcup \Theta$;
  $\Theta \leftarrow$ `GenerateExpansionCenters`$(\gamma, \overline{\Theta}, \kappa, \xi)$; // Siehe [32]
end
// Zusammenfassung von Polygonsegmenten
$\gamma \leftarrow$ `Simplify`(γ) ;
return γ;

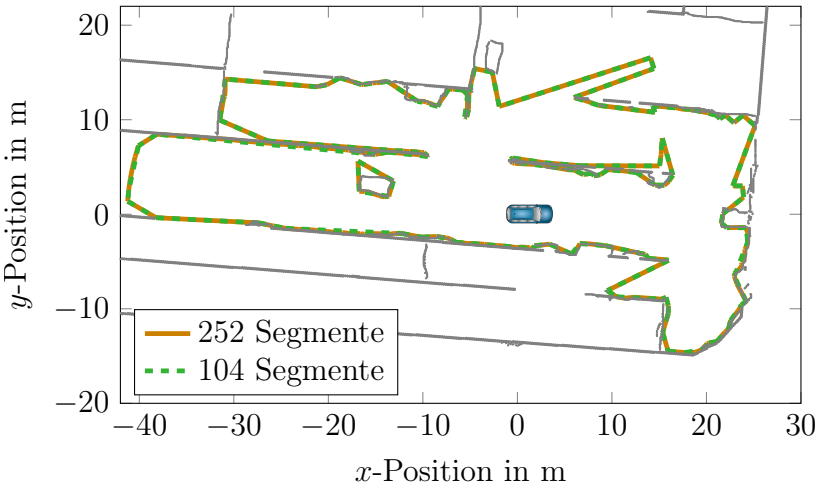
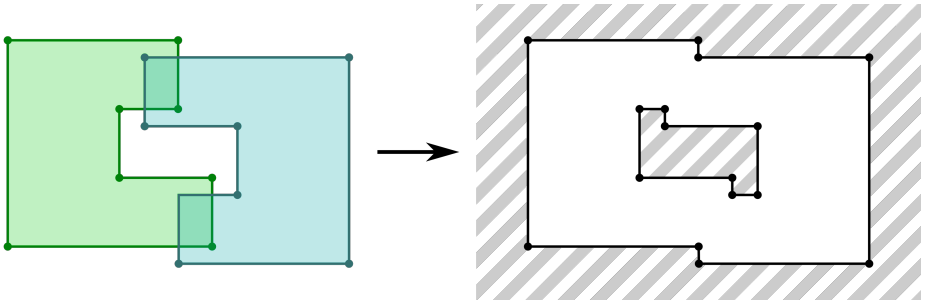


Abbildung 3.11: Unterschiedliche Auflösungen des Freibereichspolygons für den langen Planungshorizont aus Abbildung 3.5 durch die Anwendung der Polygonvereinfachung `Simplify`

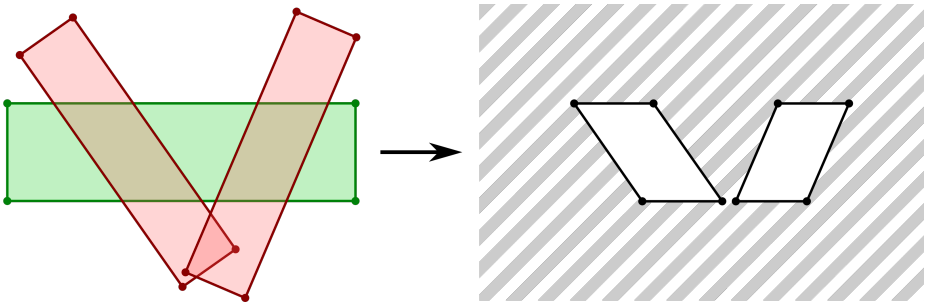
Algorithmus endet, wenn keine weiteren Expansionszentren innerhalb der vorgegebenen Expansionsweite gefunden werden können, vergleiche Zeile 1. Das resultierende Freibereichspolygon wird abschließend durch das Zusammenfassen von aufeinanderfolgenden und annähernd parallel verlaufenden Polygonsegmenten vereinfacht, siehe Zeile 10. Wann zwei Elemente dabei reduziert werden können, wird durch einen Schwellenwert festgelegt. Wird dieser moderat gewählt, kann eine deutliche Reduzierung der Polygongrößen erreicht werden, was den Umgang mit ihnen numerisch effizienter macht, ohne sie dabei geometrisch wesentlich zu verändern. Dies ist gut am Beispiel aus Abbildung 3.11 zu erkennen, in welcher die gezeigten Beispiele nur leichte Unterschiede aufweisen, obwohl die Menge der Polygonsegmente auf fast 40 % reduziert wird.

Um aus den lokalen Polygonen sowie den polygonalen Beschränkungen ein gemeinsames Freibereichspolygon γ zu erzeugen, werden diese durch Vereinigungen oder Schnitte gezielt modifiziert, siehe Zeilen 5 und 6 von Algorithmus 3.1. Diese Arten der Manipulation werden (zusammen mit Weiteren) als *Clipping*-Operationen bezeichnet. Sie können zum Beispiel durch den Algorithmus von Vatti [101] umgesetzt werden, welcher speziell für zweidimensionale und beliebig komplizierte Polygone entworfen wurde. Abbildung 3.12 zeigt für die hier genutzten Operationen jeweils ein schematisches Beispiel. Die Vereinigung zweier oder mehrerer Polygone entfernt dabei zunächst vorhandene Überlappungen. Dabei können jedoch Verschachtelungen (grundsätzlich beliebiger Art) im Ergebnis auftreten. Beim Herausschneiden mehrerer polygonaler Beschränkungen ρ^B können dagegen disjunkte Bereiche entstehen.

Anmerkung 3.11 Das Ergebnis der Clipping-Operationen in den Zeilen 5 und 6 von Algorithmus 3.1 genügt stets den Kriterien eines Freibereichspolygons gemäß Definition 3.9. Dabei ist vor allem wichtig, dass die Vereinigung nur zu einer einfa-



(a) Vereinigung des grünen Polygons mit dem in blau gezeichneten

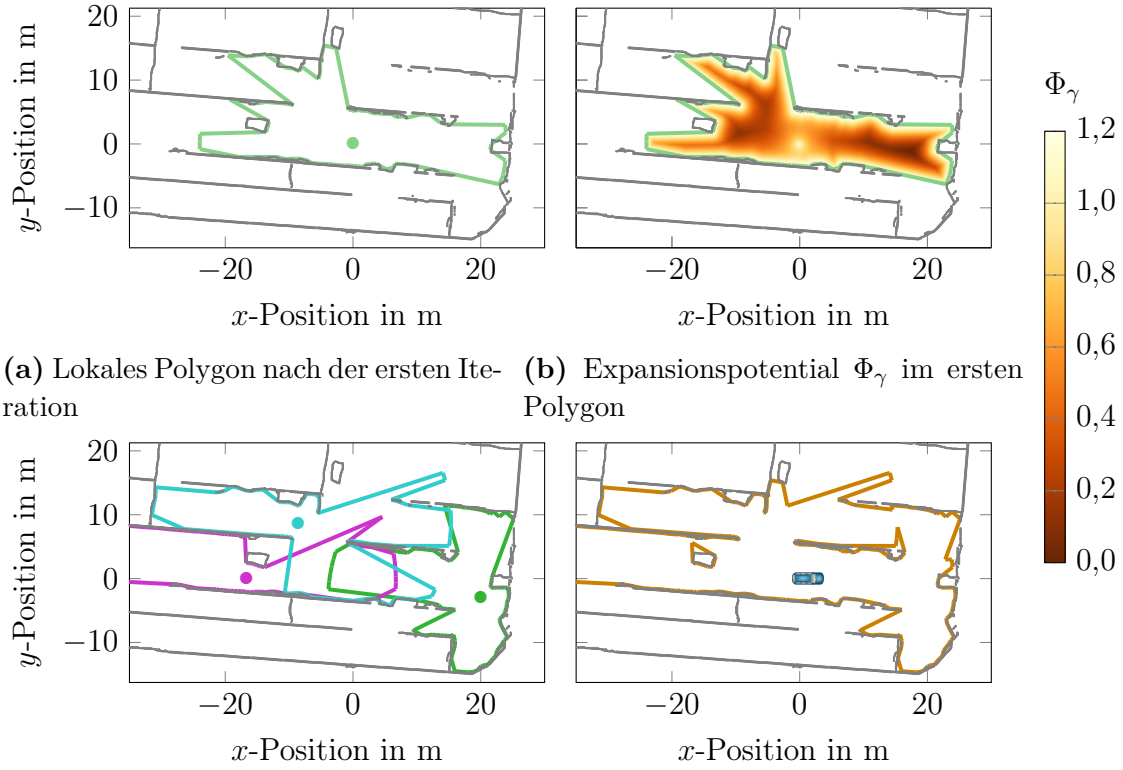


(b) Schnitt des grünen Polygons mit den in rot gezeichneten

Abbildung 3.12: Schematische Darstellung der Clipping-Operationen *Vereinigung* und *Schnitt* von einzelnen Polygonen (links) zur Generierung eines Freibereichspolygons (rechts)

chen Verschachtelung führen kann. Dies ist dadurch garantiert, dass neue Expansionszentren stets im Inneren des aktuellen Freibereichspolygons liegen und die resultierenden lokalen Polygone damit stets nur überlappend, aber niemals verschachtelt oder disjunkt sein können.

Ein exemplarischer Verlauf des Iterationsverfahrens von Algorithmus 3.1 ist in Abbildung 3.13 dargestellt. Zur Vereinfachung der Darstellung wird dabei lediglich die Punktwolke aus Abbildung 3.5 ohne polygonale Beschränkungen betrachtet, sodass keine Schnitt-Operationen notwendig sind. Die Berücksichtigung und Anwendung solcher zusätzlicher Informationen wird in Abschnitt 4.2.1 präsentiert. Abbildung 3.13a zeigt sehr gut, wie das erste lokale Polygon sich an die Beschränkungen im Umfeld von dessen Expansionszentrum anpasst. Wird der Algorithmus (wie hier) mit nur einem einzigen Startpunkt initialisiert, so ist in der ersten Iteration keine Vereinigung notwendig und das aktuelle Freibereichspolygon γ entspricht dem lokalen Polygon. Zur Identifikation der Expansionszentren für den nächsten Schritt wird ein Potentialfeld Φ_γ innerhalb des Freibereichspolygons betrachtet. Für einen



(a) Lokales Polygon nach der ersten Iteration

(b) Expansionspotential Φ_γ im ersten Polygon

(c) Lokale Polygone nach der zweiten Iteration

(d) Vereinigtes Freibereichspolygon γ nach der zweiten Iteration

Abbildung 3.13: Die ersten zwei exemplarischen Iterationen der automatischen Generierung eines Freibereichspolygons γ für das Szenario aus Abbildung 3.5. Die lokalen Polygone sind zusammen mit den jeweiligen Expansionszentren in hell- und dunkelgrün, lila sowie türkis gezeigt. Das erste Expansionszentrum ergibt sich aus dem Fahrzeugzustand. Das resultierende Freibereichspolygon γ wird in orange gezeichnet. Die Werte des Expansionspotentials für das erste lokale Polygon sind als Farbverlauf dargestellt.

Punkt q in der Ebene ist dessen Wert durch

$$\Phi_\gamma(q) := \begin{cases} \frac{1}{1+\mathrm{d}(\gamma,\, q)} + \sum\limits_{\bar{\vartheta}\in\bar{\Theta}} \frac{1}{1+\mathrm{d}\left(\bar{\vartheta},\, q\right)}, & \text{für } \operatorname{sgn}(\gamma,\, q) = 1, \\ \infty, & \text{sonst,} \end{cases}$$

gegeben. Da die verallgemeinerte Distanzfunktion $\mathrm{d}(\gamma,\, \cdot)$ zum Freibereichspolygon nur für Punkte in dessen Inneren ausgewertet wird und damit stets positiv ist, gilt $\Phi_\gamma(q) \in (0,\, \infty]$ für alle $q \in \mathbb{R}^2$. Für das erste Freibereichspolygon ist das Potentialfeld in Abbildung 3.13b gezeigt. Es ist gut erkennbar, dass vor allem im Zentrum von Bereichen, die Abseits des ersten Expansionspunktes liegen, niedrige Werte angenommen werden. Die neuen Expansionszentren werden dann als Menge von Punkten mit möglichst kleinen Potentialwerten bestimmt, die jedoch den

gegebenen Mindestabstand von ξ untereinander sowie zu sämtlichen vorigen Punkten $\overline{\Theta}$ einhalten. Im gezeigten Beispiel führt dies zu drei Lösungspunkten, welche in Abbildung 3.13c gezeigt werden. Dabei ist zu beachten, dass diese aufgrund der Abstandsbedingung nicht komplett zentral in den Tälern des Potentialfeldes liegen. Ausgehend von den aktualisierten Expansionszentren werden neue lokale Polygone erzeugt; die Vereinigung aller vier bisherigen Polygone entspricht dann der zweiten Iteration des Freibereichspolygons, wie in Abbildung 3.13d dargestellt.

Für eine genauere Darstellung der Berechnung der lokalen Polygone sowie der Expansionszentren wird auf [32] verwiesen.

3.2.3 Effizienz und Genauigkeit der Polygongenerierung

Freibereichspolygone ermöglichen insbesondere bei Daten aus Punktwolken eine drastische Verringerung im Aufwand zur Kollisionsüberprüfung. Im Beispiel aus Abbildung 3.5 reduziert sich diese von der Berechnung von Distanzen zu 7376 Punkten auf nur noch 112 beziehungsweise 252 Polygonsegmente für den kurzen respektive langen Planungshorizont. Der sich daraus ergebende Geschwindigkeitsvorteil steht dabei dem Aufwand zur Erstellung eines Freibereichspolygons gemäß Algorithmus 3.1 gegenüber. Hier ist zum einen ebenfalls die Menge der gegeben Daten relevant, zum anderen spielt die Approximationsgenauigkeit der lokalen Polygone eine wichtige Rolle. Diese ergibt sich wiederum im Wesentlichen aus der Anzahl der verwendeten Polygonsegmente zur Repräsentation des jeweiligen Umfeldes. Durch deren Kalibrierung lässt sich sehr einfach eine Abwägung zwischen Genauigkeit des Gesamtergebnisses und Rechenzeit umsetzen.

Abbildung 3.14 zeigt den benötigten Rechenaufwand für die ersten zwei Iterationen von Algorithmus 3.1 basierend auf der Punktwolke aus Abbildung 3.5 abhängig von der Anzahl der Segmente der lokalen Polygone. Indem gleichmäßig Einträge aus der Punktwolke entfernt werden, kann zudem der Verlauf für verschieden große Datenmengen untersucht werden. Für die Umsetzung des Polygonclippings und der Polygonvereinfachung wird das Softwarepaket *Clipper* [51] verwendet. Für bessere Übersichtlichkeit in der Darstellung wird in dieser Auswertung (wie zuvor) auf polygonale Beschränkungen verzichtet, sodass keine Schnitt-Operationen betrachtet werden.
Die Ergebnisse zeigen, dass sich die Laufzeit des Verfahrens bezüglich beider Komponenten annähernd linear verhält, was für eine gute Robustheit gegenüber Änderungen in den Eingabedaten spricht. Für eine sehr feine Auflösung der lokalen Polygone ergeben sich dabei vergleichsweise hohe Rechenzeiten – in dieser Auswertung bis zu 58,6 ms. Ist eine geringere Genauigkeit ausreichend, werden umgekehrt auch beliebig niedrige Werte von bis zu weit unter einer Millisekunde erzielt. Die notwendige Genauigkeit für ein hinreichend gutes Approximationsergebnis hängt dabei von der Komplexität des betrachteten Problems ab. Für das relativ unstrukturierte

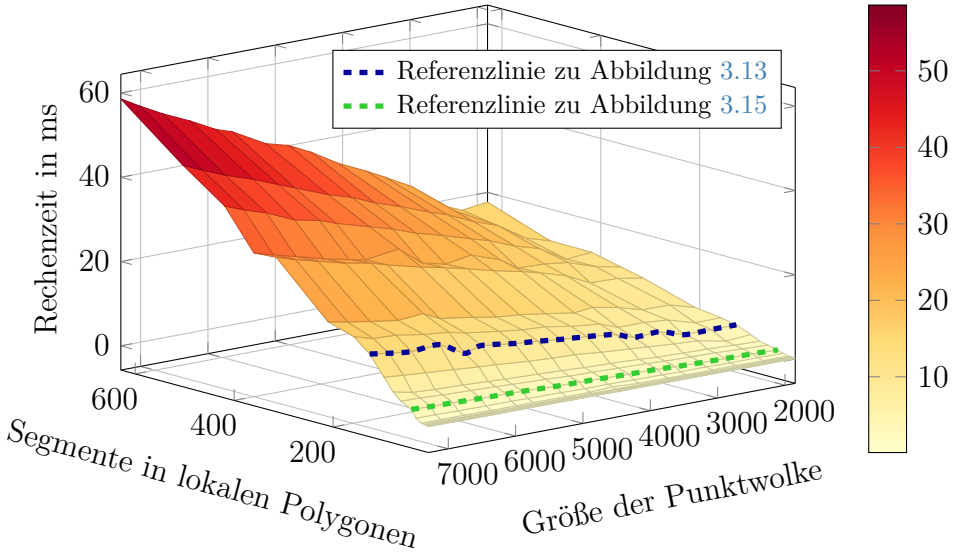
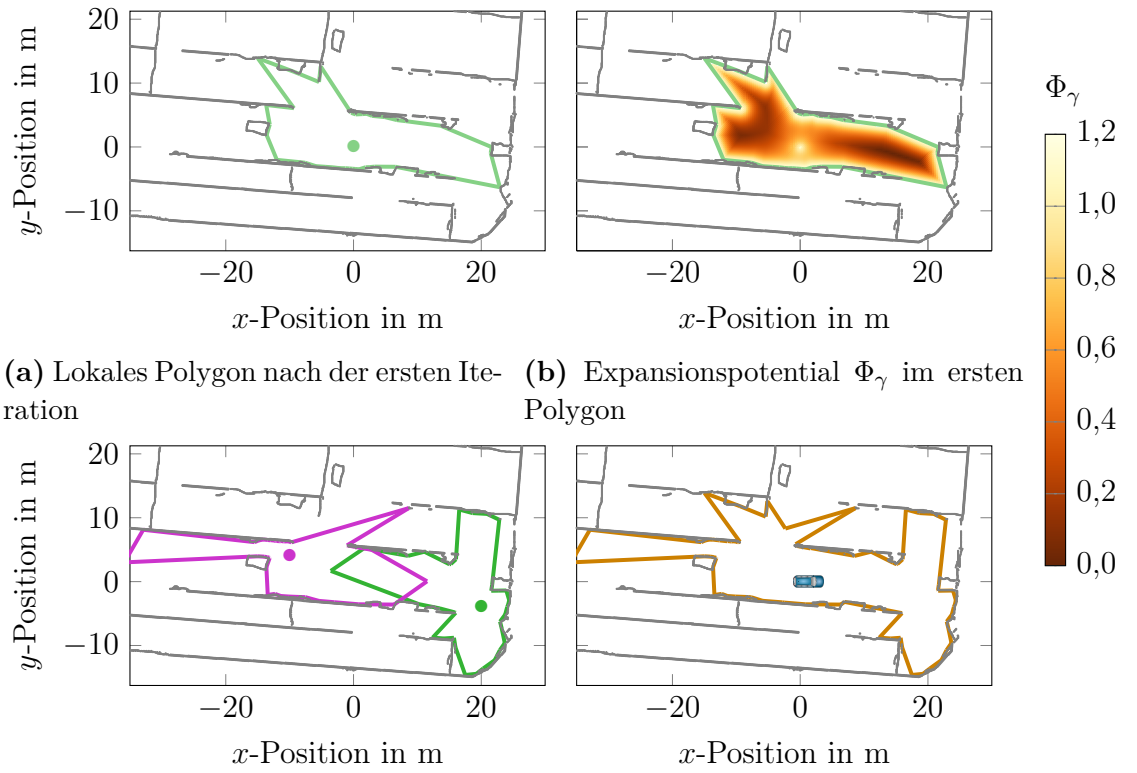


Abbildung 3.14: Exemplarische Rechenzeiten für die ersten zwei Iterationen von Algorithmus 3.1 zur Berechnung eines Freibereichspolygons für das Szenario aus Abbildung 3.5 mit unterschiedlichen Dichten der Punktwolke und für verschiedene Approximationsgenauigkeiten der lokalen Polygone. Zum Vergleich sind die Referenzlinien für die Polygonauflösungen der Abbildungen 3.13 und 3.15 hervorgehoben.

(und damit aus Sicht eines autonomen Fahrzeugs anspruchsvolle) Beispiel aus Abbildung 3.13 wurden lokale Polygone mit jeweils 126 Segmenten verwendet, wodurch ein vergleichsweise großer Bereich mit hoher Güte abgebildet werden konnte. Je nach Dichte der Punktwolke sind dazu 4,4 ms bis 13,9 ms Rechenzeit notwendig.

Im Vergleich dazu zeigt Abbildung 3.15 das Ergebnis für eine deutlich geringere Auflösung der lokalen Polygone mit nur jeweils 42 Segmenten. Es ist gut zu erkennen, dass insbesondere freie Flächen im Nahbereich der Fahrzeugs weiterhin sehr genau abgebildet werden. Kreuzungen oder Objekte auf der Straße werden hingegen vergleichsweise schlecht dargestellt. Es fällt insbesondere auf, dass in der zweiten Iteration des Verfahrens nur von zwei Expansionszentren aus gesucht wird, während im vorigen Beispiel an dieser Stelle noch drei lokale Polygone erzeugt wurden. Dies ist darauf zurückzuführen, dass durch die gröbere Expansion im ersten Schritt weniger freie Flächen entstehen, in welche der Algorithmus dann neue Expansionszentren platzieren kann. Insgesamt führt dies schließlich auch im Endergebnis zu einem räumlich deutlich kleineren Freibereichspolygon, bestehend aus nur 75 Segmenten. Entsprechend geringe Laufzeiten sind jedoch dann für dessen Berechnung nötig; in diesem Beispiel zwischen 1,1 ms und 3,4 ms, abhängig von der Dichte der Punktwolke.

Neben den beiden in Abbildung 3.14 untersuchten Aspekten gibt es noch weitere Faktoren mit Einfluss auf die Verfahrenszeit von Algorithmus 3.1. Dies betrifft zum



(a) Lokales Polygon nach der ersten Iteration

(b) Expansionspotential Φ_γ im ersten Polygon

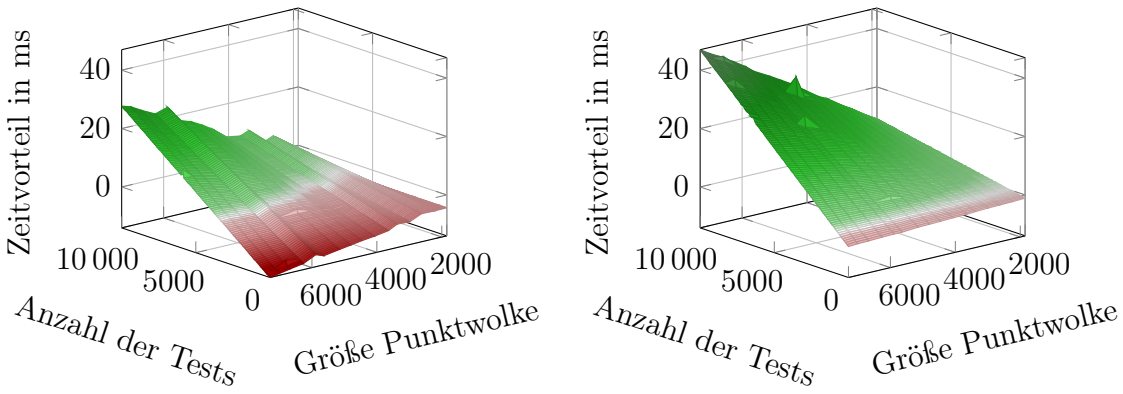
(c) Lokale Polygone nach der zweiten Iteration

(d) Vereinigtes Freibereichspolygon γ nach der zweiten Iteration

Abbildung 3.15: Die ersten zwei exemplarischen Iterationen der automatischen Generierung eines Freibereichspolygons γ. Die Darstellung ist analog zu Abbildung 3.13, allerdings wird hier eine verringerte Auflösung der lokalen Polygone von jeweils 42 Segmenten verwendet.

einen die Anzahl der Iterationen insgesamt: Wird eine größere Expansionsweite κ vorgegeben und damit mehr als die hier gezeigten zwei Schritte durchgeführt, wird ein entsprechend größerer Bereich um das Fahrzeug herum approximiert. Dabei müssen öfter neue Expansionspunkte gesucht und mehr lokale Polygone erzeugt werden, was eine höhere Laufzeit benötigt. Dem gegenüber steht die Menge der Expansionspunkte pro Iteration: Je kleiner der Mindestabstand ξ vorgegeben wird, desto dichter werden diese gesetzt und entsprechend mehr lokale Polygone pro Schritt erzeugt. Dadurch können zum Beispiel größere Unregelmäßigkeiten in der Punktwolke abgebildet werden – auf Kosten einer erhöhten Laufzeit. Insgesamt ermöglicht das hier gezeigte Verfahren damit eine genaue Abwägung zwischen gewünschtem Planungshorizont, lokaler Genauigkeit und benötigter Rechenzeit.

Der Aufwand für die Erstellung des Freibereichspolygons wird jedoch vergleichsweise schnell durch die deutlich erhöhte Effizienz in der Auswertung von Kollisionsüberprüfungen ausgeglichen. Dies wird exemplarisch in Abbildung 3.16 anhand der



(a) Lokale Polygone mit 126 Segmenten, vergleiche Abbildung 3.13

(b) Lokale Polygone mit 42 Segmenten, vergleiche Abbildung 3.15

Abbildung 3.16: Geschwindigkeitsdifferenz bei der Nutzung eines Freibereichspolygons gegenüber Punktwolken zur Berechnung von Distanzen zu zufälligen Punkten. Im Farbverlauf indiziert grün einen Geschwindigkeitsvorteil, rot einen Nachteil. Es wird die Punktwolke aus Abbildung 3.5 mit verschiedenen Dichten betrachtet. Für jede Dichte wird die benötigte Zeit zur Generierung des entsprechenden Freibereichspolygons berücksichtigt. Es werden die Ergebnisse für beide Approximationsgenauigkeiten der Polygone aus den Abbildungen 3.13 und 3.15 gezeigt.

Berechnung von minimalen Distanzen zu zufällig gewählten Punkten dargestellt. Dabei wird der Zeitvorteil bei der Verwendung von zwei Freibereichspolygonen mit unterschiedlichen Approximationsgenauigkeiten gegenüber reinen Punktwolken mit verschiedenen Dichten untersucht. Der Verlauf bezüglich der Anzahl der Kollisionstests ist dabei in allen Fällen linear; es sind lediglich sehr vereinzelte Peaks zu sehen, die als numerische Artefakte einzustufen sind. Einzig in Bezug auf die Stärke der Punktwolke zeigen die Ergebnisse für das große Polygon in Abbildung 3.16a einen leicht nicht-linearen Verlauf, der jedoch in sehr starkem Bezug zu den geringen Laufzeitschwankungen in der Polygonerstellung steht, vergleiche die zugehörige Referenzlinie in Abbildung 3.14. Entsprechend der schwankungsfreien Referenzlinie für das Polygon mit weniger Segmenten, ist der Verlauf in Abbildung 3.16b für beide Dimensionen linear.

In allen Fällen ist deutlich zu erkennen, dass der Effizienzvorteil des Polygonansatzes proportional mit der Anzahl der Distanzberechnungen zunimmt. Ab wann genau die Verwendung von Polygonen dann aus Sicht der Rechenzeit sinnvoll ist, hängt von deren Genauigkeit ab. Für 42 lokale Segmente lohnt sich die Erstellung des Freibereichspolygons für alle Punktwolken bereits ab ca. 800 Auswertungen. Die Laufzeit für 126 lokale Segmente ist dagegen deutlich höher, sodass hier ungefähr 3300 Distanzberechnungen benötigt werden. Für wenige oder keine Auswertungen verbleibt in beiden Fällen nur der jeweilige Aufwand zur Erstellung des Freibereichspolygons.

Die Berechnungen von Distanzen von gegebenen Punkten zu den Umgebungsdaten stellt im Folgenden eine wichtige Komponente dar: Sowohl das Ego-Fahrzeug als auch bewegliche Objekte im Allgemeinen werden im weiteren Verlauf als Kreisüberdeckung approximiert, wie der nächste Abschnitt 3.2.4 präsentiert. Kollisionsüberprüfungen reduzieren sich dann letztlich auf eine Reihe von Distanzberechnungen zu den Zentren dieser Kreise. Daraus resultierend sind schon bei kleinen Optimierungsaufgaben viele tausend Auswertungen notwendig und die Verwendung eines Freibereichspolygons entsprechend von Vorteil. Größenordnungen für konkrete Probleme lassen sich anhand der Anzahl der zu öffnenden hybriden Knoten in den Ergebnissen in Abschnitt 4.1 ableiten.
Zusätzlich zum hier betrachteten Aspekt der Rechenzeit gelten auch die weiteren im Eingang von Abschnitt 3.2 genannten positiven Eigenschaften, insbesondere als Grundlage zur Erstellung von Voronoi-Potentialfeldern, was in Abschnitt 3.3 detailliert betrachtet wird.

3.2.4 Kollisionsüberprüfung im statischen und dynamischen Umfeld

Die Verwendung eines Freibereichspolygons ermöglicht die vereinfachte Darstellung des statischen Umfelds von (beispielsweise) einem autonomen Fahrzeug. Zur Umsetzung einer vollständigen Kollisionsüberprüfung wird jedoch zusätzlich eine Darstellung von beweglichen Objekten, wie anderen Fahrzeugen oder Fußgängern, sowie ein Verfahren zur Berechnung von Distanzen zwischen dem Ego-Fahrzeug sowie allen Hindernisrepräsentationen benötigt. Unterschreitet eine dieser Distanzen in einem gegebenen Fahrzeugzustand einen festgelegten Schwellenwert für den Abstand, dann wird dies im weiteren Verlauf als *Kollision* bezeichnet. Kollisionsüberprüfungen auf Grundlage von detaillierten Modellierungen der räumlichen Ausdehnung von Objekten sind in der Regel sehr aufwändig und damit nicht für die häufige Auswertung in einem effizienten Optimierungsverfahren geeignet. Eine bekannte Methode zur Reduktion dieser Komplexität besteht in der Verwendung einer Überdeckung durch einfach auswertbare geometrische Strukturen [29].

Im Kontext des autonomen Fahrens stellt eine orientierte Bounding Box eine kanonische Überdeckung des Ego-Fahrzeugs sowie von bekannten beweglichen Objekten dar.

Definition 3.12 (Orientierte Bounding Box)

Sei ein Objekt der Länge $\Delta_l \in \mathbb{R}_{>0}$ und Breite $\Delta_b \in \mathbb{R}_{>0}$ gegeben. Betrachte das Tupel der Punkte $\psi^{\square} = (\psi^{[1]}, \ldots, \psi^{[4]})$ mit $\psi^{[i]} \in \mathbb{R}^2$ und definiere für diese die vier Segmente $s_{1,2}$, $s_{2,3}$, $s_{3,4}$, $s_{4,1} \in \mathcal{S}$ mit $s_{i,j} \coloneqq (\psi^{[i]}, \psi^{[j]})_{\mathcal{S}}$. Gilt

1. $\|s_{1,2}\|_2 = \|s_{3,4}\|_2 = \Delta_l$ und $\|s_{2,3}\|_2 = \|s_{4,1}\|_2 = \Delta_b$ sowie
2. die Punkte in $\psi^{\square}$ stellen ein geometrisches Rechteck dar, also $s_{1,2} \perp s_{2,3}$, $s_{1,2} \parallel s_{3,4}$ und $s_{2,3} \parallel s_{4,1}$,

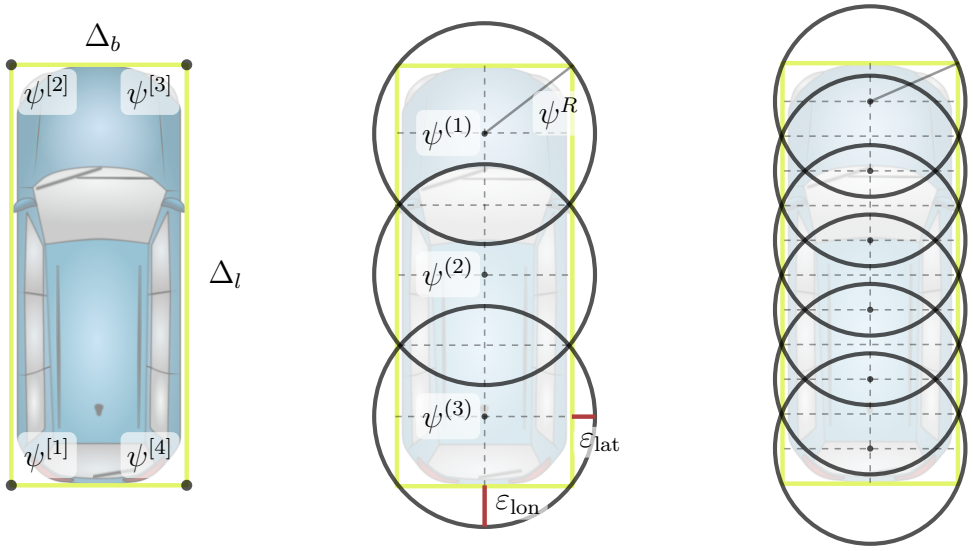
dann wird $\psi^{\square}$ als *orientierte Bounding Box* des Objektes bezeichnet. Entsprechend beschreibt

$$\Omega^{\square} \coloneqq \left\{ \psi^{\square} = (\psi^{[1]}, \ldots, \psi^{[4]}) \,\middle|\, \psi^{\square} \text{ erfüllt Punkte 1 und 2 für } \Delta_l, \Delta_b \in \mathbb{R}_{>0} \right\}$$

allgemein die Menge der orientierten Bounding Boxen.

Abbildung 3.17a zeigt ein Beispiel für eine Bounding Box. Sie approximiert die jeweilige Instanz durch ein entsprechend der Ausrichtung orientiertes Rechteck und kann damit aus dem Wissen über Position, Orientierung sowie Länge Δ_l und Breite Δ_b generiert werden. Für andere Verkehrsteilnehmer in einer realen Anwendung lassen sich diese Informationen entweder durch Schätzungen basierend auf der Sensorik des Ego-Fahrzeuges bestimmen oder ergeben sich aus externen Wissensquellen wie zum Beispiel V2X Kommunikation. Die Auswertung von Kollisionsüberprüfungen reduziert sich dann auf Überlappungstests der Bounding Box des Ego-Fahrzeugs mit denen der relevanten Objekte in dessen Umgebung, zum Beispiel auf Basis des Trennungssatzes für konvexe Mengen [29]. Zur Berechnung eines Voronoi-Feldes, zum Beispiel nach (2.5), wird jedoch der konkrete Abstand zwischen diesen Objekten zusätzlich zu einer binären Kollisionsaussage benötigt. Für zwei orientierte Bounding Boxen erfordert dies eine vergleichsweise aufwändige Reihe von Auswertungen zwischen den Kanten und Ecken beider Geometrien. Das Konzept kann generell auch einfach auf den dreidimensionalen Fall erweitert werden, wobei eine Bounding Box dann in Form eines Quaders vorliegt und entsprechend auch Höheninformationen berücksichtigt. In diesem Kontext wird für Überlappungstests gegebenenfalls sogar ein iteratives Verfahren wie der Gilbert-Johnson-Keerthi-Algorithmus benötigt [40].

Ein wesentlich effizienterer Ansatz ergibt sich durch eine Überdeckung basierend auf einer vorgegebenen Anzahl n° von Kreisen (beziehungsweise Sphären im Dreidimensionalen) mit konstantem Radius.



(a) (Orientierte) Bounding Box **(b)** Dreifache Kreisüberdeckung **(c)** Sechsfache Kreisüberdeckung

Abbildung 3.17: Zweidimensionale Approximation der räumlichen Ausdehnung eines Fahrzeuges mit einer Bounding Box und unterschiedlichen Kreisüberdeckungen

Definition 3.13 (Kreisüberdeckung)

Die Menge der n°-fachen *Kreisüberdeckungen* ist durch

$$\Omega^{(n^\circ)} := \left\{ (\psi^{(1)}, \ldots, \psi^{(n^\circ)}, \psi^R) \mid \psi^{(1)}, \ldots, \psi^{(n^\circ)} \in \mathbb{R}^2, \psi^R \in \mathbb{R}_{>0} \right\}$$

gegeben, wobei $\psi^{(1)}, \ldots, \psi^{(n^\circ)}$ die Koordinaten der Kreismittelpunkte und ψ^R den Kreisradius beschreiben. Die Menge aller möglichen Kreisüberdeckungen ergibt sich als

$$\Omega^\circ := \{ \psi^\circ \in \Omega^{(n^\circ)} \mid \text{für } n^\circ \in \mathbb{N}_{>0} \}.$$

Für eine gegebene Bounding Box lässt sich eine n°-fache Kreisüberdeckung automatisiert berechnen. Wie in den Abbildungen 3.17b und 3.17c skizziert, wird dabei die Fahrzeuglänge Δ_l durch $2 \cdot n^\circ - 1$ Schnitte unterteilt und die Mittelpunkte zentriert auf alle ungeraden Schnittlinien platziert. Der minimal notwendige Überdeckungsradius ψ^R ergibt sich mit der Fahrzeugbreite Δ_b aus

$$\psi^R = \sqrt{\left(\frac{\Delta_b}{2}\right)^2 + \left(\frac{\Delta_l}{2 \cdot n^\circ}\right)^2}.$$

Ein zusätzlicher Sicherheitsabstand kann dabei direkt durch eine Erhöhung des Radius berücksichtigt werden. Für Objekte, die breiter als lang sind, lässt sich die

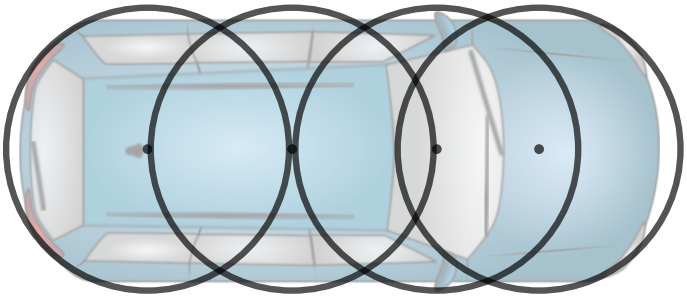


Abbildung 3.18: Speziell an Ego-Fahrzeug angepasste Kreisüberdeckung $\psi^{\circ}_{\text{Ego}} \in \Omega^{\circ}$. Wie in Abbildung 3.17 haben die Kreise einen konstanten Radius, können allerdings gegebenenfalls unregelmäßig angeordnet sein. Genaues Wissen über die Fahrzeugform (wie Abrundungen oder die Lage und Größe der Außenspiegel) werden besonders berücksichtigt. Die numerischen Parameter für das Forschungsfahrzeug aus Abschnitt 1.2 sind in Tabelle A.1 dargestellt.

gezeigte Konstruktionsvorschrift entsprechend durch das Vertauschen von Δ_b und Δ_l übertragen. Eine Verallgemeinerung ins Dreidimensionale ist ebenfalls möglich; der Rahmen dieser Arbeit beschränkt sich jedoch auf Planungsprobleme in der Ebene, sodass dies hier nicht weiter betrachtet werden soll.

Die Kreisüberdeckung kann damit als Approximation der orientierten Bounding Box angesehen werden und benötigt zur Generierung keine zusätzlichen Informationen. Je geringer die Anzahl der verwendeten Kreise ist, desto stärker ist dabei die Approximationsungenauigkeit ε_{lat} in der lateralen Fahrzeugrichtung. Mit einer größeren Kreismenge wird diese geringer – auf Kosten eines höheren Fehlers ε_{lon} in der longitudinalen Richtung. Wird ein minimales Verhältnis $\varepsilon_{\text{lon}}/\varepsilon_{\text{lat}}$ von beiden als Schwellenwert $\delta_C \in \mathbb{R}_{>0}$ festgelegt, so kann die (minimal) notwendige Anzahl an Kreisen adaptiv durch

$$n^{\circ}(\delta_C) \coloneqq \min\left\{ \bar{n}^{\circ} \in \mathbb{N}_{>0} \;\middle|\; \underbrace{\frac{\psi^R - \frac{\Delta_b}{2}}{\psi^R - \frac{\Delta_l}{2 \cdot \bar{n}^{\circ}}}}_{=\frac{\varepsilon_{\text{lon}}}{\varepsilon_{\text{lat}}}} > \delta_C \right\}$$

festgelegt werden. Generell führt die hier gezeigte Kreisüberdeckung stets zu einer vergleichsweise großen Überapproximation, wie in Abbildung 3.17 gut zu erkennen ist. Für das Ego-Fahrzeug liegen in der Regel aber zusätzliche Informationen über dessen genaue Form vor. Dies ermöglicht die Konstruktion einer verbesserten Überdeckung $\psi^{\circ}_{\text{Ego}} \coloneqq (\psi^{(1)}_{\text{Ego}}, \ldots, \psi^{(n^{\circ}_{\text{Ego}})}_{\text{Ego}}, \psi^{R}_{\text{Ego}}) \in \Omega^{\circ}$ und somit eine deutliche Reduktion des entstehenden Approximationsfehlers. Dabei werden zum Beispiel bekannte Abrundungen in der Karosserie oder die Außenspiegel besser berücksichtigt, wie Abbildung 3.18 exemplarisch für das in Abschnitt 1.2 vorgestellte Forschungsfahrzeug zeigt.
Relativ zur Kreisüberdeckung des Ego-Fahrzeuges kann ein fester Referenzpunkt

$p_{\text{Ego}} = p_{\text{Ego}}(\psi^\circ_{\text{Ego}}) \in \mathbb{R}^2$ definiert werden. In dieser Arbeit wird für diesen das Zentrum der Hinterachse gemäß dem Fahrzeugzustand z_{Ego} aus Abschnitt 3.1.1 verwendet, welcher sich durch eine konstante Verschiebung aus den Mittelpunkten der entsprechenden Kreisüberdeckung errechnen lässt. Liegen umgekehrt die Koordinaten des Referenzpunktes sowie eine Orientierung φ_{Ego} vor, so lassen sich aus diesen die Mittelpunkte der Kreisüberdeckung

$$\psi^\circ_{\text{Ego}} = \psi^\circ_{\text{Ego}}(p_{\text{Ego}},\, \varphi_{\text{Ego}})$$

bestimmen.

Die Berechnung des Abstandes zwischen den Kreisüberdeckungen $\psi^\circ_1 \in \Omega^{(n^\circ_1)}$ und $\psi^\circ_2 \in \Omega^{(n^\circ_2)}$ zweier Objekte reduziert sich schließlich auf die vergleichsweise effiziente Berechnung der Punkt-zu-Punkt-Distanzen aller Mittelpunkte $\psi^{(i)}_1$, $\psi^{(j)}_2$ für $i = 1, \ldots, n^\circ_1$, $j = 1, \ldots, n^\circ_2$, abzüglich der jeweiligen Radien ψ^R_1, ψ^R_2 durch

$$\begin{aligned} &\mathrm{d}\colon \Omega^\circ \times \Omega^\circ \to \mathbb{R}, \\ &(\psi^\circ_1,\, \psi^\circ_2) \mapsto -\psi^R_1 - \psi^R_2 + \min_{i=1,\ldots,n^\circ_1} \min_{j=1,\ldots,n^\circ_2} \|\psi^{(i)}_1 - \psi^{(j)}_2\|_2. \end{aligned} \tag{3.11}$$

Wie bei den Abstandsberechnungen bezüglich Polygonen aus Abschnitt 3.2.1 können hier negative Werte angenommen werden, sodass kein Distanzbegriff im engeren Sinne vorliegt. Tritt dies ein, so liegt im Rahmen der Approximation durch die Kreisüberdeckungen eine Kollision vor.

Im weiteren Verlauf ist vor allem die kürzeste Distanz des Ego-Fahrzeuges, approximiert mit der Kreisüberdeckung $\psi^\circ_{\text{Ego}} \in \Omega^\circ$ aus Abbildung 3.18, zu der Menge aller bekannten Objekte sowie dem Freibereichspolygon relevant. Liegen Informationen zu mehreren Objekten vor, wird dies allgemein in einem n_Ψ-Tupel

$$\Psi^\circ = \{\psi^\circ_1, \ldots, \psi^\circ_{n_\Psi}\} \in \underbrace{\Omega^\circ \times \ldots \times \Omega^\circ}_{n_\Psi} =: (\Omega^\circ)^{n_\Psi}$$

zusammengefasst. Die Distanz des Ego-Fahrzeugs zu diesen Objekten ergibt sich basierend auf (3.11) direkt durch

$$\begin{aligned} &\mathrm{d}\colon \Omega^\circ \times (\Omega^\circ)^{n_\Psi} \to \mathbb{R}, \\ &(\psi^\circ_{\text{Ego}}, \Psi^\circ) \mapsto \min_{\psi^\circ_i \in \Psi^\circ} \mathrm{d}(\psi^\circ_{\text{Ego}}, \psi^\circ_i), \end{aligned} \tag{3.12}$$

für alle $n_\Psi \in \mathbb{N}$. Zusätzlich wird im weiteren Verlauf angenommen, dass zu jedem bekannten Objekt eine prädizierte Bewegung durch die Sensorfusion assoziiert werden kann, vergleiche Abschnitt 1.2. Dies wird durch das zeitabhängige Tupel

$$\begin{aligned} &\Psi^\circ\colon \mathbb{R} \to (\Omega^\circ)^{n_\Psi}, \\ &t \mapsto \Psi^\circ(t) = \{\psi^\circ_1(t), \ldots, \psi^\circ_{n_\Psi}(t)\} \end{aligned}$$

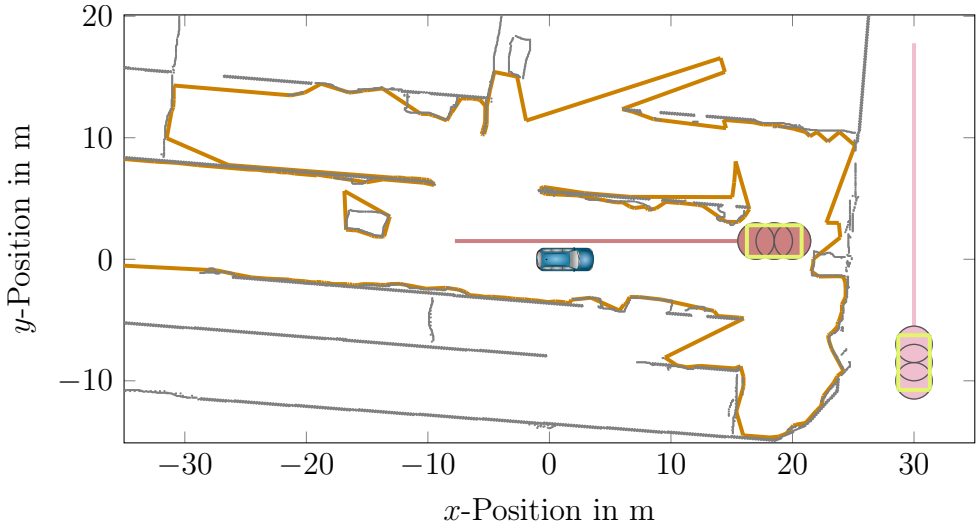


Abbildung 3.19: Beispiele für die Klassifizierung beweglicher Objekte bezüglich des Freibereichspolygons aus Abbildung 3.13. Die rot gefärbte Kreisüberdeckung beschreibt ein aktives, die Überdeckung in rosa ein passives Objekt. Die jeweils zugehörigen orientierten Bounding Boxen sind in gelb dargestellt.

mit

$$\psi_i^{\circ}(t) = \left(\psi_i^{(1)}(t),\, \ldots,\, \psi_i^{(n_i^{\circ})}(t),\, \psi_i^{R}\right), \quad i = 1,\, \ldots,\, n_\Psi$$

dargestellt, wobei die Position der jeweiligen Kreismittelpunkte zeitlich variabel wird, der Radius aber stets konstant bleibt. Analog ist die Entwicklung der orientierten Bounding Boxen der gleichen Objekte durch

$$\begin{aligned} &\Psi^{\square}\colon \mathbb{R} \to (\Omega^{\square})^{n_\Psi}, \\ &t \mapsto \Psi^{\square}(t) = \{\psi_1^{\square}(t), \ldots, \psi_{n_\Psi}^{\square}(t)\} \end{aligned}$$

mit

$$\psi_i^{\square}(t) = \left(\psi_i^{[1]}(t),\, \ldots,\, \psi_i^{[4]}(t)\right), \quad i = 1,\, \ldots,\, n_\Psi$$

gegeben.

Liegt der betrachtete Planungsbereich in Form eines Freibereichspolygons vor, kann die Berechnung in (3.12) durch die Vernachlässigung der *passiven* dynamischen Verkehrsteilnehmer vereinfacht werden. Als passiv wird dabei jedes Objekt bezeichnet, welches sich innerhalb eines vorgegebenen Planungshorizontes niemals (partiell) im Inneren des betrachteten Freibereichspolygons befindet. Alle anderen Objekte heißen entsprechend *aktiv*, vergleiche auch Abbildung 3.19. Der Vergleich zwischen einem Objekt und dem Freibereichspolygon wird dabei basierend auf der Kreisüberdeckung

durchgeführt. Mit

$$\begin{aligned} &\mathrm{d}\colon \Omega^\circ \times \Gamma \to \mathbb{R}, \\ &(\psi^\circ,\, \gamma) \mapsto -\psi^R + \min_{i=1,\dots,n^\circ} \mathrm{d}(\gamma,\, \psi^{(i)}) \end{aligned} \tag{3.13}$$

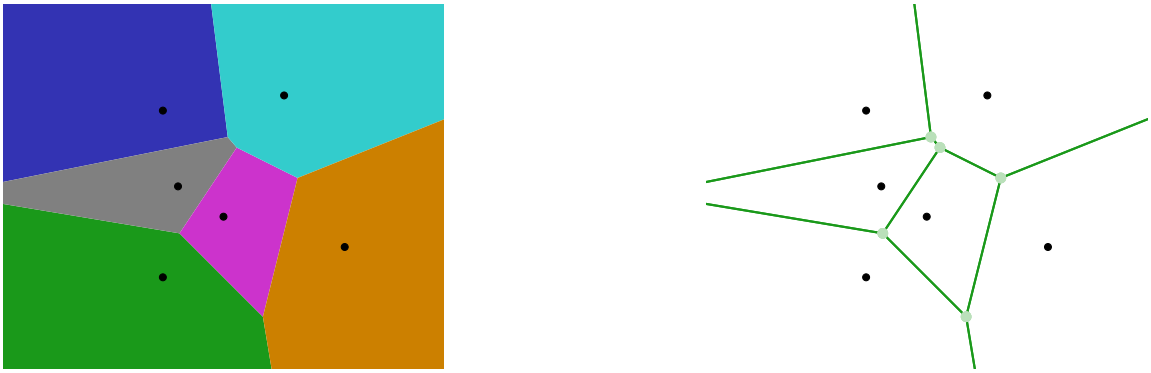
kann dies auf die Abstandsberechnung aus (3.9) zwischen den Mittelpunkten $\psi^{(i)}$ der Kreisüberdeckung und dem Freibereichspolygon reduziert werden. Dies lässt sich ebenfalls auf die Approximation $\psi^\circ_{\mathrm{Ego}}$ des Ego-Fahrzeuges anwenden, sodass schließlich mit (3.12) und (3.13) die notwendigen Grundlagen der Distanzberechnungen zu allen Darstellungen statischer sowie dynamischer Beschränkungen vorliegen. Dies ermöglicht sowohl die Kollisionsdetektion im generischen Hybrid A*-Algorithmus als auch im nächsten Abschnitt 3.3 die Definition des verallgemeinerte Voronoi-Potentials.

3.3 Verallgemeinertes Voronoi-Potential

Ziel dieses Abschnittes ist die Definition eines verallgemeinerten Voronoi-Potentialfeldes, welches sich gegenüber der originalen Variante aus Abschnitt 2.5.3 dadurch hervorhebt, dass

1. beliebige Arten von Hindernissen berücksichtigt werden können und
2. Informationen über das Zeitverhalten von beweglichen Objekten abgebildet werden.

Die Basis dazu wurde in Abschnitt 3.2 durch die abstrahierende Beschreibung aller relevanten statischen und dynamischen Beschränkungen in der Umgebung des Ego-Fahrzeuges geschaffen. Es verbleibt, für diese ein Voronoi-Diagramm zu erstellen. Dafür werden in Abschnitt 3.3.1 die Grundlagen von Voronoi-Diagrammen basierend auf Liniensegmenten gelegt. Dies wird in Abschnitt 3.3.2 zunächst auf die Segmente eines statischen Freibereichspolygons angewendet. Anschließend zeigt Abschnitt 3.3.3, wie dieser Ansatz auch auf zeitabhängige Szenarien verallgemeinert werden kann. Die resultierende verallgemeinerte Voronoi-Potentialfunktion wird schließlich in Abschnitt 3.3.4 vorgestellt. Wie zuvor wird auch hier eine Laufzeitanalyse zur Bewertung der Effizienz durchgeführt, wobei in diesem Zusammenhang der Aufwand zur Konstruktion des Potentialfeldes untersucht wird. Die entsprechenden Auswertungen in Abschnitt 3.3.5 werden durch einen neuen Algorithmus ergänzt, welcher speziell für die in dieser Arbeit vorgestellte verallgemeinerte Potentialfunktion eine wesentlich effizientere Berechnung zulässt.
Die im Vorfeld berechneten Voronoi-Diagramme lassen sich unabhängig vom Voronoi-Potential auch generell zur Analyse der Umgebung nutzen. Abschnitt 3.3.6 zeigt dies am Beispiel der automatischen Generierung von Zielkonfigurationen, welche eine wesentliche Komponente im Planungsalgorithmus in Abschnitt 3.4 spielen wird.



(a) Voronoi-Diagramm **(b)** Voronoi-Kanten und -Knoten

Abbildung 3.20: Ausschnitt des Voronoi-Diagramms sowie dessen Kanten (grün) und Knoten (hellgrün) für Punktdaten (schwarz). Die Zellen des Voronoi-Diagramms sind durch unterschiedliche Farben voneinander abgegrenzt. Der definierende Datenpunkt einer jeden Zelle liegt stets innerhalb dieser und weist die kürzeste Distanz zu allen in der Zelle enthaltenen Punkten auf.

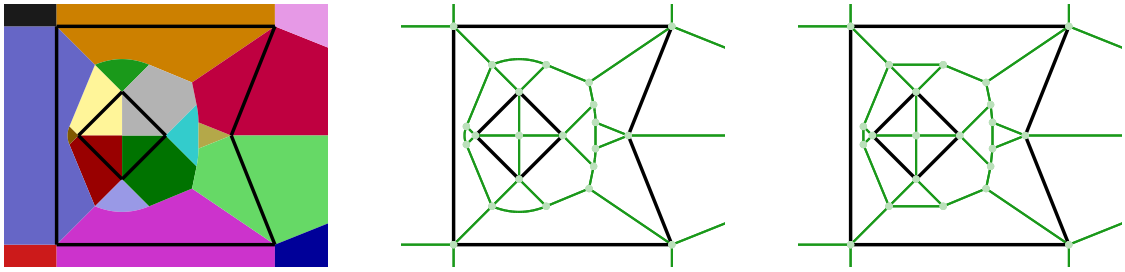
3.3.1 Voronoi-Diagramme für Liniensegmente

Wie in Abschnitt 2.5.3 bereits kurz eingeführt, unterteilt ein zweidimensionales Voronoi-Diagramm die Ebene basierend auf einer vorgegebenen Punktwolke so in Zellen, dass alle Punkte in einer Zelle einen gemeinsamen nächsten Datenpunkt haben. Dies ist exemplarisch für eine kleine Datenmenge in Abbildung 3.20a dargestellt. Die Voronoi-Zellen werden dabei durch Liniensegmente begrenzt, welche im weiteren als *Voronoi-Kanten* $\mathcal{R}$ bezeichnet werden, vergleiche auch Abbildung 3.20b. Liegt keine äußere Begrenzung der betrachteten Koordinaten vor, so liegen die Endpunkte von einigen dieser Kanten im Unendlichen. Jede Voronoi-Kante ist dabei stets mit mindestens einer anderen verbunden; die entsprechenden Verbindungspunkte heißen *Voronoi-Knoten*. Per Konstruktion teilen sich die Punkte einer Kante zwei oder mehrere nächste Datenpunkte und maximieren damit *lokal* den Abstand zu diesen. Die Berechnung eines Voronoi-Diagramms ist zum Beispiel auf Grundlage des Sweepline-Algorithmus [35] möglich. Im Rahmen dieser Arbeit wird dessen Implementierung aus der Softwarebibliothek *Boost* [96] verwendet. Der Sweepline-Algorithmus hat allgemein eine Zeitkomplexität von

$$\mathcal{O}(n \log(n)),$$

wobei n die Anzahl der Punkte in den Eingabedaten beschreibt. Vor diesem Hintergrund werden im Folgenden grundsätzlich nur endliche Mengen von Daten betrachtet, sodass auch das resultierende Voronoi-Diagramm aus endlich vielen Zellen, Kanten und Knoten besteht.

Generell lässt sich dieses Verfahren auch auf die Erstellung eines Voronoi-Diagramms für Liniensegmente anwenden. Dabei muss allerdings garantiert sein, dass Schnitt-



(a) Voronoi-Diagramm

(b) Voronoi-Kanten und -Knoten

(c) Linearisierte Voronoi-Kanten und -Knoten

Abbildung 3.21: Ausschnitt des Voronoi-Diagramms sowie dessen Voronoi-Kanten $\mathcal{R}(\Upsilon)$, deren Linearisierung (beide grün) und Voronoi-Knoten (hellgrün) für Segmentdaten (schwarz)

punkte zwischen diesen Eingabedaten nur auf den jeweiligen Randpunkten der beteiligten Segmente vorliegen. Die wird in der folgenden Definition von zulässigen Eingabesegmenten für ein Voronoi-Diagramm formalisiert.

Definition 3.14 (Zulässige Voronoi-Eingabesegmente)

Die Menge

$$\Xi := \Big\{ \Upsilon \in \Pi(\mathcal{S}) \mid |\Upsilon| < \infty \text{ und } \forall s_1, s_2 \in \Upsilon \colon s_1 \cap s_2 \subset \left\{ s_1^{\langle 1 \rangle}, s_1^{\langle 2 \rangle}, s_2^{\langle 1 \rangle}, s_2^{\langle 2 \rangle} \right\} \Big\} \subset \Pi(\mathcal{S})$$

beschreibt die zulässigen Eingabesegmente zur Erzeugung eines Voronoi-Diagramms. Dabei bezeichnet $|\cdot|$ die Anzahl der Elemente der betrachteten Menge, $\cap$ den Mengenschnitt und $\Pi(\mathcal{S})$ die Potenzmenge aller Liniensegmente.

Die Überprüfung der Schnittbedingung aus Definition 3.14 für eine gegebene Menge von Liniensegmenten kann anhand der Distanzberechnungen aus Abschnitt 3.2.1.2 durchgeführt werden.
Bei der Anwendung des Sweepline-Algorithmus auf gegebene Daten $\Upsilon \in \Xi$ werden sowohl die Segmente selbst, als auch deren Endpunkte als zu berücksichtigende Instanzen interpretiert. Dies führt dazu, dass die entstehenden Zellen durch ein gemeinsames nächstes Segment *oder* einen gemeinsamen nächsten Endpunkt charakterisiert werden, wie in Abbildung 3.21a gut zu erkennen ist. Da Voronoi-Kanten in diesem Fall auch zwischen Punkten und Segmenten entstehen, können sie insbesondere auch gekrümmt sein, vergleiche Abbildung 3.21b. Zudem können auch hier Endpunkte im Unendlichen liegen. Um die Komplexität zu reduzieren, werden sie im Folgenden zwischen ihren jeweiligen Randpunkten zu Liniensegmenten *linearisiert*. Dabei entsteht ein kleiner Approximationsfehler, wie in Abbildung 3.21c exemplarisch gezeigt, während andererseits zum Beispiel die sehr einfache Distanzberechnung

zu Punkten aus dem $\mathbb{R}^2$ gemäß Abschnitt 3.2.1.2 ermöglicht wird. Werden zusätzlich die Elemente mit unendlichen Endpunkten verworfen, so heißen die resultierenden Segmente reguläre Voronoi-Kanten.

Definition 3.15 (Reguläre Voronoi-Kanten)

Sei $\Upsilon \in \Xi$ eine Menge von Segmentdaten. Dann bezeichnet $\mathcal{R}^\diamond \in \Pi(\mathcal{S})$ mit

$$\mathcal{R}^\diamond(\Upsilon) \coloneqq \Big\{ s = (s^{\langle 1 \rangle}, s^{\langle 2 \rangle})_{\mathcal{S}} \in \mathcal{S}, \mid \sup_{p \in s}(\|p\|_2) < \infty, \\ s^{\langle 1 \rangle}, s^{\langle 2 \rangle} \text{ Randpunkte einer Voronoi-Kante in } \mathcal{R}(\Upsilon) \Big\}$$

die Menge der *regulären Voronoi-Kanten* von Υ.

3.3.2 Voronoi-Pfad im Freibereichspolygon

Die Anwendbarkeit des Sweepline-Algorithmus auf Segmentdaten erlaubt schließlich die Erstellung von regulären Voronoi-Kanten für ein Freibereichspolygon $\gamma \in \Gamma$ und damit für alle statischen Komponenten in der Fahrzeugumgebung. Dabei werden die Eingabedaten als Menge der Segmente aller enthaltenen Polygonzüge beschrieben.

Definition 3.16 (Zulässige Eingabesegmente eines Freibereichspolygons)

Sei $\gamma \in \Gamma$ ein Freibereichspolygon, für welches alle Teilpolygone keine überlappenden Liniensegmente haben. Dann definiert

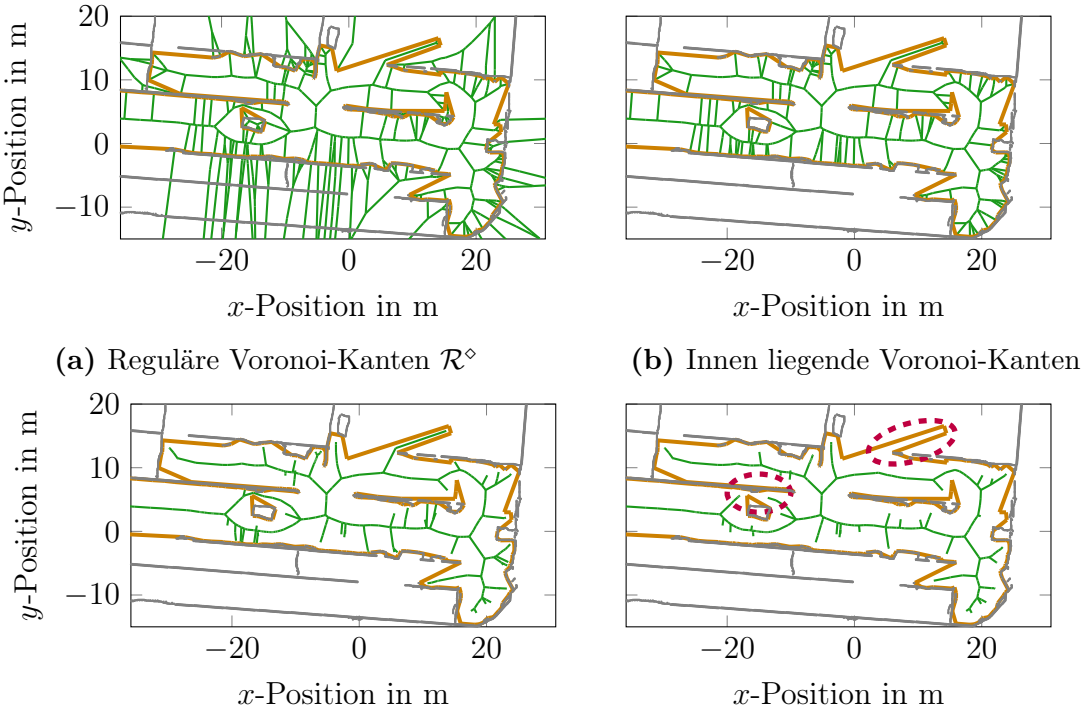
$$\Upsilon_\gamma \coloneqq \{ s \in \mathcal{S} \mid s \in \rho, \rho \in \gamma \} \in \Xi$$

die zulässigen Eingabesegmente für ein Voronoi-Diagramm aus den Segmenten des Freibereichspolygons.

Anmerkung 3.17 Für ein gemäß Algorithmus 3.1 erzeugtes Freibereichspolygon ist die Abwesenheit von überlappenden Liniensegmenten durch die Anwendung der Polygonvereinfachung `Simplify` garantiert, vergleiche auch [51].

Wie reale Punktwolken aus Sensordaten ist auch ein darauf basierendes Freibereichspolygon jedoch oft weit weniger gradlinig als im Beispiel aus Abbildung 3.21, sondern weist vergleichsweise viele kleine Knicke auf. Die regulären Kanten aus einem zugehörigen Voronoi-Diagramm passen sich an diese Vorgabe an, sodass relativ komplizierte Strukturen wie in Abbildung 3.22a entstehen können. Da das Innere des Freibereichspolygons den aktuell befahrbaren Bereich beschreibt, können zunächst alle Voronoi-Kanten entfernt werden, welche außerhalb liegen, wie in Abbildung 3.22b dargestellt. Die verbleibenden Kanten können in zwei Kategorien unterteilt werden:

1. Diejenigen Kanten, welche parallel zwischen den Segmenten des Freibereichspolygons verlaufen und damit lokal den Abstand zu diesem maximieren.



(a) Reguläre Voronoi-Kanten $\mathcal{R}^{\diamond}$

(b) Innen liegende Voronoi-Kanten

(c) Strikt innen liegende Voronoi-Kanten

(d) Voll gefilterter Voronoi-Pfad $\mathcal{R}^{*}$

Abbildung 3.22: Linearisierte Voronoi-Kanten $\mathcal{R}$ und deren Filterung zum Voronoi-Pfad $\mathcal{R}^{*}$ für das Freibereichspolygon aus Abbildung 3.13. Es werden alle Kanten verworfen, die außerhalb liegen, beziehungsweise einen Endpunkt mit einer Distanz von unter 1 m zum Freibereichspolygon haben. Die in dunkelrot gestrichelten Ellipsen heben ausgewählte Bereiche hervor, in denen durch die Filterung des Voronoi-Pfades keine Kanten mehr liegen.

Sie entsprechen der Art von Kanten, welche auch im Kontext des originalen Hybrid A*-Algorithmus zur Erstellung des Voronoi-Potentials verwendet werden, vergleiche Abbildung 2.12 aus Abschnitt 2.5.3.

2. Die von diesen abzweigenden Kanten, welche eine Verbindung zu den Endpunkten der Segmente des Freibereichspolygons schaffen und sich damit nicht als Navigationsgrundlage für einen Algorithmus zur Pfad- oder Bewegungsplanung eignen.

Werden alle regulären Voronoi-Kanten der 2. Kategorie entfernt, so resultiert die Struktur aus Abbildung 3.22c. Es verbleiben keine Elemente mehr, die einen gemeinsamen Endpunkt mit einem Segment aus denn Eingabedaten besitzen, jedoch existieren weiterhin einige quer zur Streckenführung liegende Pfade. Um einen noch stärkeren Filtereffekt zu erzielen bietet es sich daher an, alle regulären Voronoi-Kanten mit Distanz zum Freibereichspolygon unter einem gegebenen Schwellenwert $\delta_{\mathcal{R}}$ eben-

falls auszusortieren. Das Ergebnis ist in Abbildung 3.22d dargestellt und wird als Voronoi-Pfad bezeichnet.

Definition 3.18 (Voronoi-Pfad)

Für ein Freibereichspolygon γ heißt

$$\mathcal{R}^*(\gamma) \coloneqq \left\{ (s^{\langle 1 \rangle}, s^{\langle 2 \rangle})_{\mathcal{S}} \in \mathcal{R}^\diamond(\Upsilon_\gamma) \,\middle|\, \mathrm{d}(\gamma, s^{\langle 1 \rangle}) \geq \delta_{\mathcal{R}} \text{ und } \mathrm{d}(\gamma, s^{\langle 2 \rangle}) \geq \delta_{\mathcal{R}} \right\} \in \Pi(\mathcal{S})$$

Voronoi-Pfad.

Da Punkten außerhalb des Freibereichspolygons nach (3.9) eine negative Distanz zugeordnet wird, schließt Definition 3.18 alle in diesem Abschnitt diskutierten Filterschritte mit ein. Da die Distanzen aller regulärer Voronoi-Kanten zu allen Segmenten des Freibereichspolygons berechnet werden müssen, ergibt sich zur Berechnung von $\mathcal{R}^*(\gamma)$ insgesamt eine Zeitkomplexität von

$$\mathcal{O}(n \cdot m + n \log n), \quad \text{mit} \quad n = |\gamma|,\, m = |\mathcal{R}^\diamond(\Upsilon_\gamma)|, \tag{3.14}$$

wobei $|\cdot|$ jeweils die Anzahl der enthaltenen Liniensegmente beschreibt.

Orientiert sich die Größe des Schwellenwertes $\delta_{\mathcal{R}}$ an der Ausdehnung des Ego-Fahrzeuges, dann werden auch reguläre Voronoi-Kanten von zu engen und damit nicht befahrbaren Passagen direkt aussortiert. In Abbildung 3.22 gilt dies zum Beispiel für die beiden dunkelrot markierten Bereiche. Um dabei keine sinnvollen Kanten aus Kategorie 1 zu entfernen, darf der Schwellenwert jedoch nicht zu groß gewählt werden. Diese Beschränkung führt letztlich dazu, dass selbst mit einer sorgfältigen Abwägung in der Regel nicht alle abzweigenden Kanten verworfen werden können. Damit werden sie entsprechend auch bei der Definition des verallgemeinerten Voronoi-Potentials in Abschnitt 3.3.4 berücksichtigt. Wie zuvor beschrieben, verlaufen sie stets quer zu den primär zu folgenden Kanten, was sich durch die Kinematik aus Abschnitt 3.1.1 in der Regel nicht umsetzen lässt. Daher kann erwartet werden, dass diese Artefakte nur sehr geringen Einfluss auf das Ergebnis eines Planungsalgorithmus haben. Diese Vermutung wird für den generischen Hybrid A*-Algorithmus exemplarisch in Abschnitt 4.1.3.1 analysiert.

3.3.3 Dynamischer Voronoi-Pfad

Ein wesentlicher Unterschied des im Folgenden zu definierenden verallgemeinerten Voronoi-Potentials zum originalen Ansatz aus Abschnitt 2.5.3 besteht in der Berücksichtigung der bekannten dynamischen Komponenten in der Umgebung. Deren zeitabhängige Lage wird dazu anhand von diskreten Stützstellen

$$t \in \{t_0, t_1, \ldots, t_{\max}\} \subset \mathbb{R}$$

approximiert, wobei $t_{\max}$ einem (geschätzten) maximalen Zeithorizont entspricht. Anhand der prädizierten Dynamik der Objekte wird deren räumliche Ausdehnung zur Zeit t durch Liniensegmente dargestellt und als zusätzliche Datengrundlage im Sweepline-Algorithmus aus dem vorigen Abschnitt 3.3.2 verwendet. Dafür eignet sich besonders die Approximation durch Bounding Boxen $\Psi^{\square}$, wie in 3.2.4 beschrieben.

Definition 3.19 (Zulässige Eingabesegmente durch Bounding Boxen)

Sei $\Psi^{\square} \in (\Omega^{\square})^{n_\Psi}$ ein Tupel von Bounding Boxen. Sei $\Psi^{\square}$ zudem ohne paarweise Überlappungen. Dann definiert

$$\Upsilon_\Psi \coloneqq \Big\{ s_{i,j} \in \mathcal{S} \mid s_{i,j} \text{ Kante der Bounding Box } \psi^{\square} \text{ gemäß Definition 3.12}, \\ \psi^{\square} \in \Psi^{\square} \Big\} \in \Xi$$

die zulässigen Eingabesegmente für ein Voronoi-Diagramm aus den Kanten der Bounding Boxen.

Bewegliche Objekte können grundsätzlich die Grenzen des betrachteten Freibereichspolygons überschreiten, sodass für die Vereinigung von Υ_γ und Υ_Ψ nicht trivialerweise Überschneidungsfreiheit garantiert werden kann.

Definition 3.20 (Kombinierte zulässige Eingabesegmente)

Seien $\gamma \in \Gamma$ ein Freibereichspolygon und $(\Omega^{\square})^{n_\Psi}$ ein Tupel von Bounding Boxen gemäß Definitionen 3.16 und 3.19. Sei

$$\Upsilon_{\gamma,\Psi} \coloneqq \Upsilon_\gamma \cup \Upsilon_\Psi \setminus \{ s_\Psi \in \Upsilon_\Psi \mid \{s_\Psi\} \cup \Upsilon_\gamma \notin \Xi \}$$

die Menge aller Segmente aus Υ_γ und Υ_Ψ ohne diejenigen Bounding Box-Kanten, die zu einem Konflikt mit den Elementen des Freibereichspolygons gemäß Definition 3.14 führen. Dann beschreibt $\Upsilon_{\gamma,\Psi} \in \Xi$ die Menge der kombinierten zulässigen Eingabesegmente für ein Voronoi-Diagramm.

Anmerkung 3.21 Kann für die Objektdaten nicht garantiert werden, dass die gegebenen Bounding Boxen für alle betrachtete Zeiten überlappungsfrei sind, dann kann das Filterverfahren aus Definition 3.20 auch für deren Kanten eingesetzt werden, um stets eine Menge aus zulässigen Eingabesegmenten zu erhalten. Dabei muss allerdings beachtet werden, dass Teile der vorhandenen Informationen gegebenenfalls verworfen werden. Alternativ können mit zusätzlichem numerischem Aufwand auch Konflikte zwischen allen Segmenten berechnet und die betroffenen Elemente so modifiziert werden, dass Segmente an Schnittpunkten getrennt beziehungsweise bei Überlappungen zusammengefasst werden. Bei diesem Vorgehen entstehen keine Verluste bezüglich des Informationsgehalts der vorhandenen Daten.

Schließlich wird basierend auf vorstehenden Definitionen eine verallgemeinerte Variante des Voronoi-Pfades definiert.

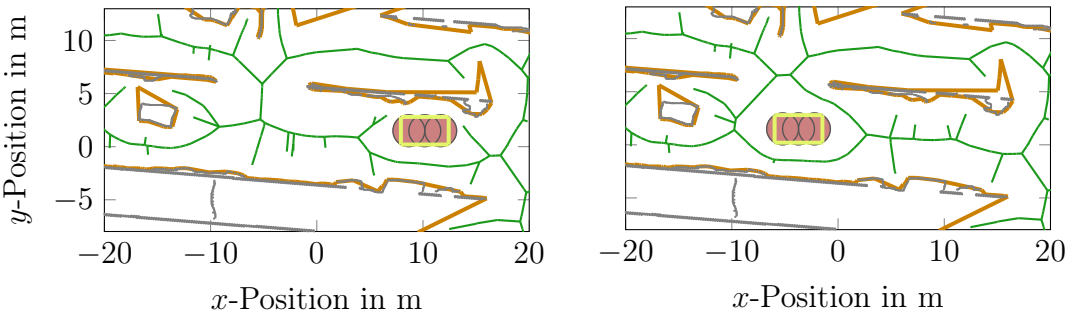


Abbildung 3.23: Dynamischer Voronoi-Pfad $\mathcal{R}_t^*$ basierend auf dem Freibereichspolygon aus Abbildung 3.13 sowie der orientierten Bounding Box $\psi^{\square}$ (in gelb) eines Fahrzeuges für zwei unterschiedliche Zeiten. Zum Vergleich wird ebenfalls die zugehörige Kreisüberdeckung ψ° in rot dargestellt.

Definition 3.22 (Dynamischer Voronoi Pfad)

Für ein Freibereichspolygon $\gamma \in \Gamma$ und eine Menge von Objekten, approximiert durch Bounding Boxen $\Psi^{\square} \in (\Omega^{\square})^{n_\Psi}$, sei der Voronoi-Pfad durch

$$\mathcal{R}^*\left(\gamma,\, \Psi^{\square}\right) := \Big\{(s^{\langle 1\rangle},\, s^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{R}^{\diamond}(\Upsilon_{\gamma,\Psi}) \;\big|\, \mathrm{d}(\gamma,\, s^{\langle 1\rangle}) \geq \delta_{\mathcal{R}} \text{ und } \mathrm{d}(\gamma,\, s^{\langle 2\rangle}) \geq \delta_{\mathcal{R}}\Big\} \in \Pi\left(\mathcal{S}\right)$$

definiert. Für eine festes Freibereichspolygon und zeitabhängige Daten $\Psi^{\square}(t)$ bezeichnet

$$\mathcal{R}_t^* := \mathcal{R}^*\left(\gamma,\, \Psi^{\square}(t)\right) \in \Pi\left(\mathcal{S}\right)$$

den *dynamischen Voronoi-Pfad* zur Zeit t.

In Abbildung 3.23 ist exemplarisch dargestellt, wie sich dieser an die Bewegung eines Objektes anpasst. Generell muss $\mathcal{R}_t^*$ für die folgenden Auswertungen für jeden Zeitschritt $t \in \{t_0, t_1, \ldots, t_{\max}\}$ berechnet werden. Da die einzelnen diskreten Zeitpunkte dabei unabhängig voneinander sind, kann dies allerdings sehr effizient parallelisiert werden, wenn die notwendigen Rechenkapazitäten dafür verfügbar sind. Ein besserer Ansatz ergibt sich durch das Ausnutzen der Eigenschaften von Voronoi-Diagrammen: Die Bewegung eines dynamischen Objektes beeinflusst generell nur diejenigen Voronoi-Kanten in dessen unmittelbarer Umgebung. Entsprechend ergeben sich beim Vergleich von Abbildung 3.23 zum statischen Pfad aus Abbildung 3.22d nur jeweils lokale Unterschiede dort, wo das Objekt sich zum jeweiligen Zeitpunkt befindet. Diese Erkenntnis kann für eine effizientere Berechnung des dynamischen Voronoi-Pfads ausgenutzt werden. Dies wird in Abschnitt 3.3.5 genauer betrachtet, nachdem zunächst das verallgemeinerte Voronoi-Potential in Abschnitt 3.3.4 eingeführt wurde.

3.3.4 Die verallgemeinerte Voronoi-Potentialfunktion

In den vorigen Abschnitten wurden Darstellungsformen für statische und dynamische Hindernisse sowie der Voronoi-Pfad als Planungsreferenz eingeführt. Die Informationen aus all diesen Quellen werden schließlich im verallgemeinerten Voronoi-Potential zusammengefasst, welches eine direkte Erweiterung der in Abschnitt 2.5.3 vorgestellten Potentialfunktion darstellt. Dabei werden die zuvor definierten (generalisierten) Distanzen einer Ego-Fahrzeug-Überdeckung $\psi^{\circ}_{\text{Ego}}$ zu einem Freibereichspolygon γ sowie zu den Kreisüberdeckungen einer Menge von n_Ψ Objekten Ψ° via (3.12) und (3.13) benötigt. Zur Vereinfachung der Schreibweise wird für diese eine gemeinsame Distanzfunktion durch

$$\begin{aligned} &\mathrm{d}\colon \Omega^{\circ} \times \Gamma \times (\Omega^{\circ})^{n_\Psi} \to \mathbb{R}, \\ &(\psi^{\circ}_{\text{Ego}},\, \gamma,\, \Psi^{\circ}) \mapsto \min\Big(\mathrm{d}(\psi^{\circ}_{\text{Ego}},\, \gamma),\, \mathrm{d}(\psi^{\circ}_{\text{Ego}},\, \Psi^{\circ})\Big) \end{aligned} \tag{3.15}$$

formuliert. Dabei ist zu beachten, dass Kollisionen mit einem Freibereichspolygon zu negativen Abständen führen. Um dies im Folgenden differenziert berücksichtigen zu können, wird zusätzlich eine nach unten beschränkte Distanzfunktion durch

$$\mathrm{d}^{+}(\cdot) \coloneqq \max(0,\, \mathrm{d}(\cdot))$$

verwendet. Auch die Kanten des Voronoi-Pfades $\mathcal{R}^*$ müssen wie im originalen Ansatz aus Abschnitt 2.5.3 berücksichtigt werden. Dazu wird ein entsprechender Abstand gemäß

$$\begin{aligned} &\mathrm{d}\colon \Omega^{\circ} \times \Pi\,(\mathcal{S}) \to \mathbb{R}_{\geq 0}, \\ &(\psi^{\circ}_{\text{Ego}},\, \mathcal{R}^*) \mapsto \min_{s \in \mathcal{R}^*} \mathrm{d}(s, p_{\text{Ego}}(\psi^{\circ}_{\text{Ego}})) \end{aligned}$$

als Minimaldistanz aller enthaltenen Segmente zum Referenzpunkt $p_{\text{Ego}}(\psi^{\circ}_{\text{Ego}}) \in \mathbb{R}^2$ des Ego-Fahrzeuges definiert, durch welchen im Folgenden die *Nähe* zum Voronoi-Pfad beschrieben wird. Die vorstehenden Distanzbegriffe werden zuletzt in

$$\begin{aligned} &\mathrm{d}\colon \Omega^{\circ} \times \Gamma \times (\Omega^{\circ})^{n_\Psi} \times \Pi\,(\mathcal{S}) \to [0,1] \subset \mathbb{R}, \\ &(\psi^{\circ}_{\text{Ego}},\, \gamma,\, \Psi^{\circ},\, \mathcal{R}^*) \mapsto \begin{cases} \dfrac{1}{2}, & \text{falls } \mathrm{d}^{+}(\psi^{\circ}_{\text{Ego}},\, \gamma,\, \Psi^{\circ}) = \mathrm{d}(\psi^{\circ}_{\text{Ego}},\, \mathcal{R}^*) = 0, \\ \dfrac{\mathrm{d}(\psi^{\circ}_{\text{Ego}},\, \mathcal{R}^*)}{\mathrm{d}(\psi^{\circ}_{\text{Ego}},\, \mathcal{R}^*) + \mathrm{d}^{+}(\psi^{\circ}_{\text{Ego}},\, \gamma,\, \Psi^{\circ})}, & \text{sonst}, \end{cases} \end{aligned} \tag{3.16}$$

zusammengefasst. Für den Fall, dass sowohl die Abstände zu den Beschränkungen als auch zum Voronoi-Pfad verschwinden, wird der Wert der Distanzfunktion mit 0,5 festgelegt. Auf Grundlage dieser Notation wird schließlich das verallgemeinerte Voronoi-Potential eingeführt.

Definition 3.23 (Verallgemeinertes Voronoi-Potential)

Sei ein Freibereichspolygon $\gamma \in \Gamma$, die Kreisüberdeckungen $\Psi^\circ(t) \in (\Omega^\circ)^{n_\Psi}$ einer Menge von n_Ψ Objekten sowie ein dynamischer Voronoi-Pfad $\mathcal{R}_t^* \in \Pi(\mathcal{S})$ gegeben. Sei zudem $\alpha \in \mathbb{R}_{>0}$ und $\mathrm{d}^{\max} \in \mathbb{R}_{\geq 0}$, dann ist das *verallgemeinerte Voronoi-Potential* für eine Kreisüberdeckung $\psi_{\mathrm{Ego}}^\circ \in \Omega^\circ$ des Ego-Fahrzeugs zur Zeit $t \in \mathbb{R}$ durch

$$\Phi_V^* \colon \Omega^\circ \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\mathrm{Ego}}^\circ,\, t) \mapsto \begin{cases} 0, & \text{falls } \mathrm{d}(\psi_{\mathrm{Ego}}^\circ,\, \gamma,\, \Psi^\circ(t)) \geq \mathrm{d}^{\max}, \\ 1, & \text{falls } \mathrm{d}(\psi_{\mathrm{Ego}}^\circ,\, \Psi^\circ(t)) \leq 0, \\ \overline{\Phi}_V^*(\psi_{\mathrm{Ego}}^\circ,\, t), & \text{sonst}, \end{cases}$$

mit

$$\overline{\Phi}_V^* \colon \Omega^\circ \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\mathrm{Ego}}^\circ,\, t) \mapsto \frac{\alpha}{\alpha + \mathrm{d}^+(\psi_{\mathrm{Ego}}^\circ,\, \gamma,\, \Psi^\circ(t))} \cdot \mathrm{d}(\psi_{\mathrm{Ego}}^\circ,\, \gamma,\, \Psi^\circ(t),\, \mathcal{R}_t^*) \cdot \frac{\mathrm{d}^{\max} - \mathrm{d}(\psi_{\mathrm{Ego}}^\circ,\, \gamma,\, \Psi^\circ(t))}{\mathrm{d}^{\max}}$$

definiert.

Dieses bewertet die getestete Konfiguration des Ego-Fahrzeuges zur gegebenen Zeit in Bezug auf sämtliche räumliche Beschränkungen sowie den Voronoi-Pfad. Der Parameter $\alpha \in \mathbb{R}_{>0}$ steuert dabei, wie in der Definition von Φ_V aus (2.5), den Anstieg des Potentials zwischen dem Voronoi-Pfad und dem Freibereichspolygon. Die maximal berücksichtigte Distanz $\mathrm{d}^{\max} \in \mathbb{R}_{\geq 0}$ lässt das Potential zudem bei hinreichend großem Abstand zu den Beschränkungen verschwinden. Ansonsten skaliert es den Verlauf auf einen Wertebereich von $[0,\, 1)$ für alle Konfigurationen, in denen kein Konflikt mit dem Freibereichspolygon oder den beweglichen Objekten besteht. Liegt eine Kollision mit letzteren vor, wird ein Potentialwert von 1 festgelegt.

Eine wesentliche Neuerung ergibt sich im Konfliktfall mit statischen Hindernissen, in welchem das verallgemeinerte Voronoi-Potential ebenfalls eine differenzierte Bewertung der Fahrzeugkonfiguration erlaubt. Dabei wird ausgenutzt, dass Positionen außerhalb eines Freibereichspolygons negative Distanzen zu diesem haben. Dies wird im letzten Term von $\overline{\Phi}_V^*$ berücksichtigt, indem, im Vergleich mit dem originalen Ansatz, die Quadrierung des Ausdrucks verworfen wird. Im Fall von Konflikten zwischen Ego-Fahrzeug und Freibereichspolygon führt dies letztlich zu Potentialwerten von $\Phi_V^* > 1$ und einem linearen Anstieg, proportional zum Betrag der negativen Distanz. Ein exemplarischer Verlauf für verschiedene Parameter der Potentialfunktion ist in Abbildung 3.24 dargestellt. Es lässt sich gut die dämpfende Wirkung des Parameters α in konfliktlosen Konfigurationen erkennen. Zudem ist der Einfluss des maximalen Abstands $\mathrm{d}^{\max}$ sehr auffällig. Für die Parameterwahl $\mathrm{d}^{\max} = 4\,\mathrm{m}$ ergibt sich im In-

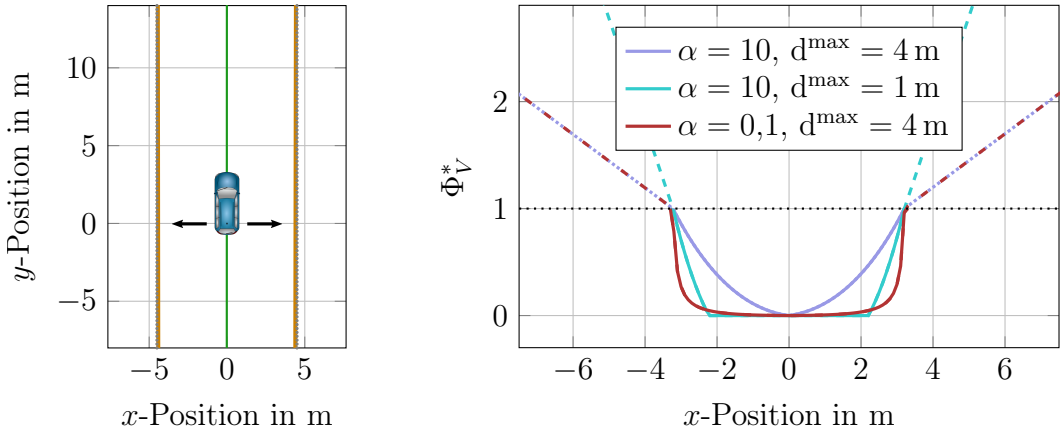
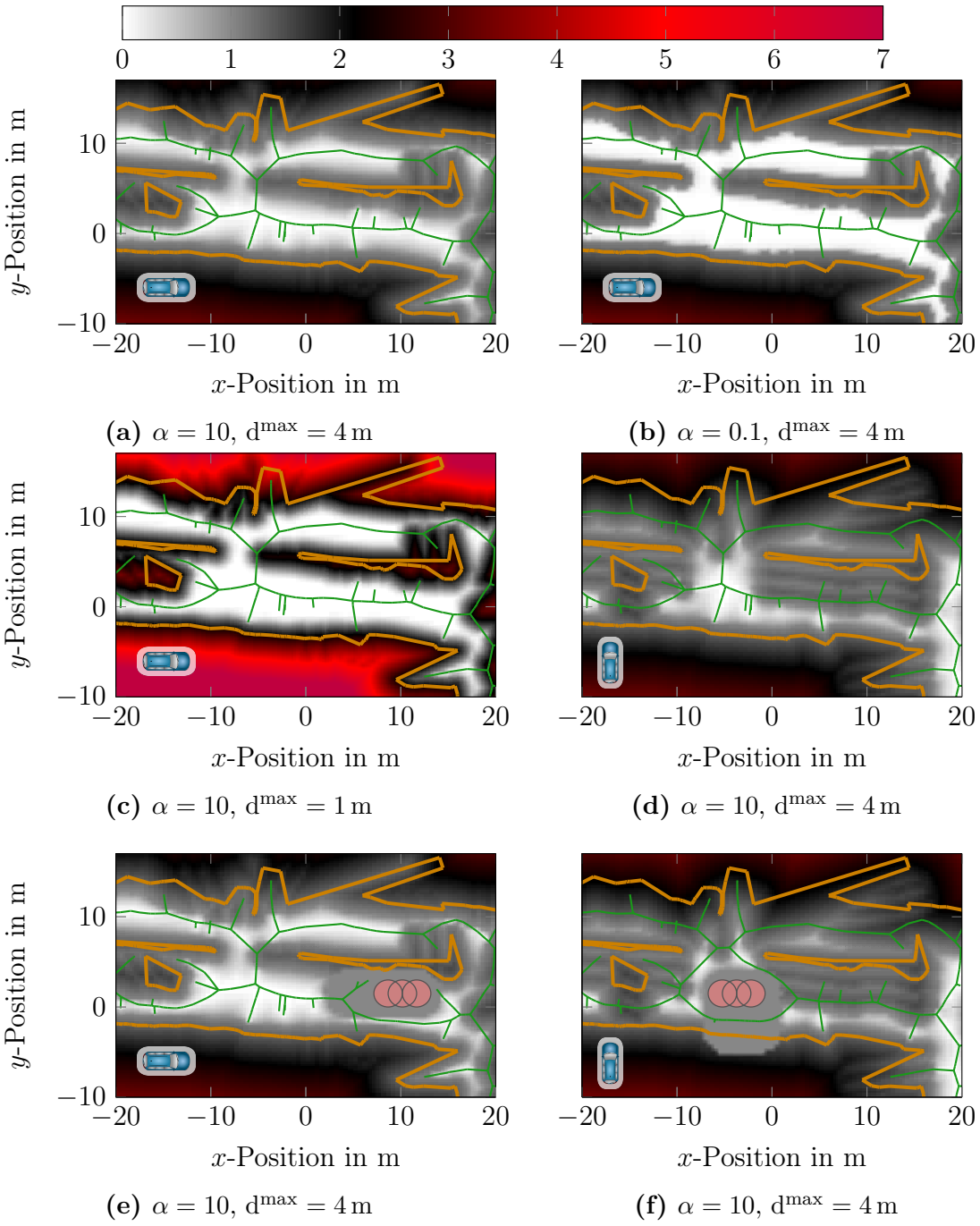


Abbildung 3.24: Exemplarischer, eindimensionaler Verlauf des verallgemeinerten Voronoi-Potentials Φ_V^* in einem linear verlaufenden Freibereichspolygon (in orange) für verschiedene x-Positionen des Ego-Fahrzeugs und für unterschiedliche Werte von α und d^{max}. Die gestrichelten Verläufe beschreiben jeweils den Potentialwert, wenn die Fahrzeugposition im Konflikt mit dem Freibereichspolygon ist. Als Referenzpunkt p_{Ego} zum Vergleich mit dem Voronoi-Pfad (in grün) wird das Zentrum der Hinterachse verwendet.

neren des Polygons ein durchweg knickfreier Verlauf, da dieser Abstand aufgrund der Fahrzeugbreite nicht angenommen werden kann. Wird hingegen ein erreichbarer Wert $d^{max} = 1\,m$ gewählt, ergibt sich ein nicht differenzierbarer Übergang an den Stellen mit entsprechendem Abstand. Da d^{max} wie zuvor erwähnt ebenfalls skalierende Wirkung hat, folgt dann zusätzlich ein deutlich steilerer Potentialanstieg in der Nähe des Polygons sowie auch außerhalb.

Die differenzierte Bewertung von Fahrzeugkonfigurationen im Inneren und Äußeren des Freibereichspolygons erweitert die Anwendbarkeit des Potentials. So können zum Beispiel neben *harten* statischen Beschränkungen ebenfalls *weiche* Vorgaben wie Spurinformationen berücksichtigt werden. Das Potential sorgt in diesem Fall für minimale Kosten im Inneren der Fahrspur, erlaubt aber auch abweichende Lösungen, wenn andere Anforderungen (wie zum Beispiel harte Hindernisse) dies erfordern. Der Einfluss dieser Gestaltungsmöglichkeiten auf die Planung mit dem generischen Hybrid A* wird in Abschnitt 4.2.1 genauer beschrieben.

Neben dem Definitionsbereich erweitert das verallgemeinerte Voronoi-Potential ebenfalls die Menge der berücksichtigten Informationen bezüglich des Ego-Fahrzeuges. Dies betrifft vor allem die Distanzberechnung basierend auf $d(\psi_{Ego}^{\circ}, \gamma, \Psi^{\circ}(t))$, welche nicht nur dessen Ausrichtung, sondern auch die räumliche Ausdehnung (basierend auf der Kreisüberdeckung) vollständig einbezieht. Der Einfluss dieser Modellierung kann in Abbildung 3.25 beobachtet werden. Dort ist der Verlauf des Potentials in 3.25a bis 3.25c noch einmal für die Parametervariation aus Abbildung 3.24 dargestellt. Neben den zuvor beschriebenen Effekten fällt hier



(a) $\alpha = 10$, $\mathrm{d}^{\max} = 4\,\mathrm{m}$ **(b)** $\alpha = 0.1$, $\mathrm{d}^{\max} = 4\,\mathrm{m}$

(c) $\alpha = 10$, $\mathrm{d}^{\max} = 1\,\mathrm{m}$ **(d)** $\alpha = 10$, $\mathrm{d}^{\max} = 4\,\mathrm{m}$

(e) $\alpha = 10$, $\mathrm{d}^{\max} = 4\,\mathrm{m}$ **(f)** $\alpha = 10$, $\mathrm{d}^{\max} = 4\,\mathrm{m}$

Abbildung 3.25: Verlauf des verallgemeinerten Voronoi-Potentials Φ_V^* bezüglich des Freibereichspolygons aus Abbildung 3.13 bei Variation der Parameter wie in Abbildung 3.24, für verschiedene Orientierungen des Ego-Fahrzeuges sowie für statische und dynamische Szenarien. Die verwendete Orientierung wird jeweils durch das abgesetzte Fahrzeug an der unteren linken Seite gekennzeichnet. Als Referenzpunkt p_{Ego} zur Berechnung von Distanzen zum Voronoi-Pfad wird das Zentrum der Hinterachse verwendet.

besonders auf, dass das Minimum des Potentials sich an einigen Stellen nicht mit dem Verlauf des Voronoi-Pfades deckt. Zwar gilt $\Phi_V^* = 0$ nur, wenn der Referenzpunkt p_{Ego} auf diesen Segmenten liegt; je nach Ausrichtung des Ego-Fahrzeuges führt dies allerdings zu einem Konflikt mit dem Freibereichspolygon. Werden durch dieses weiche Hindernisse repräsentiert, dann impliziert dies keine Kollision und führt zu nicht verschwindenden Potentialwerten. Der Spezialfall in diesem Kontext, dass sowohl $\mathrm{d}(\psi^\circ_{\text{Ego}}, \mathcal{R}^*) = 0$ als auch $\mathrm{d}^+(\psi^\circ_{\text{Ego}}, \gamma, \Psi^\circ) = 0$ gelten, wird in der Definition von d in (3.16) beachtet. Der Einfluss der Orientierung des Ego-Fahrzeuges auf das Potential wird schließlich besonders durch einen Vergleich zwischen den Abbildungen 3.25a und 3.25d deutlich, für die ansonsten alle Hyperparameter identisch sind. Während größere Freiflächen in beiden Fällen weiß gekennzeichnet sind, hängt dies in Engstellen entsprechend von der Fahrzeugausrichtung ab. Da der Referenzpunkt p_{Ego} in dieser Arbeit auf der Hinterachse des Fahrzeuges liegt, muss vor allem *vor* dem Voronoi-Pfad in der jeweils betrachteten Richtung genügend Freifläche vorhanden sein.

Die letzte wesentliche Neuerung von Φ_V^* besteht im Einbeziehen von dynamischen Objekten, sodass dessen Auswertung zeitabhängig wird. Die Änderung des Potentials basierend auf den Beispielen aus Abbildung 3.23 ist in den Abbildungen 3.25e und 3.25f für zwei unterschiedliche Ausrichtungen des Ego-Fahrzeuges dargestellt. Da die Distanzfunktion für bewegliche Objekte – anders als beim Freibereichspolygon – nicht negativ werden kann, gilt für das Potential in jeder entsprechenden Konfliktkonfiguration $\Phi_V^* = 1$. Dies spiegelt insbesondere wider, dass in diesem Fall stets eine Kollision vorliegt und damit keine differenzierte Bewertung notwendig ist. Im Potentialfeld drückt sich dies dadurch aus, dass dynamische Objekte von einheitlich gefärbten Sphären umgeben sind, deren Form von der Orientierung des Ego-Fahrzeuges abhängen.

Zur Auswertung des verallgemeinerten Voronoi-Potentials in einem Planungsalgorithmus zur Zeit t muss der zugehörige dynamische Voronoi-Pfad $\mathcal{R}_t^*$ vorliegen. Dieser kann zwar im Vorfeld für viele Zeitpunkte bereitgestellt werden, allerdings ist zum Start des Verfahrens nicht zwingend die letztlich benötigte Endzeit $t_{\max}$ bekannt. Nicht vorliegende Pfade müssen entsprechend gegebenenfalls zur Laufzeit nachgereicht werden. Dabei spielt eine Abwägung der Wahl von $t_{\max}$ eine wichtige Rolle: Werden die Voronoi-Pfade von zu hohen, nicht relevanten Zeitpunkten vorberechnet, so ist die aufgewendete Rechenleistung ohne Mehrwert geblieben. Werden dagegen zu wenige Pfade im Vorfeld bestimmt, müssen diese zur Laufzeit des Verfahrens ausgewertet werden. Dies ist tendenziell aufwändiger, da weniger Möglichkeiten zur Optimierung vorliegen, wie zum Beispiel eine Parallelisierung über alle betrachteten Zeitpunkte. Dies wird im folgenden Abschnitt 3.3.5 genauer untersucht.

3.3.5 Zeitkomplexität der Voronoi-Pfad-Generierung

Voronoi-Pfade stellen als Grundlage des Voronoi-Potentials primär eine Referenz für die Planung mit dem in dieser Arbeit entwickelten Hybrid A*-Algorithmus dar; dies wird in Abschnitt 3.4 genauer vorgestellt. Im Rahmen von Abschnitt 4.2.1 wird zudem exemplarisch präsentiert, wie sie als effektives Mittel zur Analyse der Fahrzeugumgebung in einer automatisierten taktischen Entscheidungsfindung eingesetzt werden können. Anders als Freibereichspolygone bieten Voronoi-Pfade jedoch keinen direkten Vorteil in Bezug auf die Gesamtrechenzeit, weshalb dieser Abschnitt einen Schwerpunkt auf die Betrachtung des benötigten Aufwands für deren Berechnung legt. Als Vergleichswert wird dabei die anvisierte Updatezeit von 100 ms für die Taktikautonomie in OPA³L verwendet (siehe Abschnitt 1.2). Abschnitt 3.3.5.1 stellt zunächst eine numerische Laufzeitanalyse vor, wobei insbesondere diskutiert wird, welche Auswirkung das Vorhandensein von beweglichen Objekten hat, beziehungsweise welche Optimierungsmöglichkeiten – zum Beispiel durch Parallelisierung – in diesem Fall zur Verfügung stehen. Die Ergebnisse werden das Verwenden eines *lokalen Voronoi-Updates* motivieren. Dies bezeichnet einen Algorithmus, welcher im Rahmen dieser Arbeit entwickelt wurde, um dynamische Voronoi-Pfade auch für hochgradig dynamische Szenarien und bei geringer Parallelisierungskapazität effizient berechnen zu können. Eine detaillierte Präsentation des Verfahrens findet sich in Abschnitt 3.3.5.2. Die einzelnen Teilschritte werden dabei vorrangig geometrisch motiviert; ein formaler Korrektheitsbeweis liegt nicht im Umfang dieser Arbeit.

3.3.5.1 Laufzeitanalyse der Voronoi-Pfad-Generierung

Es folgt eine numerische Laufzeitanalyse bezüglich der Erstellung von (dynamischen) Voronoi-Pfaden, wobei sowohl die Abhängigkeit von der Größe des Freibereichspolygons als auch von der vorliegenden Dynamik anhand der Situation aus Abbildung 3.26 untersucht wird. Die Variation im Freibereichspolygon wird dabei durch verschiedene Schwellenwerte in der Polygonvereinfachung `Simplify` aus Algorithmus 3.1 in Abschnitt 3.2.2 erreicht, was in direktem Zusammenhang mit der Menge der enthaltenen Polygonsegmente steht. Um die Erstellung von dynamischen Voronoi-Pfaden zu analysieren, wird ein bewegliches Objekt anhand der gezeigten Trajektorie mit unterschiedlich vielen Diskretisierungspunkten berücksichtigt.

Für ein rein statisches Szenario – ohne Beachtung des beweglichen Objektes – sind die Rechenzeiten abhängig von der Anzahl der Segmente im Freibereichspolygon in Abbildung 3.27 dargestellt. Die Ergebnisse zeigen einen annähernd linearen Verlauf; im direkten Vergleich mit der auf den Messdaten basierenden linearen Regression ist allerdings klar eine leichte Krümmung zu erkennen. Diese steht im Einklang mit der theoretischen Zeitkomplexität aus (3.14) für die Erstellung eines Voronoi-Pfades, vergleiche Abschnitt 3.3.2. Insgesamt benötigen die kleineren der diskutierten Freibereichspolygone mit unter 150 Segmenten moderate Rechenzeiten von 2 ms bis

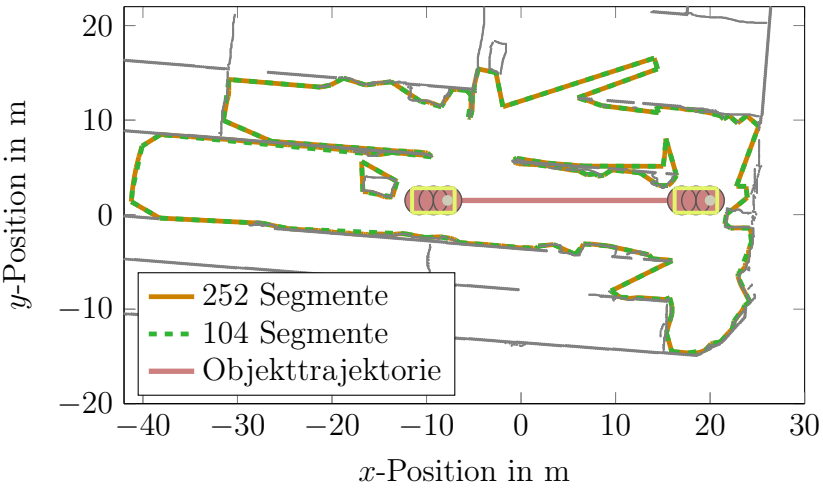


Abbildung 3.26: Exemplarische Szenarien für die Auswertung der Rechenzeit zur Erstellung des Voronoi-Pfades in statischen und dynamischen Szenarien und für die verschiedene Auflösungen des Freibereichspolygons aus Abbildung 3.11. Gezeigt wird das Freibereichspolygon mit maximaler und minimaler betrachteter Auflösung sowie die Randzustände der Trajektorie des optional berücksichtigten beweglichen Objektes.

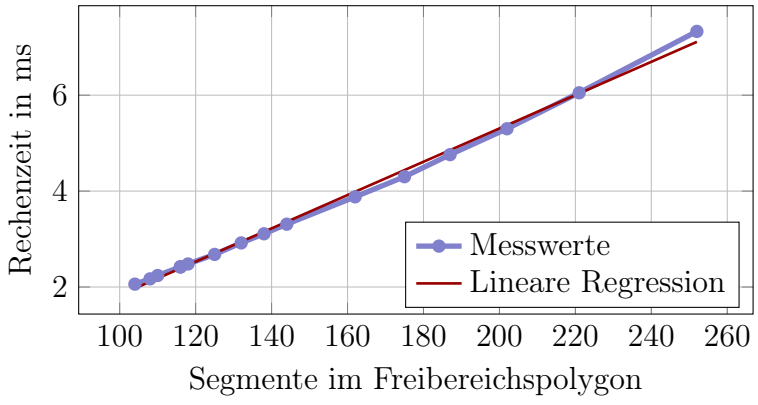
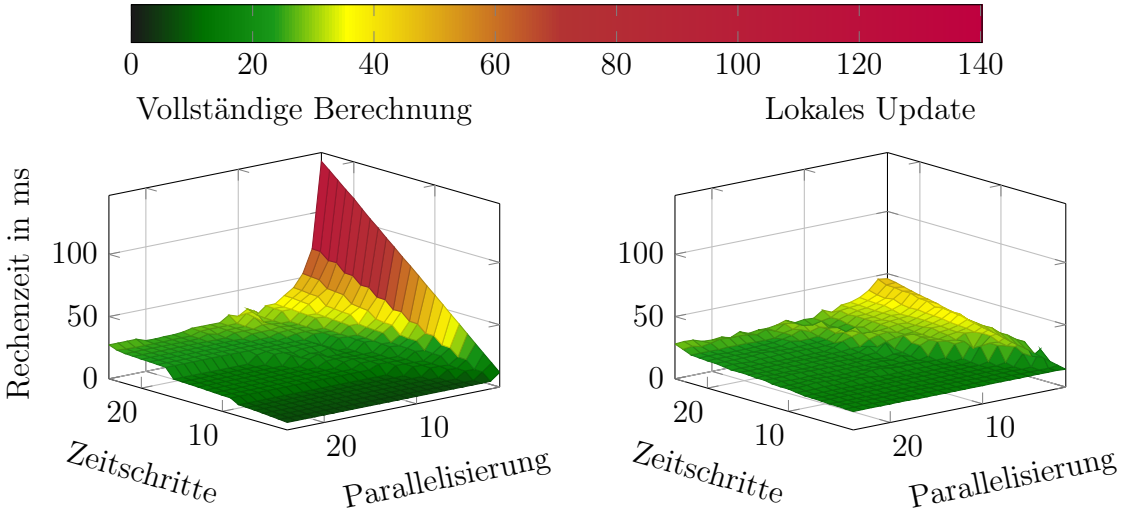


Abbildung 3.27: Rechenzeiten zur Berechnung des statischen Voronoi-Pfades für verschiedene Auflösungen des Freibereichspolygons aus Abbildung 3.26

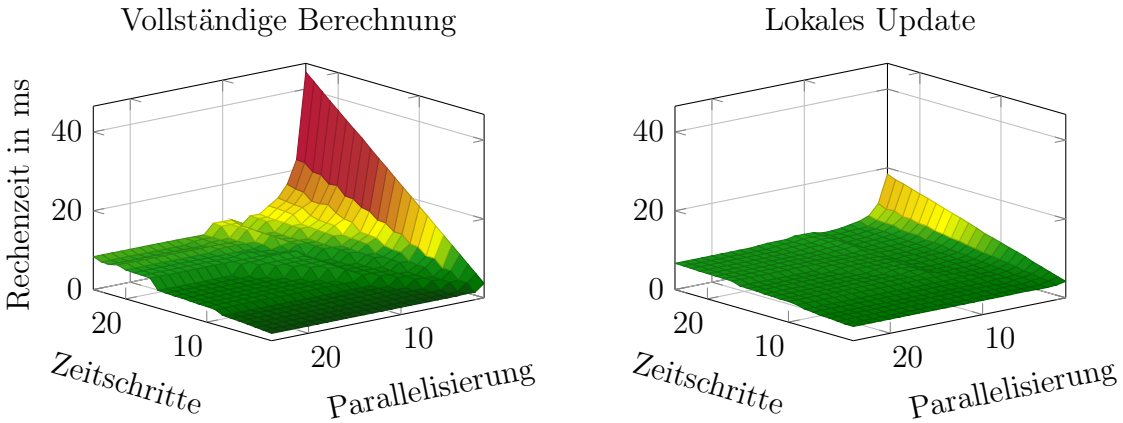
4 ms. Demgegenüber stehen die Polygone mit ca. 250 Segmenten, welche zu einem vergleichsweise hohen Aufwand von fast 8 ms führen. Je nach verfügbarer Rechenkapazität erlaubt die Polygonvereinfachung mit `Simplify` in diesem Kontext eine einfache Abwägung zwischen Approximationsgenauigkeit des Freibereichspolygons und dem Zeitaufwand für dessen Auswertung – beispielsweise zur Erstellung des Voronoi-Pfades. Um sowohl eine hohe Darstellungsgenauigkeit des Polygons als auch eine effiziente Voronoi-Pfad-Berechnung zu erreichen, kann alternativ ein hybrider Ansatz gewählt werden, bei dem eine stärker vereinfachte Variante des Freibereichspolygons nur zur Erstellung des Voronoi-Pfades generiert wird. Der Nachteil dieses Ansatzes ist jedoch, dass das detaillierte Freibereichspolygon und der vereinfachte Voronoi-Pfad dann nicht mehr konsistent sind.

Ein geringer Aufwand für die Generierung eines einzelnen Voronoi-Pfades ist vor allem in dynamischen Szenarien von Bedeutung. Wie in Abschnitt 3.3.3 dargestellt, wird die Trajektorie der beweglichen Objekte für diesen Fall in der Zeit diskretisiert und die notwendigen Berechnungen für jeden Zeitschritt ausgeführt. Da die entstehenden Aufgaben voneinander unabhängig sind, ist eine parallele Ausführung eine einfache Möglichkeit zur Effizienzsteigerung. Die daraus resultierenden Rechenzeiten für 2 bis 24 Diskretisierungsschritte sind in den Abbildungen 3.28a und 3.28b für zwei verschieden dichte Freibereichspolygone dargestellt, jeweils auf der linken Seite als „Vollständige Berechnung“ betitelt. Die Menge der Polygonsegmente beeinflusst in diesen Auswertungen im Wesentlichen die Höhe der gezeigten Kurven und führt ansonsten zu den qualitativ ähnlichen Ergebnissen. Für beide ist der Effekt der Parallelisierung daher gleichermaßen klar zu erkennen: Werden bis zu 8 Diskretisierungsschritte mit mindestens genauso vielen Prozessen verarbeitet, entsteht im Vergleich zur Berechnung des entsprechenden statischen Voronoi-Pfades kein wesentlicher zusätzlicher Aufwand. Dieser Effekt ist jedoch auf die verfügbaren 8 physischen Kerne des verwendeten AMD Ryzen 7 PRO 4750U CPU Prozessors limitiert. Eine feinere Zeitdiskretisierung kann folglich nicht mehr ohne leicht erhöhte Rechenzeit verarbeitet werden, weshalb sich selbst für 24 Prozesse kleine aber deutliche Stufen ab 9 und 17 Zeitschritten ergeben. Besonders bei einer geringen Parallelisierung führt eine feine Diskretisierung jedoch schnell zu einem vergleichsweise hohen Aufwand. Wird gar nicht parallelisiert, so steigt die benötigte Zeit für die feinste Zeitdiskretisierung in diesem Beispiel auf 140 ms beziehungsweise 45 ms an. Damit läge sie selbst für das vereinfachte Freibereichspolygon bereits bei ca. der Hälfte der verfügbaren Gesamtzeit von 100 ms.

Der Anstieg der Rechenzeit abhängig von der Anzahl der Zeitschritte bei strikt sequentieller Berechnung erfolgt in beiden Beispielen sehr linear. Dies ergibt sich direkt aus dem in Abschnitt 3.3.3 vorgestellten Ansatz zur Berechnung dynamischer Voronoi-Pfade, bei dem jeweils das Voronoi-Diagramm für sämtliche Eingabedaten aktualisiert wird und somit eine Reihe von ungefähr gleichwertigen Aufgaben entsteht. Tatsächlich unterscheiden sich die dynamischen Probleme häufig aber nur marginal voneinander. Dies ist bereits gut anhand des Beispiels aus Abbildung 3.23 er-



(a) Auswertung für das Freibereichspolygon mit 252 Segmenten



(b) Auswertung für das Freibereichspolygon mit 104 Segmenten

Abbildung 3.28: Vergleich der benötigten Zeiten zur Berechnung der dynamischen Voronoi-Pfade für die Situation aus Abbildung 3.26, abhängig von der Zeitdiskretisierung der Trajektorie des beweglichen Objektes und der Anzahl an verfügbaren parallelen Rechenprozessen. Die *vollständige Berechnung* bestimmt jeweils das Voronoi-Diagramm in jedem Zeitschritt unter Berücksichtigung aller Eingabedaten komplett neu, wie in Abschnitt 3.3.3 vorgestellt. Die Berechnung mit dem *lokalen Update* basiert auf einer Aktualisierung der durch die zusätzlichen Eingabedaten des dynamischen Objektes betroffenen Kanten des statischen Voronoi-Diagramms anhand von Algorithmus 3.2.

kennbar, bei dem die Voronoi-Pfade nur in der unmittelbaren Nähe des beweglichen Objektes untereinander abweichen. Dies folgt direkt aus der Konstruktionsvorschrift eines Voronoi-Diagramms: Wird der statische Pfad als Ausgangspunkt betrachtet, dann werden genau diejenigen Voronoi-Kanten invalide, die näher an den zusätzlichen Eingabedaten der beweglichen Objekte liegen, als an den statischen Segmenten des Freibereichspolygons. Diese Idee kann für eine effizientere Berechnungsstrategie ausgenutzt werden, bei dem zunächst der statische Voronoi-Pfad bestimmt wird und die dynamischen Kanten anschließend lediglich an den notwendigen Stellen aktualisiert werden. Die Rechenzeiten für dieses *lokale Voronoi-Update* sind auf der jeweils rechten Seite von Abbildung 3.28 dargestellt. In beiden Fällen ergibt sich im Mittel eine deutliche Reduzierung des Aufwands; dies gilt insbesondere im Fall von wenigen parallelen Prozessen. Beispielsweise liegen die jeweils höchsten Rechenzeiten bei sequentieller Berechnung bei 46 ms beziehungsweise 18 ms. Insgesamt erlaubt dieser Ansatz damit eine sinnvolle Abwägung zwischen gewünschter Zeitdiskretisierung und Polygongenauigkeit.

Der nächste Abschnitt 3.3.5.2 liefert detaillierte Berechnungsvorschriften für die Umsetzung des lokalen Voronoi-Updates.

3.3.5.2 Das lokale Voronoi-Update

Für eine gegebene Menge regulärer Voronoi-Kanten $\mathcal{R}^\diamond(\Upsilon)$ mit zugehörigen Eingabedaten Υ sind die notwendigen Schritte zur Durchführung eines lokalen Updates in Algorithmus 3.2 zusammengefasst und in Abbildung 3.29 exemplarisch visualisiert. Dabei wird der Filtervorgang gemäß Definition 3.18 aus Abschnitt 3.3.2 zur abschließenden Generierung eines Voronoi-Pfades zugunsten einer vereinfachten Darstellung ohne Einschränkung der Korrektheit vernachlässigt – dieser zusätzliche Schritt lässt sich am Ende des Verfahrens ergänzen.
Wie bereits motiviert, müssen im Rahmen dieser Methode zunächst aus den gegebenen Voronoi-Kanten die *betroffenen*, invaliden Elemente S_E identifiziert werden. Diese kennzeichnen sich dadurch, dass es einen Punkt auf der Voronoi-Kante gibt, welcher näher an einem Segment der zusätzlichen Eingabedaten $\overline{\Upsilon}$ liegt, als an allen Ursprünglichen. Das Überprüfen dieser Bedingung ist durch eine einfache Analyse der distanzbezogenen Lagebeziehung zwischen je zwei Liniensegmenten gemäß Abschnitt 3.2.1.2 möglich.

Liegt der Schnittpunkt zwischen einem zusätzlichen Eingabesegment und einer Voronoi-Kante im Inneren letzterer, so ist diese betroffen und muss durch Algorithmus 3.2 aktualisiert werden. Liegt der Schnittpunkt am Rand der Kante gilt dies nur, wenn keines der ursprünglichen Segmente hier ebenfalls schneidet. Liegt kein Schnittpunkt vor, so können die vier Vergleichswerte zwischen einer Voronoi-Kante und einem zusätzlichen Eingabesegment gemäß (3.7) berechnet werden. Dabei entstehen entsprechend bis zu vier Vergleichspunkte auf der Kante. Diese ist

Algorithmus 3.2 : Lokales Update der regulären Kanten eines bereits berechneten Voronoi-Diagramms für zusätzliche Segmentdaten

Eingabe : Eingabedaten $\Upsilon \in \Xi$, zugehörige reguläre Voronoi-Kanten $\mathcal{R}^\diamond(\Upsilon) \in \Pi(\mathcal{S})$, zusätzliche Eingabedaten $\overline{\Upsilon} \in \Xi$ sodass $\Upsilon \cup \overline{\Upsilon} \in \Xi$

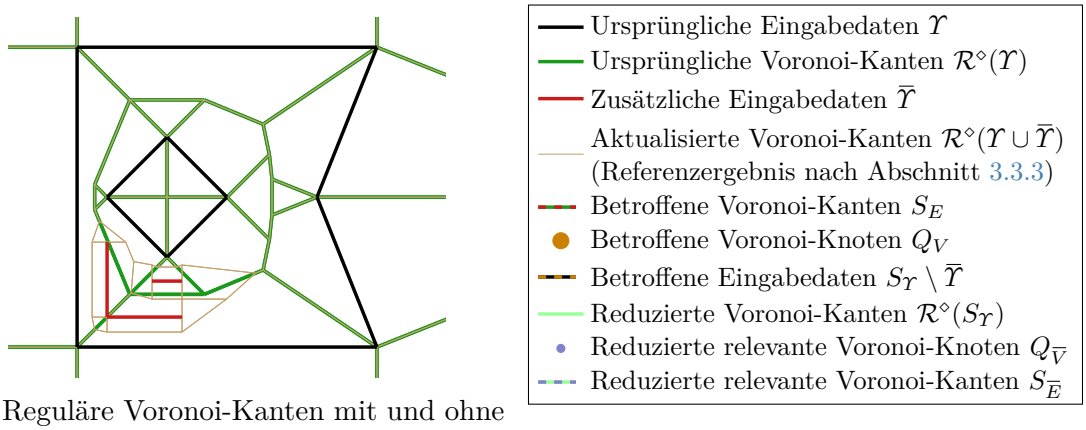
Ausgabe : Aktualisierte reguläre Voronoi-Kanten $\mathcal{R}^\diamond(\Upsilon \cup \overline{\Upsilon})$

```
// Indentifikation der relevanten Eingabesegmente
1 S_E ← GetAffectedEdges(R°(Υ), Υ, Ῡ);
2 Q_V ← GetAffectedVertices(S_E);
3 S_Υ ← GetRelevantSites(Q_V, Υ);

// Berechnung und Verarbeitung der regulären Kanten des reduzierten
   Voronoi-Diagramms
4 Q_Ῡ ← GetRelevantVertices(R°(S_Υ), Υ, Ῡ);
5 S_Ē ← GetRelevantEdges(R°(S_Υ), Q_Ῡ);

// Zusammenfügen der Kanten beider Diagramme
6 return Merge(R°(Υ), S_E, S_Ē);
```

genau dann als *betroffen* zu klassifizieren, wenn der Vergleich einer dieser Punkte mit allen ursprünglichen Eingabedaten in strikt größeren Distanzen resultiert. Eine detaillierte Beschreibung dieses Vorgehens ist in der Funktion `GetAffectedEdges` dargestellt. In Abbildung 3.29b ist darüber hinaus die Identifizierung betroffener Voronoi-Kanten für exemplarische zusätzliche Eingabedaten visualisiert.

Die Aktualisierung der betroffenen Voronoi-Kanten ergibt sich aus dem um die zusätzlichen Eingabedaten erweiterten Voronoi-Diagramm. Wie bereits motiviert genügt es hierfür jedoch, nur diejenigen ursprünglichen Eingabesegmente zu berücksichtigen, welche originär zu den nun betroffenen Kanten S_E geführt haben. Diese betroffenen Segmente entsprechen per Konstruktion eines Voronoi-Diagramms genau denjenigen, die zu einem Randpunkt einer betrachteten Kante die kürzeste Distanz haben. Zusammen mit den zusätzlichen Eingabedaten ergeben sich daraus die für die Aktualisierung *relevanten* Segmente S_Υ. Deren Berechnung auf Grundlage der betroffenen Knoten Q_V ist in den Funktionen `GetAffectedVertices` sowie `GetRelevantSites` beschrieben und ebenfalls in Abbildung 3.29b beispielhaft dargestellt.

Es resultieren schließlich die reduzierten regulären Voronoi-Kanten $\mathcal{R}^\diamond(S_\Upsilon)$ wie in Abbildung 3.29c gezeigt. Für genau diesen Schritt zeigt sich sich die Stärke von Algorithmus 3.2, da der begrenzte Einflussbereich der neuen Daten ausgenutzt wird: Ist das Verhältnis zwischen der Anzahl an nicht betroffenen und betroffenen Eingabesegmenten groß, so folgt daraus ein deutlicher Geschwindigkeitsvorteil bei der ansonsten vergleichsweise aufwändigen Berechnung der entsprechenden Voronoi-Kanten.



(a) Reguläre Voronoi-Kanten mit und ohne Berücksichtigung der zusätzlichen Eingabedaten (zur Referenz)

(b) Betroffene Voronoi-Kanten und zugehörige Knoten und Eingabedaten

(c) Reguläre Voronoi-Kanten für reduzierte Eingabedaten

(d) Relevante Voronoi-Knoten und zugehörige reguläre Kanten des reduzierten Voronoi-Diagramms

(e) Vereinigung der relevanten Kanten des reduzierten Diagramms mit den nicht-betroffenen ursprünglichen Kanten

Abbildung 3.29: Visualisierung der einzelnen Schritte des Algorithmus 3.2 zur lokalen Aktualisierung der regulären Kanten eines Voronoi-Diagramms für zusätzliche Eingabedaten; basierend auf dem Beispiel aus Abbildung 3.21.

Funktion `GetAffectedEdges`$(\mathcal{R}^\diamond, \Upsilon, \overline{\Upsilon})$

Initialisiere leere Menge von Segmenten $S_E \leftarrow \emptyset \subset \mathcal{S}$;
`// Teste alle regulären Voronoi-Kanten`
for $s_\mathcal{R} \leftarrow ({s_\mathcal{R}}^{\langle 1\rangle}, {s_\mathcal{R}}^{\langle 2\rangle})_\mathcal{S} \in \mathcal{R}^\diamond$ **do**
  if `CheckForIntersection`$(({s_\mathcal{R}}^{\langle 1\rangle}, {s_\mathcal{R}}^{\langle 2\rangle})_\mathcal{S}, \overline{\Upsilon})$
    or $\min_{s_{\overline{\Upsilon}} \in \overline{\Upsilon}} \mathrm{d}(s_\mathcal{R}{}^{\langle 1\rangle}, s_{\overline{\Upsilon}}) < \min_{s_\Upsilon \in \Upsilon} \mathrm{d}(s_\mathcal{R}{}^{\langle 1\rangle}, s_\Upsilon)$
    or $\min_{s_{\overline{\Upsilon}} \in \overline{\Upsilon}} \mathrm{d}(s_\mathcal{R}{}^{\langle 2\rangle}, s_{\overline{\Upsilon}}) < \min_{s_\Upsilon \in \Upsilon} \mathrm{d}(s_\mathcal{R}{}^{\langle 2\rangle}, s_\Upsilon)$
    or `CheckIfInteriorOfEdgeIsAffected`$(s_\mathcal{R}, \Upsilon, \overline{\Upsilon})$ **then**
    $S_E \leftarrow S_E \cup \{s_\mathcal{R}\}$;
  end
end
return S_E;

Funktion `CheckForIntersection`$(({s_\mathcal{R}}^{\langle 1\rangle}, {s_\mathcal{R}}^{\langle 2\rangle})_\mathcal{S}, \overline{\Upsilon})$

for $({s_{\overline{\Upsilon}}}^{\langle 1\rangle}, {s_{\overline{\Upsilon}}}^{\langle 2\rangle})_\mathcal{S} \in \overline{\Upsilon}$ **do**
  `// Stelle sicher, dass Segmente nicht kollinear sind`
  if $({s_\mathcal{R}}^{\langle 2\rangle} - {s_\mathcal{R}}^{\langle 1\rangle}) \times_2 ({s_{\overline{\Upsilon}}}^{\langle 2\rangle} - {s_{\overline{\Upsilon}}}^{\langle 1\rangle}) \neq 0$ **then**
    `// Berechnung des Schnittpunktes gemäß (3.5) und (3.6)`
    $\tau_1 \leftarrow \left(({s_{\overline{\Upsilon}}}^{\langle 1\rangle} - {s_\mathcal{R}}^{\langle 1\rangle}) \times_2 ({s_{\overline{\Upsilon}}}^{\langle 2\rangle} - {s_{\overline{\Upsilon}}}^{\langle 1\rangle})\right) / \left(({s_\mathcal{R}}^{\langle 2\rangle} - {s_\mathcal{R}}^{\langle 1\rangle}) \times_2 ({s_{\overline{\Upsilon}}}^{\langle 2\rangle} - {s_{\overline{\Upsilon}}}^{\langle 1\rangle})\right)$;
    $\tau_2 \leftarrow \left(({s_\mathcal{R}}^{\langle 1\rangle} - {s_{\overline{\Upsilon}}}^{\langle 1\rangle}) \times_2 ({s_\mathcal{R}}^{\langle 2\rangle} - {s_\mathcal{R}}^{\langle 1\rangle})\right) / \left(({s_{\overline{\Upsilon}}}^{\langle 2\rangle} - {s_{\overline{\Upsilon}}}^{\langle 1\rangle}) \times_2 ({s_\mathcal{R}}^{\langle 2\rangle} - {s_\mathcal{R}}^{\langle 1\rangle})\right)$;
    `// Prüfe, ob Schnitt strikt im Inneren der Voronoi-Kante vorliegt (`τ_1`). Für das Eingabesegment (`τ_2`) kann auch der Rand betroffen sein.`
    if $0 < \tau_1 < 1$ **and** $0 \leq \tau_2 \leq 1$ **then**
      return true;
    end
  end
end
return false;

Funktion `CheckIfInteriorOfEdgeIsAffected`$(s_\mathcal{R}, \Upsilon, \overline{\Upsilon})$

for $({s_{\overline{\Upsilon}}}^{\langle 1\rangle}, {s_{\overline{\Upsilon}}}^{\langle 2\rangle})_\mathcal{S} \in \overline{\Upsilon}$ **do**
  `// Eindeutige Minimierer existieren`
  if $\mathrm{d}({s_{\overline{\Upsilon}}}^{\langle 1\rangle}, s_\mathcal{R}) < \min_{s_\Upsilon \in \Upsilon} \mathrm{d}\left(\arg\min_{p \in s_\mathcal{R}} \mathrm{d}({s_{\overline{\Upsilon}}}^{\langle 1\rangle}, p), s_\Upsilon\right)$
    or $\mathrm{d}({s_{\overline{\Upsilon}}}^{\langle 2\rangle}, s_\mathcal{R}) < \min_{s_\Upsilon \in \Upsilon} \mathrm{d}\left(\arg\min_{p \in s_\mathcal{R}} \mathrm{d}({s_{\overline{\Upsilon}}}^{\langle 2\rangle}, p), s_\Upsilon\right)$ **then**
    return true;
  end
end
return false;

Funktion `GetAffectedVertices`(S_E)
  Initialisiere leere Menge von Punkten $Q_V \leftarrow \emptyset \subset \mathbb{R}^2$;
  for $(s_{\mathcal{R}}{}^{\langle 1\rangle}, s_{\mathcal{R}}{}^{\langle 2\rangle})_{\mathcal{S}} \in S_E$ **do**
    // Betroffene Voronoi-Knoten entsprechen Endpunkten der betroffenen Voronoi-Kanten
    $Q_V \leftarrow Q_V \cup \{s_{\mathcal{R}}{}^{\langle 1\rangle}, s_{\mathcal{R}}{}^{\langle 2\rangle}\}$;
  end
  return Q_V;

Funktion `GetRelevantSites`$(Q_{\overline{V}}, \Upsilon, \overline{\Upsilon})$
  // Alle zusätzlichen Eingabedaten sind per Konstruktion relevant
  Initialisiere Segmente $S_\Upsilon \leftarrow \overline{\Upsilon} \subset \mathcal{S}$;
  for $q_{\overline{V}} \in Q_{\overline{V}}$ **do**
    for $s_\Upsilon \in \Upsilon$ **do**
      // Die nahesten Segmente zu den betroffenen Voronoi-Knoten entsprechen den betroffenen ursprünglichen Eingabedaten
      if $\mathrm{d}(q_{\overline{V}}, s_\Upsilon) = \min_{\tilde{s}_\Upsilon \in \Upsilon} \mathrm{d}(q_{\overline{V}}, \tilde{s}_\Upsilon)$ **then**
        $S_\Upsilon \leftarrow S_\Upsilon \cup \{s_\Upsilon\}$;
      end
    end
  end
  return S_Υ;

Diese Voraussetzung gilt häufig in realitätsnahen Anwendungsbeispielen, wie sich zum Beispiel anhand der Ergebnisse zur Rechenzeitauswertung für das dynamische Szenarios aus Abbildung 3.26 andeutet.

Aus den regulären Kanten des reduzierten Diagramms müssen nun die relevanten Voronoi-Kanten herausgefiltert werden, welche die Aktualisierung der betroffenen ursprünglichen Kanten darstellen. Wie zuvor können hierzu wieder die Voronoi-Knoten als Bindeglied zwischen Kanten und Eingabesegmenten analysiert werden: Eine Kante ist genau dann relevant, wenn die kürzeste Distanz mindestens einer ihrer Knoten zu den Eingabedaten S_Υ einer Kante aus den zusätzlichen Segmenten $\overline{\Upsilon}$ zuzuordnen ist. Die entsprechende Berechnung der relevanten Knoten $Q_{\overline{V}}$ und Kanten $S_{\overline{E}}$ ist in den Funktionen `GetRelevantVertices` und `GetRelevantEdges` zusammengefasst und in Abbildung 3.29d visualisiert.

Die regulären Kanten des aktualisierten Voronoi-Diagramms $\mathcal{R}^\diamond(\Upsilon \cup \overline{\Upsilon})$ ergeben sich nach den vorigen Schritten abschließend durch das Zusammenfügen der nicht betroffenen ursprünglichen Kanten $\mathcal{R}^\diamond(\Upsilon) \setminus S_E$ mit den zuletzt berechneten relevanten Kanten des reduzierten Diagramms $S_{\overline{E}}$ in der Funktion `Merge`. Das Ergebnis für das zuvor betrachtete Beispiel ist in Abbildung 3.29e dargestellt. Zum Vergleich ist

Funktion `GetRelevantVertices`($\mathcal{R}^\diamond$, Υ, $\overline{\Upsilon}$)
  Initialisiere leere Menge von Punkten $Q_{\overline{V}} \leftarrow \emptyset \subset \mathbb{R}^2$;
  `// Überprüfe beide Knoten`
  for $(s^{\langle 1\rangle}, s^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{R}^\diamond$ **do**
    if `VertexIsRelevant`($s^{\langle 1\rangle}$, Υ, $\overline{\Upsilon}$) **then**
      $Q_{\overline{V}} \leftarrow Q_{\overline{V}} \cup \{s^{\langle 1\rangle}\}$;
    end
    if `VertexIsRelevant`($s^{\langle 2\rangle}$, Υ, $\overline{\Upsilon}$) **then**
      $Q_{\overline{V}} \leftarrow Q_{\overline{V}} \cup \{s^{\langle 2\rangle}\}$;
    end
  end
  return $Q_{\overline{V}}$;

Funktion `VertexIsRelevant`(q, Υ, $\overline{\Upsilon}$)
  `// Überprüfe, ob Knoten mindestens ein Eingabesegment als Nahestes Element hat`
  return $\min_{s_{\overline{\Upsilon}} \in \overline{\Upsilon}} \mathrm{d}(q, s_{\overline{\Upsilon}}) \leq \min_{s_\Upsilon \in \Upsilon} \mathrm{d}(q, s_\Upsilon)$;

Funktion `GetRelevantEdges`($\mathcal{R}^\diamond$, $Q_{\overline{V}}$)
  Initialisiere leere Menge von Segmenten $S_{\overline{E}} \leftarrow \emptyset \subset \mathcal{S}$;
  for $s_\mathcal{R} \leftarrow ({s_\mathcal{R}}^{\langle 1\rangle}, {s_\mathcal{R}}^{\langle 2\rangle})_{\mathcal{S}} \in \mathcal{R}^\diamond$ **do**
    `// Voronoi-Kanten von relevanten Knoten sind relevant`
    if ${s_\mathcal{R}}^{\langle 1\rangle} \in Q_{\overline{V}}$ **or** ${s_\mathcal{R}}^{\langle 2\rangle} \in Q_{\overline{V}}$ **then**
      $S_{\overline{E}} \leftarrow S_{\overline{E}} \cup \{s_\mathcal{R}\}$;
    end
  end
  return $S_{\overline{E}}$;

dort auch das (identische) Referenzergebnis mit eingezeichnet, welches sich durch die direkte Berechnung des Voronoi-Diagramms gemäß Abschnitt 3.3.3 ergibt.

Im direkten Vergleich kann das im Vorigen vorgestellte Verfahren zum lokalen Voronoi-Update eine deutliche Effizienzsteigerung bewirken. Wie Abbildung 3.28 zeigt, gilt dies umso stärker, je feiner der zeitliche Verlauf von Objekttrajektorien diskretisiert wird, beziehungsweise je weniger Ressourcen zur parallelen Erstellung der dynamischen Voronoi-Pfade verfügbar sind. In diesen Fällen nutzt Algorithmus 3.2 den lokal beschränkten Wirkungsbereich zusätzlicher Eingabesegmente aus, sodass nur Voronoi-Diagramme in minimaler Komplexität berechnet werden müssen. Potentiell aufwändig wird dieser Ansatz jedoch dann, wenn vergleichsweise viele Voronoi-Kanten durch die zusätzlichen Daten betroffen sind – zum Beispiel weil sehr viele bewegliche Objekte betrachtet werden. Durch die zusätzlichen Analyseschritte des Verfahrens wird der Gesamtaufwand dann den der originären Methode

Funktion `Merge`$(\mathcal{R}^\diamond(\Upsilon), S_E, S_{\overline{E}})$
  return $\{\mathcal{R}^\diamond(\Upsilon) \setminus S_E\} \cup \{S_{\overline{E}}\}$;

aus Abschnitt 3.3.3 übersteigen. Ein weiterer Nachteil ergibt sich bei einem großen Parallelisierungspotential: Liegt ein rein dynamisches Szenario vor, so muss für das lokale Voronoi-Update trotzdem zuvor der Pfad für das statische Pendant berechnet werden. Wenn es mehr parallele Prozesse als Diskretisierungsschritte gibt, so führt dies in der Folge zu einem Geschwindigkeitsnachteil gegenüber dem ursprünglichen Ansatz; im diskutierten Beispiel aus Abbildung 3.26 entspricht dieser ungefähr der Zeit für die statische Berechnung aus Abbildung 3.27. Beide Kritikpunkte können im Kontext eines Systems für autonomes Fahren jedoch relativiert werden: Aufgrund von Sicherheitsabständen sind bewegliche Objekte selten auf engem Raum verdichtet. Zudem sind die Rechenkapazität auf einer solchen mobilen Plattform in der Regel beschränkt und werden von einer Vielzahl von Modulen in Anspruch genommen, um die vielen komplexen Probleme zu lösen, vergleiche beispielsweise das OPA^{3}L-System aus Abschnitt 1.2.

Zu zeigen bleibt die Korrektheit von Algorithmus 3.2, was hier nur geometrisch motiviert und an einem exemplarischen Beispiel präsentiert wurde. Als wesentlicher Aspekt ist dabei zu beachten, dass das Update bereits mit der Linearisierung der Voronoi-Kanten arbeitet, sodass Teile der Eigenschaften der Kanten des ursprünglichen Voronoi-Diagramms verloren gehen. Als Kernargument für die Korrektheit ist dabei die Linearität sowie die Überschneidungsfreiheit der Eingabesegmente zu nennen. Dies hat zur Folge, dass die Linearisierung der Voronoi-Kanten keinen qualitativen Unterschied in Bezug auf die vorliegende Geometrie bewirkt, sodass es zu keinen Änderungen in den Minimierern der vorstehenden Teilfunktionen kommt. Eine rigorose Beweisführung basierend auf diesem Argument wird nicht im Umfang dieser Dissertationsschrift durchgeführt, sondern für nachfolgende Arbeiten offen gelassen.

Ganz allgemein können die in diesem Abschnitt eingeführten Voronoi-Pfade als eine sichere Referenzbahn in Bezug auf die unmittelbare Umgebung (zum Beispiel eines autonomen Fahrzeuges) betrachtet werden. Dies kann neben der Definition eines Voronoi-Potentials für den Planungsalgorithmus auch als Analyseinstrument im Rahmen einer Taktikautonomie genutzt werden. Abschnitt 3.3.6 zeigt in diesem Kontext exemplarisch, wie basierend auf dem statischen Voronoi-Pfad automatisch valide Zielpunkte innerhalb des Freibereichspolygons generiert werden können. Diese Art der Anwendung stellt schließlich eine weitere Rechtfertigung für den zunächst zusätzlichen Aufwand der Berechnung des statischen Pfades im lokalen Voronoi-Update dar.

3.3.6 Automatische Zielpfadgenerierung

Ein verbleibendes Detail bei der Bewegungsplanung mit dem Hybrid A*-Algorithmus 2.3 betrifft die Definition der Start- und Zielzustände $z^{(\text{start})}$ sowie $z^{(\text{end})}$. Ersterer ergibt sich dabei kanonisch aus dem aktuell geschätzten Zustand des Fahrzeuges in der Welt. Dies gilt jedoch explizit nicht für die Zielkonfiguration, weshalb sie beispielsweise durch eine externe Wissensquelle – dies kann ein übergeordneter Algorithmus oder etwa ein menschlicher Nutzer sein – vorgegeben werden muss. Als Alternative dazu wird im Folgenden ein Verfahren vorgestellt, durch welches ein grober Zielpfad automatisiert aus dem Wissen über die Fahrzeugumgebung in Form eines Freibereichspolygons berechnet werden kann. Dabei werden sowohl aktuell betrachtete Explorationsstrategien (zum Beispiel *Geradeausfahren* oder *nach links fahren*) als auch Beschränkungen (zum Beispiel *plane nicht weiter als eine vorgegebene Entfernung*) berücksichtigt. Aus dem Zielpfad kann dann Wissen über die Umgebung aber auch eine Zielkonfiguration für Planungsalgorithmen extrahiert werden. Dies ermöglicht schließlich die konsekutive Berechnung von immer neuen Trajektorien, beispielsweise im Rahmen eines taktischen Planers, für ein sich durch die Welt bewegendes Fahrzeug, ohne das spezielle Annahmen an die Umwelt getroffen werden müssen. Das Zusammenspiel all dieser Komponenten wird exemplarisch in Abschnitt 4.2 vorgestellt.

Die im weiteren Verlauf vorgestellte automatisierte Zielpfadberechnung basiert im Kern auf der Eigenschaft des Voronoi-Pfades, zentral im aktuell befahrbaren Bereich zu verlaufen. Damit eignet sich die Menge aller enthaltenen Voronoi-Knoten grundsätzlich als Quelle für sinnvolle Endkonfigurationen. Die Auswahl des am besten geeigneten Elementes erfolgt durch die Minimierung einer Kostenfunktion

$$f_T \colon \mathbb{R}^2 \to \mathbb{R}.$$

Durch diese lassen sich unterschiedlichste semantische Explorationsstrategien umsetzen, wie die folgende Auflistung exemplarisch zeigt.

- Geradeausfahren:

$$f_T^{\mathrm{g}}(p) \coloneqq -p_x.$$

- Fahren nach links:

$$f_T^{\mathrm{l}}(p) \coloneqq -p_y.$$

- Fahren in die Nähe eines Explorationspunktes $p_\diamond$:

$$f_T^{p_\diamond}(p) \coloneqq \mathrm{d}(p,\, p_\diamond).$$

- Möglichst weit vom aktuellen Referenzpunkt $p_{\text{Ego}} = p_{\text{Ego}}(\psi^{\circ}_{\text{Ego}})$ des Ego-Fahrzeug entfernen:

$$f_T^{\text{Ego}}(p) \coloneqq -\mathrm{d}(p,\, p_{\text{Ego}}).$$

Analoge Kostenfunktionen lassen sich für beliebige Richtungen definieren; insbesondere können auch verschiedene Ansätze für eine mehrkriterielle Vorgabe kombiniert werden. So beschreibt beispielsweise

$$f_T^{\text{gr}}(p) \coloneqq \frac{f_T^{\text{g}}(p) + f_T^{\text{r}}(p)}{2} = -\frac{p_x - p_y}{2}$$

das Ziel, gleichzeitig geradeaus und nach rechts zu fahren. Anhand einer vorliegenden Kostenfunktion kann grundsätzlich zunächst der *günstigste* Knoten $p_{\text{Ego},T}$ als Zielkandidat ausgewählt werden. Da es jedoch – je nach konkreter Situation – nicht passierbare Engstellen im vorliegenden Freibereichspolygon geben kann, ist für diesen a priori keine Erreichbarkeit gewährleistet. Unter der Annahme, dass entsprechende Engstellen im Filterprozess des Voronoi-Pfades aus Abschnitt 3.3.2 entfernt wurden, kann die Existenz eines Voronoi-Teilpfades ausgehend vom Fahrzeug zu $p_{\text{Ego},T}$ jedoch als hinreichendes Kriterium betrachtet werden. Um dies zu prüfen, wird die vorliegende Datenstruktur zunächst im Kontext von Kapitel 2 als Graph formalisiert.

Definition 3.24 (Voronoi-Graph)

Für einen gegebenen Voronoi-Pfad $\mathcal{R}^*$ wird der gewichtete Voronoi-Graph $G_{\mathcal{D}}^{\mathcal{R}^*} = (\mathcal{V}^{\mathcal{R}^*},\, E^{\mathcal{R}^*},\, \mathcal{D}^{\mathcal{R}^*})$ durch

$$\begin{aligned}
&\mathcal{V}^{\mathcal{R}^*} \coloneqq \{s^{\langle i\rangle} \in \mathbb{R}^2 \mid (s^{\langle i\rangle},\, \tilde{s})_{\mathcal{S}} \in \mathcal{R}^* \text{ oder } (\tilde{s},\, s^{\langle i\rangle})_{\mathcal{S}} \in \mathcal{R}^*\},\\
&E^{\mathcal{R}^*} \coloneqq \{e_{i,j} \coloneqq (s^{\langle i\rangle},\, s^{\langle j\rangle}) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid (s^{\langle i\rangle},\, s^{\langle j\rangle})_{\mathcal{S}} \in \mathcal{R}^* \text{ oder } (s^{\langle j\rangle},\, s^{\langle i\rangle})_{\mathcal{S}} \in \mathcal{R}^*\},\\
&\mathcal{D}^{\mathcal{R}^*} \colon E^{\mathcal{R}^*} \to \mathbb{R}_{\geq 0}, \quad e_{i,j} \mapsto \mathrm{d}(s^{\langle i\rangle},\, s^{\langle j\rangle})
\end{aligned}$$

definiert.

Dabei entsprechen die Knoten und Kanten des Voronoi-Pfades also denen des Graphen, wobei die Kanten stets beidseitig passierbar sind. Die Distanzfunktion ergibt sich aus der euklidischen Norm. Insgesamt kann damit eine effiziente Suche nach einem Voronoi-Teilpfad zum Beispiel durch den einfachen A*-Algorithmus aus Abschnitt 2.4 umgesetzt werden.

Algorithmus 3.3 fasst die beschriebenen Ideen zur Berechnung eines Voronoi-Teilpfades zusammen. Um einen Ausgangspunkt für das A*-Verfahren zu definieren, wird der räumlich naheste Knoten $p_{\text{Ego}}^{\mathcal{R}^*}$ zum aktuellen Referenzpunkt p_{Ego} des Ego-Fahrzeuges identifiziert, vergleiche Zeile 1. Anschließend werden die Knoten des Voronoi-Graphen der durch f_T vorgegebenen Priorisierung entsprechend auf die Existenz eines zulässigen Pfades hin untersucht. Wie in Zeile 4 beschrieben, wird

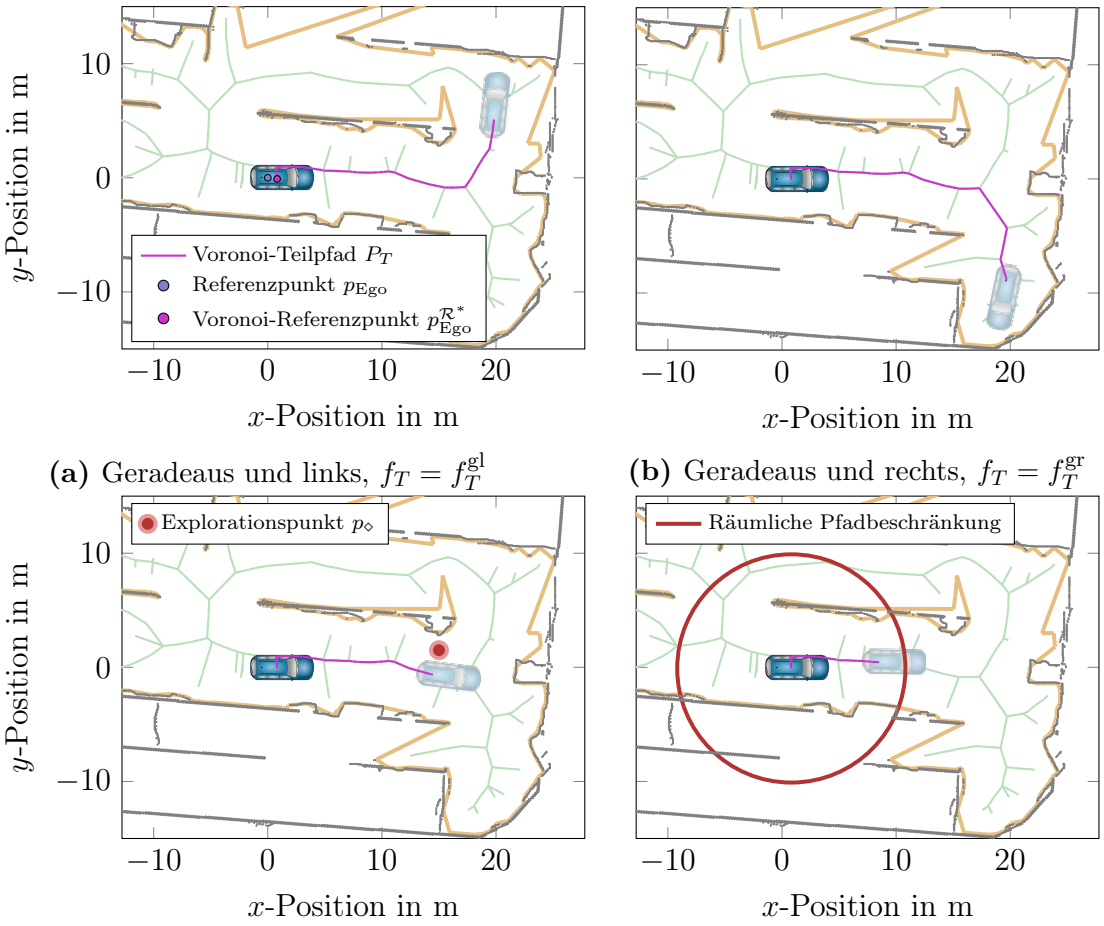
Algorithmus 3.3 : Automatisierte Berechnung eines Zielpfades entlang eines Voronoi-Pfades

Eingabe : Referenzpunkt des Ego-Fahrzeuges $p_{\text{Ego}} \in \mathbb{R}^2$, Voronoi-Pfad $\mathcal{R}^* \in \Pi(\mathcal{S})$, zugehöriges Freibereichspolygon $\gamma \in \Gamma$, Kostenfunktion $f_T \colon \mathbb{R}^2 \to \mathbb{R}$, Testfunktion für vorgegebene Pfadbeschränkung `PathValid`

Ausgabe : Zulässiger Voronoi-Teilpfad P_T, Leerer Pfad $\langle\rangle$ falls keine Lösung existiert

Initialisierung : Initialisiere gewichteten Voronoi-Graphen $G_{\mathcal{D}}^{\mathcal{R}^*} \leftarrow (\mathcal{V}^{\mathcal{R}^*}, E^{\mathcal{R}^*}, \mathcal{D}^{\mathcal{R}^*})$ gemäß Definition 3.24 sowie die Arbeitsmenge $\overline{\mathcal{V}}^{\mathcal{R}^*} \leftarrow \mathcal{V}^{\mathcal{R}^*}$

// Bestimme Referenzpunkt des Ego-Fahrzeuges auf dem Voronoi-Pfad
// (Wähle beliebigen Minimierer, falls nicht eindeutig)
$p_{\text{Ego}}^{\mathcal{R}^*} \leftarrow \arg\min_{p \in \mathcal{V}^{\mathcal{R}^*}} \mathrm{d}(p, p_{\text{Ego}})$;
// Iteriere über priorisierte Voronoi-Knoten
while $\overline{\mathcal{V}}^{\mathcal{R}^*} \neq \emptyset$ **do**
  // Wähle aktuell priorisierten Knoten (beliebig, falls nicht eindeutig)
  $p_{\text{Ego},T} \leftarrow \arg\min_{p \in \mathcal{V}^{\mathcal{R}^*}} f_T(p)$;
  // Rückwärtssuche zur Effizienzsteigerung
  Verwende den A*-Algorithmus 2.2 mit $v^{(\text{start})} := p_{\text{Ego},T}$, $v^{(\text{end})} := p_{\text{Ego}}^{\mathcal{R}^*}$ sowie mit der Heuristikfunktion h_{eukl} zur Berechnung eines (inversen) Pfades $\tilde{P}_T \leftarrow \langle p_{\text{Ego},T}, p_{\text{Ego},T}^{\rightarrow}, \ldots, p_{\text{Ego}}^{\mathcal{R}^*} \rangle$ zum Voronoi-Referenzpunkt des Ego-Fahrzeuges;
  // Prüfe, ob ein nicht trivialer Pfad berechnet werden konnte
  // $|\tilde{P}_T|$ entspricht der Anzahl der Knoten im Pfad $\tilde{P}_T$
  if $|\tilde{P}_T| > 1$ **then**
    $P_T \leftarrow \langle p_{\text{Ego}}^{\mathcal{R}^*}, \ldots, p_{\text{Ego},T}^{\rightarrow}, p_{\text{Ego},T} \rangle$ als Inverses des Lösungspfades $\tilde{P}_T$;
    // Generiere die Zentren einer Kreisüberdeckung des Ego-Fahrzeugs durch den Referenzpunkt $p_{\text{Ego},T}$ und der Orientierung $\varphi_{\text{Ego},T}$
    $\varphi_{\text{Ego},T} \leftarrow \operatorname{arctan2}\left(p_{\text{Ego},T_y} - p_{\text{Ego},T_y}^{\rightarrow}, p_{\text{Ego},T_x} - p_{\text{Ego},T_x}^{\rightarrow}\right)$;
    $\psi_{\text{Ego},T}^{\circ} \leftarrow \psi_{\text{Ego}}^{\circ}(p_{\text{Ego},T}, \varphi_{\text{Ego},T})$;
    // Prüfe Zulässigkeit von Pfad und Kreisüberdeckung
    if $\mathrm{d}(\psi_{\text{Ego},T}^{\circ}, \gamma) > 0$ **and** `PathValid`(P_T) **then**
      return P_T;
    end
  end
  $\overline{\mathcal{V}}^{\mathcal{R}^*} \leftarrow \overline{\mathcal{V}}^{\mathcal{R}^*} \setminus \{p_{\text{Ego},T}\}$; // Entferne getesteten Punkt aus Arbeitsmenge
end
return $\langle\rangle$; // Fehlschlag



(a) Geradeaus und links, $f_T = f_T^{\text{gl}}$

(b) Geradeaus und rechts, $f_T = f_T^{\text{gr}}$

(c) Möglichst nahe an Explorationspunkt $p_\diamond := (15,1{,}5)$, $f_T = f_T^{p_\diamond}$

(d) Geradeaus, $f_T = f_T^{\text{g}}$, Pfadlänge kleiner als 10 m

Abbildung 3.30: Teilpfade des statischen Voronoi-Pfades aus Abbildung 3.22 als Ergebnis von Algorithmus 3.3 für verschiedene Kostenfunktionen und Beschränkungen. Der Startpunkt entspricht dem Referenzpunkt p_{Ego} auf dem Zentrum der Hinterachse des Fahrzeuges und befindet sich im Koordinatenursprung. Die resultierende Zielkonfiguration wird durch das hell überzeichnete Fahrzeugbild repräsentiert.

dabei eine Rückwärtssuche durchgeführt, also vom aktuellen Zielkandidaten $p_{\text{Ego},T}$ ausgehend gestartet, vergleiche Abschnitt 2.3. Dies ermöglicht die Wiederverwendung von bereits bekannten verbleibenden Distanzen zum Referenzpunkt $p_{\text{Ego}}^{\mathcal{R}^*}$ aus potentiellen vorigen Iterationen und damit ein effizienteres Verfahren; dies wird am Ende des Abschnittes genauer vorgestellt. Die euklidische Heuristik h_{eukl} wird als kanonischer Ansatz in der A*-Suche verwendet. Existiert eine Lösung, so ergibt sich der tatsächliche Pfad P_T dann einfach durch deren Invertierung, wie in Zeile 6 dargestellt. Um die Zulässigkeit der gefundenen Lösung abschließend zu bewerten, wird auf eine positive Distanz der resultierenden Kreisüberdeckung des Ego-Fahrzeuges zum gegebenen Freibereichspolygon geprüft. Die entsprechende Orientierung kann dazu, wie in Zeile 7 gezeigt, mit der erweiterten inversen Tangens-Winkelfunktion

$$\arctan2 : \mathbb{R} \times \mathbb{R} \setminus \{(0,0)\} \to (-\pi,\, \pi]$$

direkt aus den Elementen des Lösungspfades berechnet werden. Zudem kann es zusätzliche Anforderungen an den Pfad geben – zum Beispiel aus der taktischen Entscheidungsfindung – welche durch eine entsprechend vorgegebene Funktion `PathValid` überprüft werden. Sind alle Anforderungen erfüllt, wird der Lösungspfad zurückgegeben. Kann kein zulässiger Knoten gefunden werden, so ist die Rückgabe ein leerer Pfad.

Abbildung 3.30 zeigt exemplarische Lösungspfade für verschiedene Kostenfunktionen sowie Beschränkungen. Als Referenzpunkt p_{Ego} des Ego-Fahrzeuges wird dabei, dem Einspurmodell aus Abschnitt 3.1.1 entsprechend, das Zentrum der Hinterachse verwendet. Es ist zu sehen, dass die Zielvorgabe durch die Kostenfunktion jeweils sinnvoll umgesetzt wird. Die resultierende Zielkonfiguration führt darüber hinaus in keinem Fall zu einem Konflikt mit dem Freibereichspolygon und der zugehörige Pfad hat jeweils minimale Länge – dies ergibt sich unmittelbar aus der Optimalitätseigenschaft des A*-Algorithmus aus Abschnitt 2.4. Zusätzliche Beschränkungen, wie die Einschränkung der Pfadlänge in Abbildung 3.30d, werden ebenfalls eingehalten.

Die benötigten Zeiten zur Erstellung der gezeigten Voronoi-Teilpfade sind in Tabelle 3.1 dargestellt. Präsentiert werden drei verschiedene Suchstrategien, um den Einfluss der wiederverwendeten Teilpfade auf die Effizienz des Verfahrens deutlich zu machen. Die numerische Erstellung des Voronoi-Graphen $G_{\mathcal{D}}^{\mathcal{R}^*}$ dauert dabei unabhängig von diesen unterschiedlichen Ansätzen 0,25 ms. Werden keine bereits bekannten Teilpfade zur Effizienzverbesserung verwendet (Suchstrategie **Schlicht**), so wird ein Gesamtaufwand von 2,11 ms benötigt. Die Rechenzeiten der einzelnen Beispiele sind in diesem Fall ein grober Indikator für die Anzahl der benötigten Iterationen von Algorithmus 3.3. Dies macht sich insbesondere beim Beispiel aus Abbildung 3.30c bemerkbar, bei dem direkt ein zulässiger Pfad gefunden wird und damit nur 0,02 ms gebraucht werden. Dies steht im starken Gegensatz zu dem Beispiel aus Abbildung 3.30d, welches aufgrund der Pfadlängenbeschränkung sowohl in den Iterationen als auch in der Berechnungszeit wesentlich aufwändiger ist. Werden bekannte Teilpfade aus vorigen Iterationen innerhalb eines Beispiels verwendet

Tabelle 3.1: Rechenzeiten von Algorithmus 3.3 für die Erstellung von Voronoi-Teilpfaden zur Bestimmung der Zielkonfigurationen in Abbildung 3.30. Die gezeigten Suchstrategien unterscheiden sich dabei wie folgt: **Schlicht** – Kein Verwendung von bekannten Teilpfaden; **Separat** – Verwendung von bekannten Teilpfaden nur innerhalb eines Beispiels; **Kollektiv** – Konsekutive Ausführung der Beispiele und Verwendung der bekannten Teilpfaden aus den vorigen Berechnungen. Die **Iterationen** entsprechen den notwendigen Schleifendurchläufen des Verfahrens (unabhängig von der Suchstrategie).

Suchstrategie	**Setup**	**Beispiel aus Abbildung**				$\sum$
	von $G_{\mathcal{D}}^{\mathcal{R}^*}$	**3.30a**	**3.30b**	**3.30c**	**3.30d**	
Schlicht	0,25 ms	0,34 ms	0,22 ms	0,02 ms	1,28 ms	2,11 ms
Separat		0,25 ms	0,18 ms	0,07 ms	0,61 ms	1,36 ms
Kollektiv			0,02 ms	0,01 ms	0,07 ms	0,6 ms
Iterationen	–	22	23	1	137	183

(Suchstrategie **Separat**), so kann die benötigte Rechenzeit auf insgesamt 65 % im Vergleich zur Suchstrategie **Schlicht** reduziert werden.
Unter der Annahme, dass die Aufgaben sukzessive ausgeführt werden, können bekannte Teilpfade auch übergreifend für verschiedene Zielkriterien verwendet werden (Suchstrategie **Kollektiv**). In diesem Fall ergibt sich für das erste Szenario aus Abbildung 3.30a zunächst derselbe Aufwand, anschließend aber nochmal eine deutliche Reduktion in allen Beispielen, sodass die Gesamtrechenzeit sich auf nur 28 % im Vergleich zur Suchstrategie **Schlicht** verringert.
Dies zeigt, dass ein Wiederverwenden der bekannten Zwischenergebnisse und die damit verbundene Rückwärtssuche zu einer deutlich verbesserten Effizienz der Zielpfadberechnung in Algorithmus 3.3 führt.

3.4 Der generische Hybrid A*-Algorithmus für ein autonomes Fahrzeug

Der in Abschnitt 2.5 vorstellte Hybrid A*-Algorithmus 2.3 beschreibt eine grundsätzliche Idee, mit der Trajektorien für dynamische Szenarien im Kontext einer graphenbasierten Suchstrategie berechnet werden können. Dies ist zunächst nicht auf eine bestimmte Anwendung festgelegt, sondern lässt sich, je nach Umsetzung, auf verschiedenste Probleme und Systeme anwenden. Im Rahmen dieser Arbeit wird das autonome Fahren als Anwendungsfall betrachtet, sodass für diesen Kontext in den vorangegangenen Abschnitten 3.1 bis 3.3 ein kinematisches Fahrzeugmodell,

räumliche Beschränkungen und der dynamische Voronoi-Pfad sowie das zugehörige Potentialfeld eingeführt wurden. All diese Komponenten werden abschließend im Folgenden mit Kosten, Heuristiken und einer Knotendiskretisierung kombiniert, um einen gewichteten hybriden Graphen im Sinne von Abschnitt 2.5.1 zu definieren. Dies resultiert schließlich in einer konkreten Implementierung des generischen Hybrid A*-Algorithmus zur Bewegungsplanung für ein autonomes Fahrzeug.

Ein entscheidendes Merkmal des Hybrid A*-Algorithmus ist die verwendete Systemdynamik, welche hier dem Einspurmodell aus Abschnitt 3.1.1 entspricht. Für den entsprechenden Zustandsvektor $z_{\text{Ego}} \in \mathbb{R}^4$ wird dann eine Knotendiskretisierung durch

$$\lceil z_{\text{Ego}} \rfloor := \begin{pmatrix} \lceil x_r / \delta_p \rfloor_{\text{round}} \cdot \delta_p \\ \lceil y_r / \delta_p \rfloor_{\text{round}} \cdot \delta_p \\ \lceil \varphi / \delta_\varphi \rfloor_{\text{round}} \cdot \delta_\varphi \\ \text{clamp}\left(\lceil v_r / \delta_{v_r} \rfloor_{\text{round}} \cdot \delta_{v_r},\, v_{\min},\, v_{\max} \right) \end{pmatrix} \tag{3.17}$$

definiert, wobei $\lceil \cdot \rfloor_{\text{round}}$ der Rundungsfunktion entspricht. Der Parameter $\delta_p \in \mathbb{R}_{>0}$ heißt *räumliche Diskretisierung*, $\delta_\varphi \in \mathbb{R}_{>0}$ *Orientierungsdiskretisierung* und $\delta_{v_r} \in \mathbb{R}_{>0}$ *Geschwindigkeitsdiskretisierung*. Der Wert der diskreten Geschwindigkeit wird dabei durch

$$\begin{aligned} &\text{clamp}\colon \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \\ &(x,\, x_l,\, x_u) \mapsto \min(\max(x,\, x_l),\, x_u) \end{aligned}$$

auf das Intervall $[v_{\min},\, v_{\max}]$ eingeschränkt, wobei $v_{\min} \in \mathbb{R}_{\leq 0}$ und $v_{\max} \in \mathbb{R}_{>0}$ situationsabhängig gewählt werden. Zur *Zeitdiskretisierung* der Bewegung von Objekten wird zudem die Schrittweite δ_t verwendet, sodass sich das Zeitgitter

$$\{t_0,\, t_0 + \delta_t,\, t_0 + 2 \cdot \delta_t,\, \ldots\} \subset \mathbb{R}$$

ergibt, ausgehend von einer Startzeit $t_0 \in \mathbb{R}$, vergleiche Abschnitt 3.3.3.
Zur Beschreibung der möglichen diskreten Steuerungen werden die Mengen der Beschleunigungen

$$\begin{aligned} &\overline{U}_{a_r} := \{\bar{a}_r^{(1)}, \ldots, \bar{a}_r^{(m)}, 0, \bar{a}_r^{(m+1)}, \ldots, \bar{a}_r^{(n_{a_r})}\}, \quad \text{mit} \\ &\bar{a}_r^{(i)} \in [a_r^{\min}, 0) \subset \mathbb{R}_{<0}\ \forall i\colon 1 \leq i \leq m \quad \text{und} \\ &\bar{a}_r^{(i)} \in (0, a_r^{\max}] \subset \mathbb{R}_{>0}\ \forall i\colon m + i \leq i \leq n_{a_r}, \end{aligned}$$

sowie der Lenkwinkel

$$\begin{aligned} &\overline{U}_{\delta_f} := \{-\bar{\delta}_f^{(n_{\delta_f})}, \ldots, -\bar{\delta}_f^{(1)}, 0, \bar{\delta}_f^{(1)}, \ldots, \bar{\delta}_f^{(n_{\delta_f})}\}, \quad \text{mit} \\ &\bar{\delta}_f^{(i)} \in (0, \delta_f^{\max}] \subset \mathbb{R}_{>0}\ \forall i\colon 1 \leq i \leq n_{\delta_f} \end{aligned}$$

festgelegt. Aus diesen ergibt sich schließlich direkt die Steuermenge

$$\overline{U} \coloneqq \{(\bar{a}_r,\, \bar{\delta}_f) \mid \bar{a}_r \in \overline{U}_{a_r} \text{ und } \bar{\delta}_f \in \overline{U}_{\delta_f}\}$$

als Zusammenfassung aller möglicher Kombinationen beider Komponenten.

Systemdynamik, Knotendiskretisierung und Steuermenge definieren schließlich einen hybriden Graphen $G^{\mathcal{H}}$; um diesen im generischen Hybrid A*-Algorithmus zum Finden einer *optimalen* Trajektorie zwischen zwei Zuständen $z^{(\text{start})}$ und $z^{(\text{end})}$ verwenden zu können, müssen seinen Kanten jedoch noch entsprechende Kosten zugeordnet werden. Dies wird wie folgt durch insgesamt drei unterschiedliche Optimierungskriterien und zugehörige Kostenterme umgesetzt. Diese sind im weiteren Verlauf jeweils allein abhängig vom qualitativen Unterschied der diskreten Repräsentation eines hybriden Knotens $v^{\mathcal{H}}_{\bar{u}|\mathcal{V}}$ zu seinem Vorgänger $v^{\mathcal{H}}_{|\mathcal{V}}$ beziehungsweise einem gegebenen Referenzwert. Um die einzelnen Terme dabei untereinander zu normieren, werden verschiedene Distanzschätzungen verwendet:

- $\mathrm{d}_{\oplus}$: Geschätzte Gesamtdistanz zwischen $\lceil z^{(\text{start})} \rfloor$ und $\lceil z^{(\text{end})} \rfloor$.
- $\mathrm{d}_{\odot}$: Geschätzte verbleibende Distanz zwischen $v^{\mathcal{H}}_{\bar{u}|\mathcal{V}}$ und $\lceil z^{(\text{end})} \rfloor$.
- $\mathrm{d}_{\ominus}$: Aktuelle Schrittweite als zurückgelegte Distanz zwischen $v^{\mathcal{H}}_{|\mathcal{V}}$ und $v^{\mathcal{H}}_{\bar{u}|\mathcal{V}}$.

Alle Approximationen lassen sich mit dem Verfahren von Dubins oder Reeds und Shepp berechnen, vergleiche Abschnitt 3.1.2. Im weiteren Verlauf wird dabei stets letzteres verwendet, um vergleichbare Auswertungen in beide Fahrtrichtungen zu ermöglichen. Mögliche Auswirkungen auf die Rechenzeit werden in Abschnitt 4.1.2 im Detail untersucht.
Schließlich werden Optimierungskriterien durch die folgenden Kantenkosten formuliert:

1. Es soll eine möglichst kurze Trajektorie zum Ziel gefunden werden. Dazu wird die Expansion eines neuen Knotens konstant durch

   $$d_p \coloneqq 1$$

   bestraft.

2. Es soll eine vorgegebene Geschwindigkeit angenommen werden. Seien dazu v_{set} die diskretisierte Geschwindigkeit des Zielzustandes $z^{(\text{end})}$ sowie $\bar{v}$ diejenige des aktuell betrachteten Knotens $v^{\mathcal{H}}_{\bar{u}|\mathcal{V}}$. Damit werden die entsprechenden Kantenkosten als normierte quadratische Abweichung

   $$d_v \coloneqq \frac{(\bar{v} - v_{\text{set}})^2}{\max(v^2_{\text{set}},\, v^2_{\text{set,\,min}})}$$

   definiert. Die Normierung erfolgt dabei gegenüber dem vorgegebenen v_{set}, wobei diese durch einen konstanten Parameter $v_{\text{set,\,min}} \in \mathbb{R}_{>0}$ nach unten beschränkt wird.

3. Es sollen sichere Manöver berechnet werden. Entspricht t der Zeit und $\psi^{\circ}_{\text{Ego}} = \psi^{\circ}_{\text{Ego}}(v^{\mathcal{H}}_{\bar{u}\,|\mathcal{V}})$ der Kreisüberdeckung des Ego-Fahrzeugs im betrachteten diskreten Knoten, dann wird dies durch den Wert des Voronoi-Potentials $\Phi^{*}_{V}(\psi^{\circ}_{\text{Ego}}, t)$ bewertet. Um in der Nähe des Zielzustandes eine höhere Manövrierfähigkeit zu gewähren, wird dieser Term zusätzlich mit der relativen verbleibenden Gesamtdistanz $\mathrm{d}_{\odot}/\mathrm{d}_{\oplus}$ gewichtet, sodass insgesamt

$$d_{\mathcal{R}} \coloneqq \Phi^{*}_{V}(\psi^{\circ}_{\text{Ego}}, t) \cdot \frac{\mathrm{d}_{\odot}}{\mathrm{d}_{\oplus}}$$

resultiert.

Alle Optimierungskriterien werden mit der relativen Schrittweite $\mathrm{d}_{\ominus}/\mathrm{d}_{\oplus}$ normiert, sodass sich beispielsweise die Gesamtkosten der Pfadlänge zu ungefähr 1 ergeben. Abschließend werden die Terme untereinander durch $\omega_p, \omega_v, \omega_{\mathcal{R}} \in \mathbb{R}_{\geq 0}$ gewichtet, um die Kostenfunktion

$$\mathcal{D}^{\mathcal{H}}(v^{\mathcal{H}}, \cdot, v^{\mathcal{H}}_{\bar{u}}) \coloneqq (\omega_p d_p + \omega_v d_v + \omega_{\mathcal{R}} d_{\mathcal{R}}) \cdot \frac{\mathrm{d}_{\ominus}}{\mathrm{d}_{\oplus}}$$

und damit den gewichteten hybriden Graphen

$$G^{\mathcal{H}}_{\mathcal{D}} \coloneqq (\mathcal{V}^{\mathcal{H}}, U^{\mathcal{H}}, \mathcal{D}^{\mathcal{H}}) \tag{3.18}$$

zu definieren. Zusätzlich oder alternativ zu den hier betrachteten Komponenten aus den Punkten 1 bis 3 könnten auch andere Zielkriterien durch die hybriden Kantenkosten dargestellt werden; beispielsweise könnten *sanftere* Trajektorien durch eine Minimierung des verwendeten Steueraufwandes erreicht werden, indem deren Magnitude ebenfalls als Kantenkosten berücksichtigt wird. Da jedoch die Wahl von passenden Optimierungskriterien stets stark vom jeweils betrachteten Anwendungsfall abhängt, wird auf eine Analyse, welche über die hier dargestellten Komponenten hinausgeht, im weiteren Verlauf verzichtet.

Basierend auf $\mathcal{D}^{\mathcal{H}}$ wird der Wert der Heuristikfunktion im betrachteten Knoten dann durch

$$\begin{aligned} h(v^{\mathcal{H}}_{\bar{u}\,|\mathcal{V}}) &\coloneqq (\omega_p d_p + \omega_v d_v + \omega_{\mathcal{R}} d_{\mathcal{R}}) \cdot \frac{\mathrm{d}_{\odot}}{\mathrm{d}_{\oplus}} \\ &=: \omega_p h_p + \omega_v h_v + \omega_{\mathcal{R}} h_{\mathcal{R}} \end{aligned} \tag{3.19}$$

festgelegt. Im Vergleich zur Kostenfunktion ändert sich dabei nur die Skalierung, sodass statt der relativen Schrittweite die relative verbleibende Distanz zum Ziel verwendet wird. Reduziert auf die Komponente h_p zur Minimierung der Trajektorienlänge führt dies stets zu einer monotonen Teilheuristik und würde gemäß Abschnitt 2.4 die Optimalität des A*-basierten Suchverfahrens implizieren. Für die Komponenten zur Bewertung der Geschwindigkeit h_v sowie Lage im Voronoi-Potential $h_{\mathcal{R}}$ gilt diese Aussage dagegen nicht. Unabhängig von den jeweiligen Kosten

im aktuellen Knoten könnten beide Terme für alle weiteren Elemente den Wert 0 annehmen, sodass die einzige nicht negative aber trotzdem monotone Heuristik durch $h_v \equiv h_{\mathcal{R}} \equiv 0$ gegeben wäre. Wie in Abschnitt 2.4.1 dargestellt würde dies einem vergleichsweise aufwändigem Dijkstra-Verfahren entsprechen. Die in (3.19) verwendete Heuristikfunktion stellt dagegen eine eher pessimistische Schätzung dar, in dem Sinne, dass sie Konstanz und keine Verbesserung in den aktuellen Kosten bezüglich Geschwindigkeit und Voronoi-Potential erwartet. Dies führt gegebenenfalls zu einem gezielten Überschätzen und damit dem Verlust der Optimalitätsgarantie, was in Abwägung mit einer verbesserten Effizienz jedoch akzeptiert werden kann.

Mit den vorstehenden Definitionen wird der generische Hybrid A*-Algorithmus 3.4 für die Anwendung in einem autonomen Fahrzeug definiert. Durch diesen wird eine Trajektorie ausgehend von einem gegebenen Startzustand $z^{(\text{start})}$ zu einem implizit festgelegten Ziel berechnet, wobei sowohl beliebige statische als auch räumliche Beschränkungen sowie eine vorgegebene Zielgeschwindigkeit berücksichtigt werden. Durch die Konzepte aus den Abschnitten 3.2 bis 3.3 werden dann automatisiert ein Freibereichspolygon, statische und dynamische Voronoi-Pfade sowie ein Zielzustand generiert, um schließlich eine Trajektorie mit dem Suchverfahren aus Algorithmus 2.3 zu berechnen. Das Terminierungskriterium in `IsTerminating` stoppt das Verfahren dann, wenn ein offener Knoten mit identischer diskreter Position zum Endzustand $z^{(\text{end})}$ zur Expansion ausgewählt wird. Je nach Anforderung kann dies auch auf einen Vergleich mit der Orientierung oder Geschwindigkeit erweitert werden. Die Kollisionsüberprüfung `IsCollision` ergibt sich ohne expliziten Aufwand direkt aus der Berechnung des Voronoi-Potentials für die aus dem betrachteten hybriden Knoten resultierende Kreisüberdeckung, vergleiche Abschnitt 3.3.4. Dabei werden auch Konflikte mit dem Freibereichspolygon als Kollision gewertet. Da der Startzustand $z^{(\text{start})}$ für den Hybrid A*-Algorithmus in dessen Inneren liegt, ist die Menge der diskreten Positionen stets beschränkt. Mit der Anwendung von clamp in (3.17) gilt dies auch für die Geschwindigkeitswerte. Die Orientierungen können schließlich kanonisch auf das Intervall $[-\pi, \pi]$ projiziert werden. Damit ist die Gesamtheit aller diskreten Fahrzeugzustände beschränkt und die Voraussetzung für die Vollständigkeit von Algorithmus 3.4 gegeben, vergleiche Satz 2.39.

Algorithmus 3.4 : Implementierung des generischen Hybrid A*-Algorithmus zur automatisierten Berechnung zulässiger Trajektorien für ein autonomes Fahrzeug

Eingabe : Aktueller Fahrzeugzustand $z^{(\text{start})} \in \mathbb{R}^4$ und Zeit $t_0 \in \mathbb{R}$, räumliche Beschränkungen aus Punktwolke $D \subset \mathbb{R}^2$ und Polygonen $\rho^B \subset \Lambda$, (lokale) Expansionstiefe und -abstand $\kappa \in \mathbb{R}_{>0}$, $\kappa_\rho \in \mathbb{R}_{>0}$, $\xi \in \mathbb{R}_{\geq 0}$, zeitabhängige Bounding Boxen $\Psi^\square$ und Kreisüberdeckungen Ψ° einer Menge von n_Ψ dynamischen Objekten, Kostenfunktion $f_T \colon \mathbb{R}^2 \to \mathbb{R}$ und Testfunktion für Beschränkung `PathValid` zur Berechnung eines Voronoi-Teilpfades, Zielgeschwindigkeit $v_{\text{set}} \in \mathbb{R}$

Ausgabe : Im Erfolgsfall Trajektorie $P^{\mathcal{H}}$ von $z^{(\text{start})}$ ausgehend in Richtung durch f_T und `PathValid` beschriebenen Zielkonfiguration, ansonsten leere Trajektorie $\langle\rangle$

`// Verarbeitung der Umgebungsinformationen`

1 Berechne Freibereichspolygon $\gamma \in \Gamma$ zur Darstellung aller Umgebungsinformationen in D und ρ^B durch Algorithmus 3.1, mit $z^{(\text{start})}$ als initialem Expansionszentrum sowie (lokaler) Expansionstiefe und -abstand κ, κ_ρ, ξ;

`// Voronoi-Pfade`

2 Berechne den statischen und die dynamischen Voronoi-Pfade $\mathcal{R}^*$ und $\mathcal{R}_t^*$ für das Freibereichspolygon γ und die zeitabhängigen Bounding Boxen $\Psi^\square$;

`// Bestimmung des Zielzustandes`

3 Berechne einen Voronoi-Teilpfad P_T durch Algorithmus 3.3 für den statischen Pfad $\mathcal{R}^*$, das Freibereichspolygon γ sowie die Beschreibung der Zielkonfiguration f_T und `PathValid`;

4 Definiere den Zielzustand $z^{(\text{end})}$ als Kombination des Zielreferenzpunktes $p_{\text{Ego},T}$ und der Zielorientierung $\varphi_{\text{Ego},T}$ aus dem Voronoi-Teilpfad (vergleiche Zeile 8 von Algorithmus 3.3) sowie der vorgegebenen Referenzgeschwindigkeit v_{set};

`// Berechne Trajektorie und gebe sie zurück, ist bei Fehlschlag leer`

5 **return** Trajektorie $P^{\mathcal{H}}$ von $z^{(\text{start})}$ zu $z^{(\text{end})}$, berechnet durch den generischen Hybrid A*-Algorithmus 2.3 mit dem gewichteten Graphen $G_{\mathcal{D}}^{\mathcal{H}}$ aus (3.18), der Heuristikfunktion aus (3.19), der Knotendiskretisierung $\lceil \cdot \rfloor$ aus (3.17), der Startzeit t_0, `IsCollision` zur Kollisionsüberprüfung sowie `IsTerminating` als Terminierungskriterium;

Funktion `IsCollision`$(v^{\mathcal{H}})$

// Überprüfe, ob die Kreisüberdeckung des Ego-Fahrzeuges, basierend auf der kontinuierlichen Repräsentation $v^{\mathcal{H}}_{|\mathcal{Z}}$ eines gegebenen hybriden Knotens, zur Zeit t eine Kollision implziert, vergleiche die Definition von (3.15) aus Abschnitt 3.3.4

return $\mathrm{d}\left(\psi^{\circ}_{\mathrm{Ego}}\left(v^{\mathcal{H}}_{|\mathcal{Z}}\right), \gamma, \Psi^{\circ}(t)\right) \leq 0;$

Funktion `IsTerminating`$(v^{\mathcal{H}}, z^{(end)})$

// Überprüfe, ob die diskrete Repräsentation $v^{\mathcal{H}}_{|\mathcal{V}}$ eines gegebenen hybriden Knotens identisch mit dem diskretisierten Zielzustand $\lceil z^{(\mathrm{end})} \rfloor$ ist. Es werden dabei nur die Punktinformationen, also jeweils die x- und y-Koordinaten, gemäß Abschnitt 3.2.1.1 miteinander verglichen.

return $\mathrm{d}\left(v^{\mathcal{H}}_{|\mathcal{V}}, \lceil z^{(\mathrm{end})} \rfloor\right) = 0;$

Kapitel 4

Hybride Trajektorienoptimierung und taktische Entscheidungen

Die vorangegangenen Kapitel haben den generischen Hybrid A* vorgestellt sowie seine Komponenten – wie etwa Freibereichspolygone und Voronoi-Pfade – eingeführt und numerisch untersucht. Im weiteren Verlauf wird diese Analyse auf den generischen Hybrid A*-Algorithmus 3.4 zur Bewegungsplanung selbst ausgeweitet. Dazu werden dessen Ergebnisse und Optimierungsverläufe in Abschnitt 4.1 zunächst anhand von exemplarischen Lösungen basierend auf simulierten Problemen diskutiert. Abschließend präsentiert Abschnitt 4.2, wie die in dieser Arbeit vorgestellten Verfahren im Rahmen einer taktischen Entscheidungsfindung eines realen autonomen Forschungsfahrzeugs eingesetzt werden können. Dies wird anhand von zwei Testfahrten in verschiedenen Szenarien numerisch analysiert.

4.1 Simulative Evaluation des generischen Hybrid A*-Algorithmus

Es werden in Abschnitt 4.1.1 zunächst verschiedene simulierte Probleme vorgestellt und die Lösungen des generisches Hybrid A*-Algorithmus 3.4 gezeigt und diskutiert. Dabei wird wie zuvor ein besonderer Fokus auf die jeweiligen Rechenzeiten gelegt. Um diese möglichst zu minimieren, werden in der Implementierung des Bewegungsplaners vor allem Distanzberechnungen sehr effizient umgesetzt; die dafür genutzten Ansätze werden in Abschnitt 4.1.2 präsentiert. Schließlich steht in Abschnitt 4.1.3 das Voronoi-Potential und dessen Einfluss auf die Güte der resultierenden Trajektorien im Fokus.

Die für die Auswertungen in diesem Abschnitt verwendeten Hyperparameter der Methoden aus Kapitel 3 ist in Tabelle A.1 im Anhang aufgelistet.

4.1.1 Exemplarische Lösungen

Im Folgenden werden auf simulativer Ebene verschiedene Szenarien betrachtet, welche typischen Situationen aus dem realen Straßenverkehr ähneln. Zur Vereinfachung werden dabei nur räumliche Beschränkungen basierend auf künstlich erzeugten, leicht verrauschten Punktwolken angenommen. Das Ziel ist zunächst die Analyse des Algorithmus 3.4 zur Bewegungsplanung, weshalb an dieser Stelle außerdem weder Spurinformationen noch Verkehrsregeln berücksichtigt werden – dies erfolgt in der abschließenden Demonstration der taktischen Entscheidungsfindung in Abschnitt 4.2. Im weiteren Verlauf werden sowohl statische als auch dynamische Szenarien in den Abschnitten 4.1.1.1 bis 4.1.1.2 untersucht.

Um diese besser miteinander vergleichen zu können, sind Kennzahlen für alle Beispiele in Tabelle 4.1 zusammengefasst. Dies umfasst unter anderem die Größen von Punktwolken sowie die Anzahl der Elemente in den resultierenden Trajektorien, Freibereichspolygonen und Voronoi-Pfaden. Um letztere möglichst einheitlich zu berechnen, werden sie in den dynamischen Szenarien im Vorfeld für alle Zeitpunkte bereitgestellt, für welche Objekte innerhalb des Freibereichspolygons vorliegen. Wie in Abschnitt 3.3.4 diskutiert, kann dies für größere Effizienz auch anteilig auf die Laufzeit des Suchalgorithmus verlagert werden, wenn die zeitliche Planungsdauer voraussichtlich kürzer ist. Die Anzahl der notwendigen diskreten Zeitschritte ist zur Referenz in der Tabelle hinterlegt.
Zudem wird der Iterationsverlauf des Hybrid A*-Algorithmus anhand der Menge der geöffneten sowie expandierten Knoten dargestellt. Dabei wird ein geöffneter Knoten stets einem Kollisionscheck unterzogen, sodass sich daraus eine Schätzung der notwendigen Distanzberechnungen ableiten lässt. Zum Beispiel sind zur Kollisionsprüfung zwischen einem Freibereichspolygon und dem Ego-Fahrzeug, aufgrund von

Tabelle 4.1: Kenngrößen der Probleme und der A* Iterationen sowie Rechenzeiten für die Lösung der Szenarien aus den Abbildungen 4.1 bis 4.6 mit der Implementierung des Hybrid A*-Algorithmus 3.4. Rot markierte Einträge zeigen ein Überschreiten der Updatezeit für die taktische Entscheidungsfindung an. Ergebnisse für die beiden Trajektorien aus Abbildung 4.6b sind einzeln aufgeführt. Dynamische Voronoi-Pfade wurden mit 4 parallelen Prozessen durch Algorithmus 3.2 berechnet.

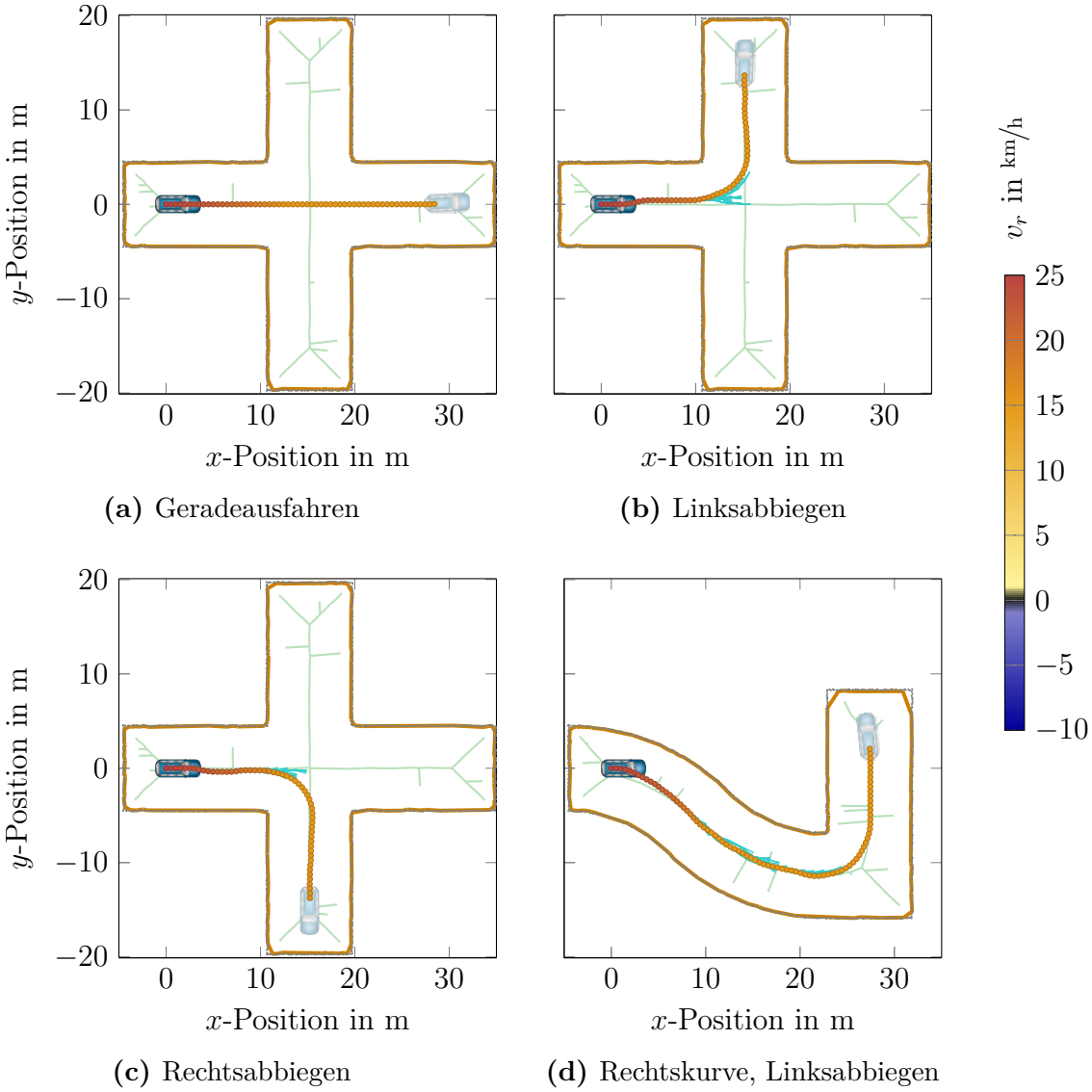
		Kreuzungen (statisch)				**Parken**		**Kreuzungen (dynamisch)**			
	Abbildung	4.1a	4.1b	4.1c	4.1d	4.2a	4.2b	4.4	4.5	4.6a	4.6b
Diskrete Zeitschritte		1	1	1	1	1	1	70	37	37	37
# Elemente in	Punktwolke	789	789	789	637	705	705	789	789	789	789
	Freibereichspolygon	114	114	114	83	118	124	114	114	114	114
	Voronoi-Pfad(e)	143	143	143	106	149	151	10 930	5421	5421	5421
	Trajektorie	58	54	55	85	44	73	68	64	35	34 + 32
A*	Geöffnete Knoten	1112	3808	3611	10 562	10 595	33 368	19 152	87 437	14 828	19 129 + 12 270
	Expandierte Knoten	85	334	309	1123	1535	2745	2187	10 382	1588	1860 + 1267
Rechenzeit in ms	Freibereichspolygon	3,7	3,7	3,7	1,8	3,4	3,4	3,7	3,7	3,7	3,7
	Voronoi-Pfade	2,5	2,5	2,5	1,6	2,7	2,9	39,1	14,6	12,1	12,1
	Zielzustand	0,2	0,2	0,2	0,2	–	–	0,3	0,3	0,3	–
	Trajektorie	1,2	3,4	3,2	8,5	16,3	26,7	19,4	103,9	38,6	35,4 + 9,4
	Gesamt	7,6	9,8	9,6	12,1	22,4	33,0	62,5	122,5	54,7	60,6

dessen vierfacher Kreisüberdeckung, jeweils vier Vergleiche notwendig. Mit Blick auf Abbildung 3.16 in Abschnitt 3.2.3 stellt dies eine erneute Rechtfertigung für die Verwendung von Freibereichspolygonen zur Approximation von Daten aus Punktwolken dar.
Als letzter Aspekt wird die Rechenzeit für die verschiedenen Bestandteile sowie die jeweils kumulierte Laufzeit von Algorithmus 3.4 gezeigt. Diese werden im Folgenden unter Berücksichtigung der maximalen Rechenzeit von 100 ms für die taktische Entscheidungsfindung analysiert, vergleiche dazu auch Abschnitt 1.2.

4.1.1.1 Statische Probleme

Zur Auswertung von statischen Aufgaben wird zunächst eine Kreuzungssituation mit den drei möglichen Ausgangsrichtungen *geradeaus*, *links* und *rechts* betrachtet, vergleiche Abbildungen 4.1a bis 4.1c. In allen Szenarien hat das Fahrzeug eine Startgeschwindigkeit von 25 $^{\mathrm{km}}/_{\mathrm{h}}$, mit dem Ziel, diese auf 15 $^{\mathrm{km}}/_{\mathrm{h}}$ zu reduzieren. Es lässt sich feststellen, dass die gegebene Punktwolke gut durch das Freibereichspolygon approximiert wird und der resultierende statische Voronoi-Pfad den Verlauf der Kreuzung beschreibt. Damit stellt er eine sinnvolle Referenz sowohl zur automatisierten Bestimmung der Zielkonfigurationen als auch zur Berechnung der Trajektorie im Rahmen des Voronoi-Potentials dar. Letzteres lässt sich gut am Verlauf der Lösung beobachten, welche fast durchgängig sehr große Nähe zum Voronoi-Pfad aufweist; einzig in den Abbildungen 4.1b und 4.1c wird dieser zum Abbiegen im Zentrum der Kreuzung verlassen. Dies entspricht jeweils der Stelle, an der die Optimierungskriterien zur Minimierung der Pfadlänge und des Voronoi-Potentials am stärksten konkurrieren: Während erstere die Expansion der Knoten in Richtung der Zielkonfiguration lenkt, zieht das Voronoi-Potential die Expansion eher in die Mitte der Kreuzung. Dieser Konflikt lässt sich an dem erhöhten Expansionsaufwand in den Grafiken erkennen. Das Erreichen der Zielgeschwindigkeit steht zu keiner Zeit im Widerspruch zu einem anderen Zielkriterium, sodass diese sehr schnell angenommen und dann auch gehalten wird. Die Trajektorie für das Geradeausfahren verläuft erwartungsgemäß frei von Kurven. Die Lösungen für das Links- und Rechtsabbiegen sind annähernd symmetrisch – vollständige Symmetrie kann zum Beispiel aufgrund der verrauschten Punktwolken nicht erwartet werden. Mit Blick auf Tabelle 4.1 kann festgehalten werden, dass die drei betrachteten Kreuzungssituationen sehr effizient durch Algorithmus 3.4 gelöst werden: In allen Fällen müssen maximal 3808 Knoten untersucht werden und die Rechenzeiten für die gesamte Verarbeitungskette bleibt stets unter 10 ms. Dabei hält die Vorverarbeitung der Eingabedaten im Rahmen der Generierung von Freibereichspolygon und Voronoi-Pfad den größten Anteil am Aufwand.

Ein leicht verändertes Bild ergibt sich für die Berechnung der Trajektorie in Abbildung 4.1d. Hier ist zunächst eine gestreckte Rechtskurve zu fahren, bevor ein Linksabbiegen um 90° notwendig ist. Diese entgegengesetzten Richtungen verstärken den

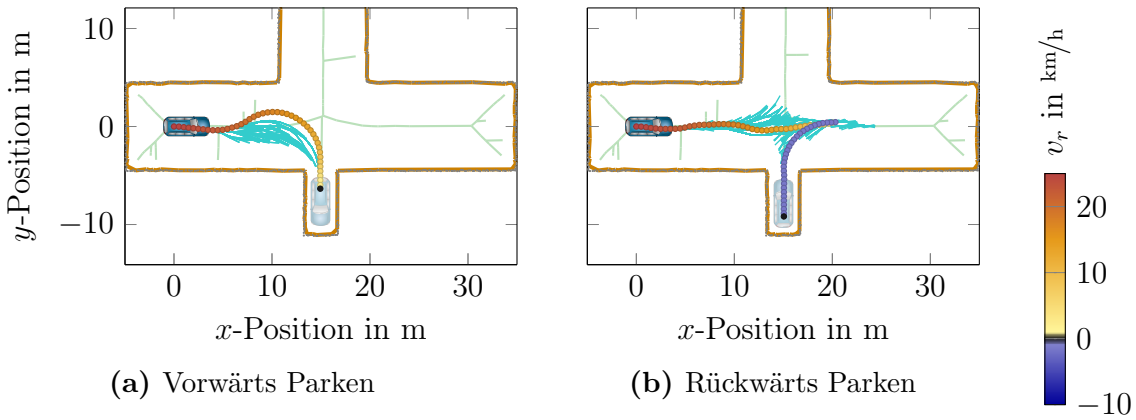


(a) Geradeausfahren **(b)** Linksabbiegen

(c) Rechtsabbiegen **(d)** Rechtskurve, Linksabbiegen

Abbildung 4.1: Exemplarische Lösungen von Algorithmus 3.4 für statisches Abbiegen an Kreuzungen. Gezeigt werden räumlichen Beschränkungen durch die Punktwolke (grau), das resultierende Freibereichspolygon (orange), der statische Voronoi-Pfad (hellgrün), Start- und Endkonfiguration des Fahrzeugs sowie die finale Trajektorie inklusive Geschwindigkeitsinformation. Die Expansion des zugrunde liegenden A*-Algorithmus wird durch die Pfade in türkis visualisiert.

zuvor erwähnten Konflikt zwischen den beiden Zielkriterien bezüglich Pfadlänge und Voronoi-Potential. Dies spiegelt sich ebenfalls in einem vermehrten Suchaufwand wider, wie auch an der Anzahl der geöffneten Knoten gut zu erkennen ist. Zusätzlich ist die resultierende Trajektorie im Vergleich zu den vorigen Beispielen deutlich länger, siehe hierzu Tabelle 4.1. Diese Kombination führt dazu, dass die Rechenzeit hier im



(a) Vorwärts Parken **(b)** Rückwärts Parken

Abbildung 4.2: Exemplarische Lösungen von Algorithmus 3.4 für statisches vorwärts und rückwärts Einparken

Wesentlichen für die Berechnung der Trajektorie benötigt wird; insgesamt ist aber auch dieses Beispiel mit einem Gesamtaufwand von 12,1 ms als sehr effizient lösbar zu bewerten.

Wird das Erreichen der Zielkonfiguration und damit das Lösen des Problems erschwert, so erhöht sich im Allgemeinen die Anzahl an expandierten und geöffneten Knoten und damit die benötigte Rechenzeit für die Trajektorie. Dies wird exemplarisch in Abbildung 4.2 deutlich gemacht, in welcher Manöver für vorwärts und rückwärts Einparken gezeigt werden. Die Startgeschwindigkeit beträgt dabei erneut 25 km/h, am Ziel werden in diesem Fall aber 0 km/h gefordert (dies wird als zusätzliches Kriterium in `IsTerminating` von Algorithmus 3.4 umgesetzt). Da die verfügbare Parklücke in Relation zur Breite des Ego-Fahrzeugs schmal ist, müssen für beide Szenarien ebenfalls vergleichsweise viele Knoten expandiert werden, um eine kollisionsfreie Lösung zu finden (beispielsweise in Relation zu den Situationen aus Abbildungen 4.1a bis 4.1c). Darüber hinaus schaffen die Geschwindigkeitsvorgaben eine zusätzliche Beschränkung für die Länge der Trajektorie, welche sich aus der Mindestlänge zum Anhalten ausgehend von der Startgeschwindigkeit ergibt.
Im direkten Vergleich der beiden Einparkrichtungen vorwärts und rückwärts zeigt sich, dass die Berechnung für Letzteres mit einem Unterschied von ca. 10 ms aufwändiger ist; beide erreichen jedoch eine Gesamtrechenzeit von unter 35 ms. Der wesentliche Anteil davon entfällt auf die Berechnung der Trajektorien. Hier benötigt insbesondere das Manöver zum rückwärts Einparken zusätzliche Zeit, was zum einen mit der erhöhten Länge der resultierenden Bahn zusammen hängt, zum anderen mit der zusätzlichen Komplexität, einen passenden Punkt für den Richtungswechsel zu identifizieren. Dieser Aspekt wird auch durch die Verläufe der kontinuierlichen Zustände und diskreten Steuerungen der Lösungstrajektorie hervorgehoben, welche in Abbildung 4.3 dargestellt sind. Anhand der Geschwindigkeit ist gut zu sehen, dass der Umschaltpunkt nach ca. 6 s erreicht wird. Anschließend wird die Parklücke mit

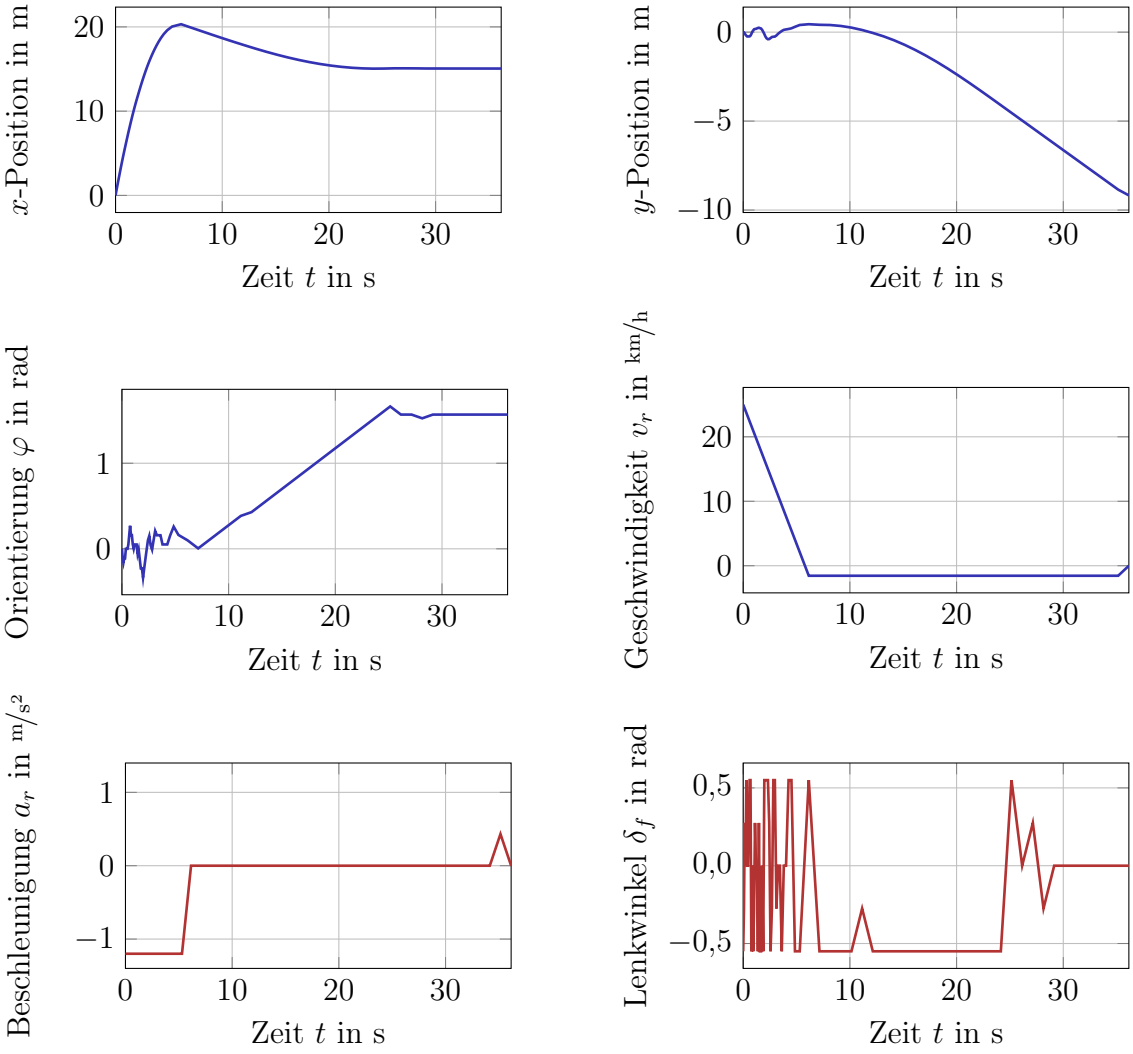
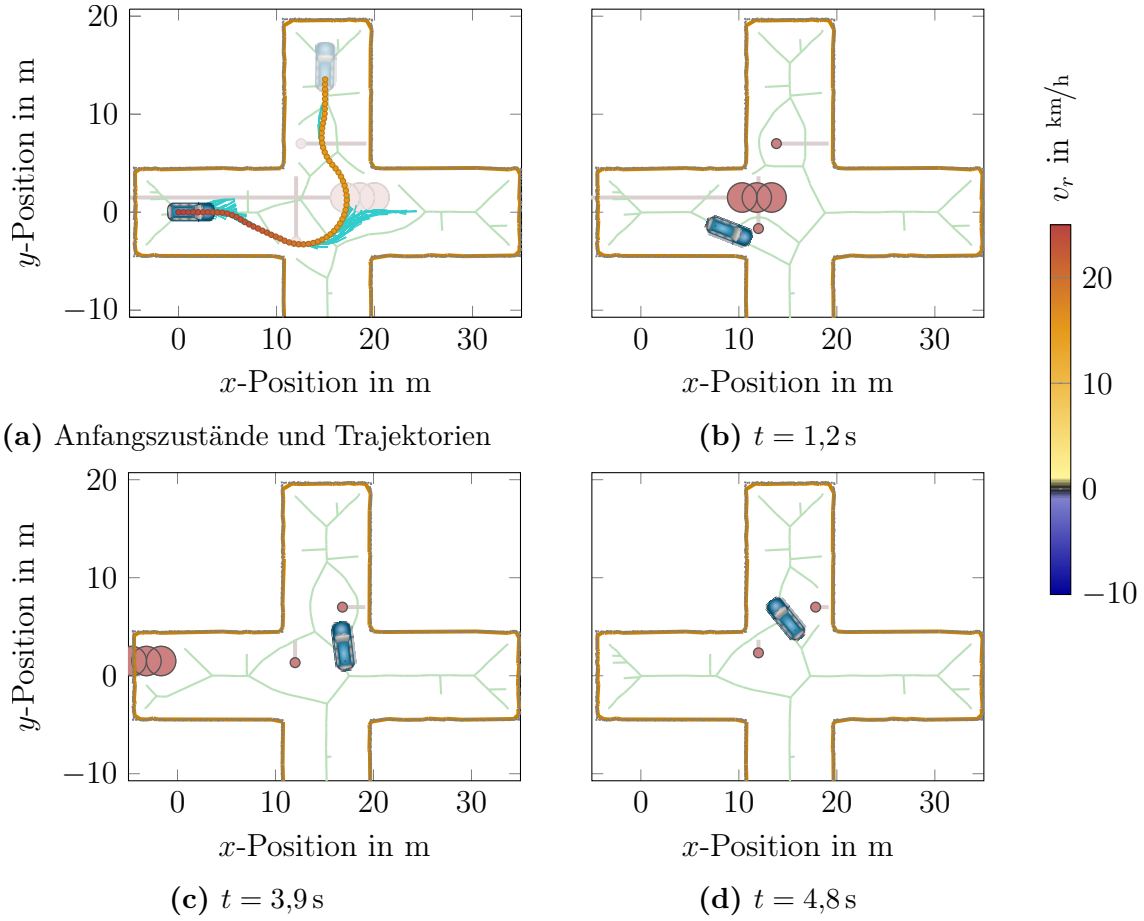


Abbildung 4.3: Zustände aus den kontinuierlichen Knotenrepräsentationen von Algorithmus 2.3 und zugehörige diskrete Steuerungen zur Berechnung des rückwärts Einparkmanövers aus Abbildung 4.2b

ungefähr 1,5 km/h angesteuert, bis das Manöver nach 36 s beendet ist. Insbesondere in der ersten Phase drückt sich die Komplexität der Suche in der äußerst unstetigen Auswahl der Lenkwinkel-Steuerungen aus. Dies hat auch Auswirkungen auf den Verlauf der Orientierung und in begrenztem Maße auch auf den der y-Position, welche hier beide vergleichsweise stark schwanken. Das rückwärtsgerichtete Anfahren der Parklücke in der zweiten Phase ist dann sowohl in den Steuerungen als auch in den Zuständen von wesentlich weniger Fluktuation geprägt, was wiederum mit der reduzierten Knotenexpansion für diesen Abschnitt im Einklang steht.



(a) Anfangszustände und Trajektorien **(b)** $t = 1{,}2\,\mathrm{s}$

(c) $t = 3{,}9\,\mathrm{s}$ **(d)** $t = 4{,}8\,\mathrm{s}$

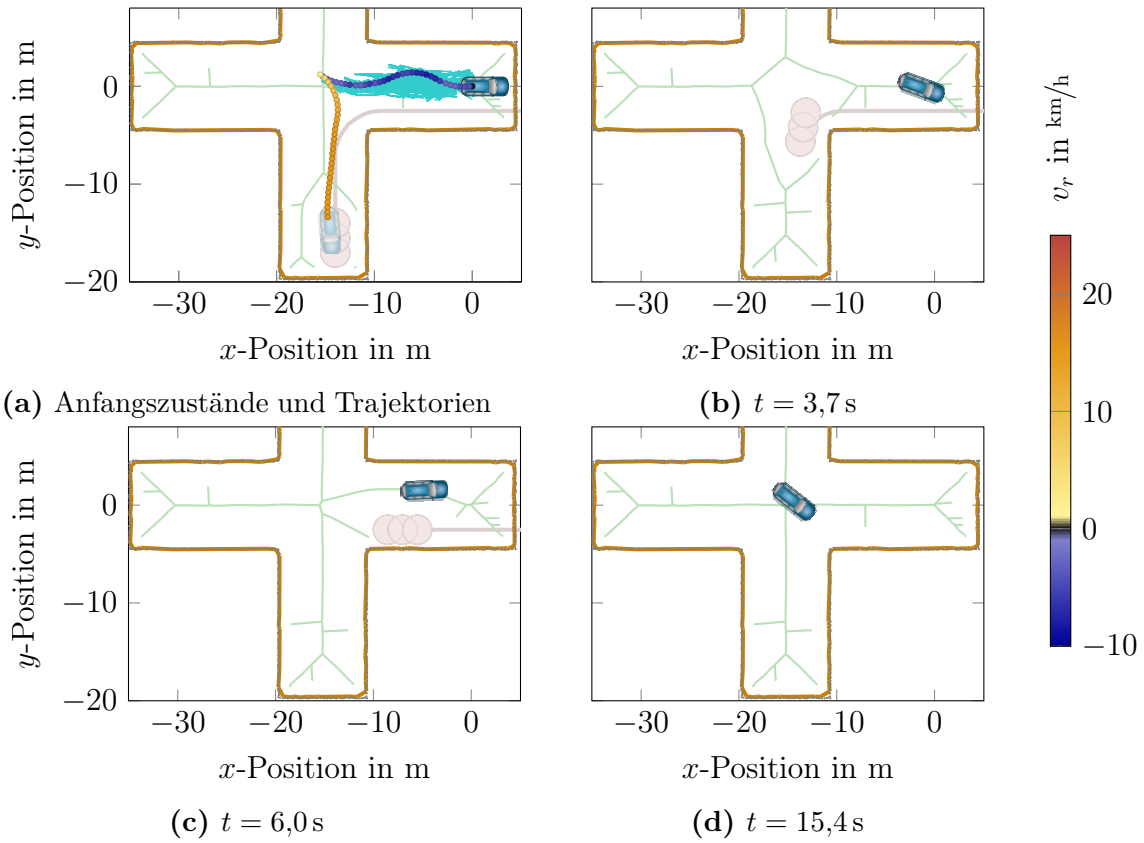
Abbildung 4.4: Exemplarische Lösung von Algorithmus 3.4 für dynamisches Abbiegen an Kreuzungen. Oben links werden die Startkonfigurationen und Trajektorien des Ego-Fahrzeugs sowie der beweglichen Objekte gezeigt. Die weiteren Plots stellen die geplanten beziehungsweise prädizierten Zustände aller Beteiligten für repräsentative Zeitpunkte dar.

4.1.1.2 Dynamische Probleme

Die Komplexität von Szenarien kann nicht nur durch schwer erreichbare Zielzustände, sondern auch durch anspruchsvolle Dynamiken anderer Verkehrsteilnehmer erhöht werden. Dies wird exemplarisch für ein Kreuzungsszenario in Abbildung 4.4 gezeigt, welches eine Erweiterung des Beispiels zum Linksabbiegen aus Abbildung 4.1b um bewegliche Komponenten darstellt. Dies beinhaltet ein größeres, entgegenkommendes Objekt (zum Beispiel ein Auto) sowie zwei Kleinere (zum Beispiel Menschen), welche die Straßen kreuzen. Die Lösung des gezeigten Problems muss die Konfigurationen aller Verkehrsteilnehmer zu allen betrachteten Zeiten berücksichtigen; dies führt insbesondere am Anfang und direkt im Anschluss an die Drehung nach rechts zu vergleichsweise großem Suchaufwand, wie anhand der Expansions-

pfade in Abbildung 4.4a zu erkennen ist. Dies ist konsistent mit den zugehörigen Konfigurationen der beweglichen Objekte, welche in den Abbildungen 4.4b bis 4.4d gezeigt werden. Insbesondere für die genannten Stellen ist an den zugehörigen Zeiten wenig Platz zur Navigation verfügbar, sodass dem Suchverfahren nur eine geringe Anzahl zulässiger Knoten zur Verfügung stehen und viele unzulässige Pfade verworfen werden müssen. Das Finden von konfliktlosen Zuständen wird durch das verallgemeinerte Voronoi-Potential unterstützt. Dieses basiert auf den entsprechenden dynamischen Voronoi-Pfaden, welche für die gewählten Zeitpunkte ebenfalls graphisch dargestellt sind. Allgemein müssen diese für alle diskreten Zeitpunkte berechnet werden, in denen sich andere bewegliche Objekte innerhalb des Freibereichspolygons befinden. Wie Tabelle 4.1 zeigt, betrifft dies insgesamt 70 Zeitpunkte, was in einem (verglichen mit den vorigen Szenarien) großen Zeitaufwand von 39,1 ms resultiert und damit ungefähr der doppelten Rechenzeit zur Generierung der Trajektorie entspricht. Der Rechenaufwand für die Erstellung des Freibereichspolygons und des Zielzustandes ändert sich im Vergleich zum statischen Szenario nicht wesentlich. Damit kann die Lösung des gezeigten Problems insgesamt in ca. 62 ms und damit – insbesondere mit Blick auf dessen Komplexität – effizient von Algorithmus 3.4 gefunden werden.

Zuletzt wird mit Abbildung 4.5 eine Situation diskutiert, die für den generischen Hybrid A*-Algorithmus, verglichen mit den vorigen Szenarien, deutlich schwerer zu lösen ist. In dieser wird dasselbe Kreuzungsszenario wie zuvor als Grundlage genommen, allerdings startet das Ego-Fahrzeug in diesem Fall aus dem Stand und muss rückwärts setzen, um in die gewünschte Richtung auf die Zielgeschwindigkeit von 15 $^{\mathrm{km}}/_{\mathrm{h}}$ zu beschleunigen. Erschwert wird das Manöver durch ein größeres, abbiegendes Objekt. Wie anhand der Expansionspfade in Abbildung 4.5a gut zu erkennen ist, führt die Berücksichtigung von dessen Dynamik unter anderem in der ersten, rückwärts gerichteten Phase des Szenarios zu einem hohen Suchaufwand. Dies ergibt sich daraus, dass aus dem Stand zunächst eine Bahn gefunden werden muss, mit der eine Kollision vermieden wird. Zudem muss auch hier ein passender Umschaltpunkt bezüglich der Fahrtrichtung gefunden werden, um schließlich vorwärts auf den Zielzustand zusteuern zu können. Dies resultiert in einer zusätzlichen Komplexität, wie zuvor beim Parkmanöver aus Abbildung 4.2b bereits beobachtet werden konnte. Ist die Expansion des Hybrid A*-Algorithmus im entsprechenden unteren Arm der Kreuzung angekommen, kann die zweite, vorwärts gerichtete Phase des Problems sehr schnell gelöst werden. Insgesamt benötigt die Lösung dieser Aufgabe jedoch über 10 000 expandierte Knoten, was im Vergleich zum bisher aufwändigsten Manöver aus Abbildung 4.4 fast eine Verfünffachung darstellt. Dies drückt sich direkt in der benötigten Rechenzeit von ca. 100 ms aus. Auch wenn die Generierung der dynamischen Voronoi-Pfade hier nicht so aufwändig wie zuvor ist (aufgrund von einer geringeren Menge an diskreten Zeiten mit dynamischen Objekten), so liegt die benötigte Gesamtzeit mit 122,5 ms trotzdem über dem vorgegebenen Zeithorizont von 100 ms für die taktische Entscheidungsfindung.

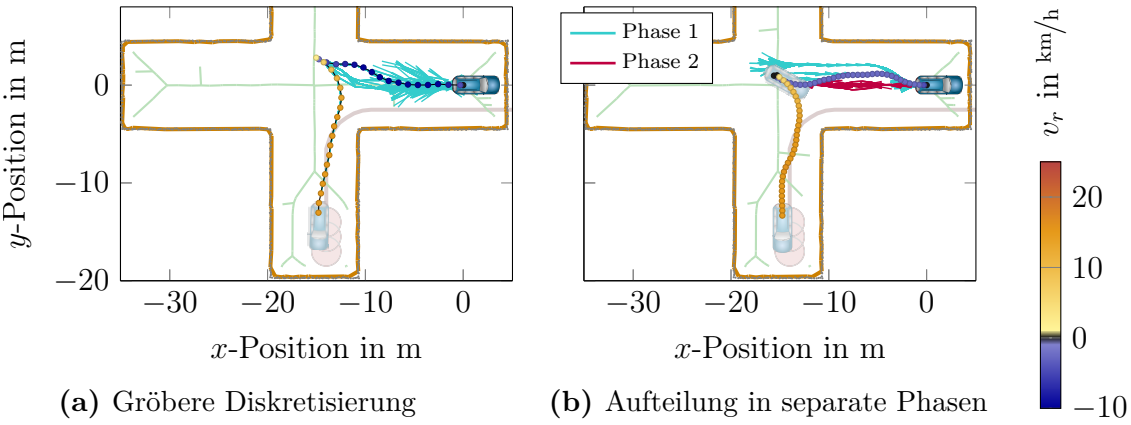


(a) Anfangszustände und Trajektorien **(b)** $t = 3{,}7\,\mathrm{s}$

(c) $t = 6{,}0\,\mathrm{s}$ **(d)** $t = 15{,}4\,\mathrm{s}$

Abbildung 4.5: Exemplarische Lösung von Algorithmus 3.4 für dynamisches, zweizügiges Abbiegen an Kreuzungen

Abbildung 4.6 zeigt abschließend zwei Ansätze zur Kompensation des hohen Aufwands für dieses Beispiel. Ganz allgemein kann die Anzahl der expandierten Knoten drastisch verringert werden, indem die Genauigkeit der Lösung reduziert wird. In Abbildung 4.6a wird dies durch eine Verdopplung sowohl der Orts- als auch der Winkeldiskretisierung erreicht. Die resultierenden Trajektorien unterscheiden sich jeweils lediglich am Anfang, es müssen jedoch nur noch ca. 1500 Knoten expandiert werden. Damit benötigt der Suchalgorithmus selber knapp unter 40 ms, was zu einer Gesamtzeit von ungefähr 55 ms führt.
Insbesondere für mehrzügige Manöver bietet sich der Ansatz aus Abbildung 4.6b an. Hier wird angenommen, dass der Umschaltpunkt in der Mitte der Kreuzung bekannt ist (zum Beispiel aus vorigen Berechnungen). In diesem Fall können beide Phasen separat berechnet werden, wobei der Stillstand am Umschaltpunkt entsprechend als End- beziehungsweise Startzustand vorgegeben wird. Damit entfällt ein Teil der Komplexität der Aufgabe: die Gesamtzahl der expandierten Knoten reduziert sich auf ca. 3000 und die Gesamtrechenzeit auf 60 ms. Der Zeitaufwand ist damit vergleichbar zur Strategie aus Abbildung 4.6a, für die resultierende Trajektorie ist mit diesem Vorgehen jedoch kein Genauigkeitsverlust verbunden. Beide Ansätze ver-



(a) Gröbere Diskretisierung **(b)** Aufteilung in separate Phasen

Abbildung 4.6: Alternative Ansätze zur Lösung des dynamischen, zweizügigen Abbiegeszenarios aus Abbildung 4.5. Das linke Bild zeigt die resultierende Trajektorie, wenn die Orts- und Winkelauflösung δ_p und δ_φ jeweils verdoppelt wird. In der rechten Grafik werden die beiden Phasen des Problems separat gelöst, wobei als Endpunkt der ersten, beziehungsweise Anfangspunkt der zweiten Phase der Umschaltpunkt aus Abbildung 4.5 verwendet wird.

bleiben damit im Zeithorizont der taktischen Entscheidungsfindung und können in deren Kontext zum Beispiel eingesetzt werden, nachdem ein Problem zuvor nur mit stark erhöhter Rechenzeit gelöst werden konnte. Auch wenn sich für die nächste Aktualisierung der taktischen Planung (also nach ungefähr 100 ms) Startzustände und Umgebungsbedingungen leicht verändert haben, so können trotzdem analoge Laufzeitverbesserungen durch die hier gezeigten Verfahren erwartet werden.

4.1.2 Effiziente Distanzberechnungen

Zur abschließenden Optimierung der Trajektorie in Zeile 5 von Algorithmus 3.4 müssen für jeden explorierten Knoten eine Reihe von Distanzauswertungen durchgeführt werden: Im Rahmen der Berechnung des verallgemeinerten Voronoi-Potentials Φ_V^* und der damit zusammenhängenden Kollisionsüberprüfung betrifft dies Abstände des Ego-Fahrzeugs zum statischen bzw. entsprechenden dynamischen Voronoi-Pfad, zu relevanten beweglichen Objekten sowie zum Freibereichspolygon, vergleiche Abschnitt 3.3.4. Für die Berechnung von Kosten und Heuristiken gemäß Abschnitt 3.4 wird zudem jeweils eine Schätzung der verbleibenden Distanz $\mathrm{d}_\odot$ zur Zielkonfiguration mit der Methode von Dubins beziehungsweise Reeds und Shepp benötigt. Alle genannten Auswertungen haben gemein, dass sie sich jeweils auf den diskretisierten Zustand des Fahrzeuges beziehen.

Für die Distanzberechnungen im Rahmen des Voronoi-Potentials bedeutet dies, dass bereits berechnete Ergebnisse wiederverwendet werden können, falls mehrmals die gleichen diskreten Komponenten durch die Knotenexpansion erreicht werden. Für die

Tabelle 4.2: Rechenzeiten für die Berechnung einer Trajektorie durch Zeile 5 von Algorithmus 3.4, abhängig vom Memorieren von Distanzen diskreter Knoten zum Voronoi-Pfad, den beweglichen Objekten oder dem Freibereichspolygon sowie bei Vorberechnung der Distanzheuristik. ✓ – Memorieren/Vorberechnung wird angewendet. ✗ – Memorieren/Vorberechnung wird nicht angewendet. ✗ – Anwendung des Memorierens hat in diesem Szenario keinen Einfluss.

Szenario	Distanz-Memorieren von: Stat. Voronoi-Pfad	Dyn. Voronoi-Pfad	Dyn. Objekte	Polygon	Vorberechnung: Distanzheuristik	Trajektorie: Rechenzeit	Exp. Knoten
Statisches Abbiegen Abbildung 4.1d	✗	✗	✗	✗	✗	41,1 ms	1115
	✓	✗	✗	✗	✗	38,6 ms	1115
	✗	✗	✗	✓	✗	26,5 ms	1115
	✓	✗	✗	✓	✗	24,0 ms	1115
	✓	✗	✗	✓	✓	**8,5 ms**	1123
Dynamisches Abbiegen Abbildung 4.4	✗	✗	✗	✗	✗	198,2 ms	4225
	✗	✓	✗	✗	✗	185,4 ms	4225
	✗	✗	✓	✗	✗	191,5 ms	4225
	✗	✗	✗	✓	✗	122,4 ms	4225
	✗	✓	✓	✓	✗	99,3 ms	4225
	✗	✓	✓	✓	✓	**19,4 ms**	2187

Kollisionsüberprüfung mit dynamischen Objekten und einem Freibereichspolygon bezieht sich dies auf das Tupel aus Position und Orientierung. Die Berechnung des Abstandes zum Voronoi-Pfad hängt dagegen nur vom Referenzpunkt des Fahrzeuges ab, sodass hier sogar identische diskrete Positionen zur Wiederverwendung voriger Ergebnisse ausreichen. Das Abspeichern und Wiederverwenden bereits berechneter Werte wird im weiteren Verlauf als *Memorieren* bezeichnet. Auf numerischer Ebene wird dazu eine Hashtabelle verwendet, vergleiche auch Abschnitt 2.4.2.

Tabelle 4.2 zeigt anhand von jeweils einem statischen und dynamischen Beispiel aus Abschnitt 4.1.1 den Einfluss dieses Memorierens von Distanzen auf die Berechnungszeit der Trajektorie. Im Falle des statischen Abbiegens aus Abbildung 4.1d ergibt sich, dass sowohl die Wiederverwendung der Abstände zum statischen Voronoi-Pfad, als auch zum Freibereichspolygon jeweils einen positiven Einfluss auf die Verfahrensdauer haben. Während ohne ein Memorieren 41,1 ms benötigt werden, können basierend auf dem Voronoi-Pfad ca. 2,5 ms und auf Grundlage des Freibereichspolygons sogar 24,6 ms eingespart werden. Sind beide Ansätze berücksichtigt, addiert sich der Vorteil entsprechend. Für das dynamische Manöver aus Abbildung 4.4 ergibt sich ein verhältnismäßig ähnlich großer Vorteil beim Memorieren der Distanzen zum Freibereichspolygon, womit die Rechenzeit von 198,2 ms auf 122,4 ms reduziert werden kann. Da sich in diesem Szenario in allen Zeitschritten dynamische Objekte innerhalb des Freibereichspolygons befinden, kann hier zusätzlich eine Effizienzsteigerung durch die Wiederverwertung von Distanzen zum dynamischen Voronoi-Pfad, als auch zu den dynamischen Objekten selbst erreicht werden. Dabei führt insbesondere Ersteres zu einer verhältnismäßig deutlichen Verbesserung. Werden alle drei Ansätze parallel benutzt, kann in diesem Fall bereits eine Halbierung der ursprünglichen Rechenzeit erreicht werden.

Um die in Tabelle 4.1 gezeigten Ergebnisse zu erreichen, muss zusätzlich eine Vorberechnung der Funktion zur Distanzschätzung erfolgen. Dafür werden die Pfadlängen basierend auf dem Ansatz von Dubins beziehungsweise Reeds und Shepp, ausgehend vom Koordinatenursprung, vor der eigentlichen Trajektorienoptimierung für alle diskreten Konfigurationen bis zu einer maximalen Entfernung Δ_p ausgewertet und in einem dreidimensionalen Tensor abgelegt. Der Speicher- und Zeitaufwand für diesen Prozess ist für verschiedene Diskretisierungskonfigurationen in Tabelle 4.3 dargestellt. Für die Referenzparameter aus Tabelle A.1 benötigt dieser Ansatz 13,1 MB Speicherplatz und – je nach Vorgehen – zwischen 50,2 ms und 2000,8 ms. Dabei ist die Berechnung der einfacheren Pfade von Dubins prinzipiell schneller. Zudem kann deutlich an Zeit gespart werden, wenn die Berechnung parallelisiert erfolgt. Aufgrund der Dreidimensionalität führt eine Veränderung in der Diskretisierung oder maximalen Entfernung jeweils zu einem starken Einfluss auf den benötigten Aufwand, sodass bei diesem Vorgehen die verfügbaren Ressourcen beachtet werden müssen. Da die Auswertung der Distanzschätzung mit Dubins oder Reeds und Shepp jedoch ohne Berücksichtigung von Hindernissen (und damit unabhängig vom konkreten Szenario) vorgenommen wird, kann dieser Schritt in einer zeitlich unkritischen Initialisierungs-

Tabelle 4.3: Anzahl der Elemente, benötigter Speicherplatz und Zeitaufwand für die diskretisierte Vorberechnung von Dubins- beziehungsweise Reeds-Shepp-Pfaden für verschiedene Diskretisierungen und Reichweiten sowie abhängig davon ob sequentiell auf einem oder parallel auf 16 Prozessoren gerechnet wird

Konfig.	**Elemente**	**Speicher in MB**	**Rechenzeit in ms**			
			Dubins		**Reeds-Shepp**	
			1	16	1	16
$\delta_p = 0{,}5\,\mathrm{m}$ $\delta_\varphi = 0{,}1\,\mathrm{rad}$ $\Delta_p = 40\,\mathrm{m}$	1 633 023	13,1	342,8	50,2	2000,8	335,3
$\delta_p = 1{,}0\,\mathrm{m}$ $\delta_\varphi = 0{,}2\,\mathrm{rad}$ $\Delta_p = 40\,\mathrm{m}$	203 391	1,6	43,6	6,7	258,0	44,7
$\delta_p = 0{,}5\,\mathrm{m}$ $\delta_\varphi = 0{,}1\,\mathrm{rad}$ $\Delta_p = 100\,\mathrm{m}$	10 130 463	81,0	1989,2	292,6	11 578,7	2014,6

phase eines Programms ablaufen, in welcher in der Regel auch die Kapazitäten für eine parallele Berechnung verfügbar sind.

Der vorberechnete Tensor kann schließlich genutzt werden, um für einen gegebenen Knoten die verbleibende Distanzschätzung zum Ziel $\mathrm{d}_\odot$ zu erhalten. Da die Entfernungen innerhalb dieser Initialisierung ausgehend vom Ursprung bestimmt wurden, muss die diskrete Zielkonfiguration in relative Koordinaten $\lceil z^{(\mathrm{end})} \rfloor_{\mathrm{rel}}$ der diskreten Komponente des aktuell betrachteten Knotens transformiert werden. Diese Konvertierung erhält jedoch nicht zwingend die Diskretisierung, es kann also $\lceil \lceil z^{(\mathrm{end})} \rfloor_{\mathrm{rel}} \rfloor \neq \lceil z^{(\mathrm{end})} \rfloor_{\mathrm{rel}}$ gelten. Folgerichtig muss zum Zugriff auf den Tensor nachdiskretisiert und damit ein entsprechender Fehler im Wert der Heuristik akzeptiert werden. Würde der vorberechnete Tensor nicht nur vom Koordinatenursprung, sondern von allen möglichen diskreten Konfigurationen aus erstellt werden, könnte die Auswertung ohne Approximationsungenauigkeit erfolgen. Allerdings würde dies ebenfalls zu einer Quadrierung der benötigten Elemente und damit, in Bezug auf die Diskretisierung aus Tabelle A.1, zu einem Speicheraufwand von 21,4 TB führen sowie – im parallelen Ansatz für Dubins Pfade – voraussichtlich ca. 23 h benötigen. Da die verbleibende Distanzschätzung zum Ziel $\mathrm{d}_\odot$ nur für die Heuristik (nicht aber für die Kosten) benötigt wird, hat ein zusätzlicher Diskretisierungsfehler entsprechend jedoch auch nur Einfluss auf den Verlauf der A*-Suchiteration, vergleiche Abschnitt 2.4.1, und führt nicht zur einer Veränderung der optimalen Lösung selbst.

Hinzu kommt eine deutliche Verbesserung im Blick auf die Rechenzeit, wie Tabelle 4.2 zeigt. Für das statische Abbiegen ergibt sich demnach eine weitere Reduktion auf 8,5 ms, was 35 % des vorigen Ergebnisses entspricht. Durch den Diskretisierungsfehler müssen dabei lediglich marginal mehr Knoten expandiert werden. Für das dynamische Abbiegen ergibt sich durch die vorberechnete Heuristik sogar eine fast halbierte relative Restlaufzeit von 20 % (in Bezug auf das vorige Ergebnis), was unmittelbar mit der annähernd halbierten Anzahl der expandierten Knoten in diesem Fall zusammenhängt.

Insgesamt zeigt dieser Abschnitt, dass die benötigten Distanzberechnungen für Kollisionsüberprüfungen, Voronoi-Potential und Heuristik einen Großteil der nominellen Zeit zur Optimierung der Trajektorien durch den Hybrid A*-Algorithmus einnehmen. Durch gezielte Verfahrensverbesserungen kann dies deutlich reduziert werden: in den gezeigten Beispielen auf jeweils insgesamt nur noch 20,1 % im statischen beziehungsweise 9,8 % im dynamischen Fall.

4.1.3 Planung mit dem verallgemeinerten Voronoi-Potential

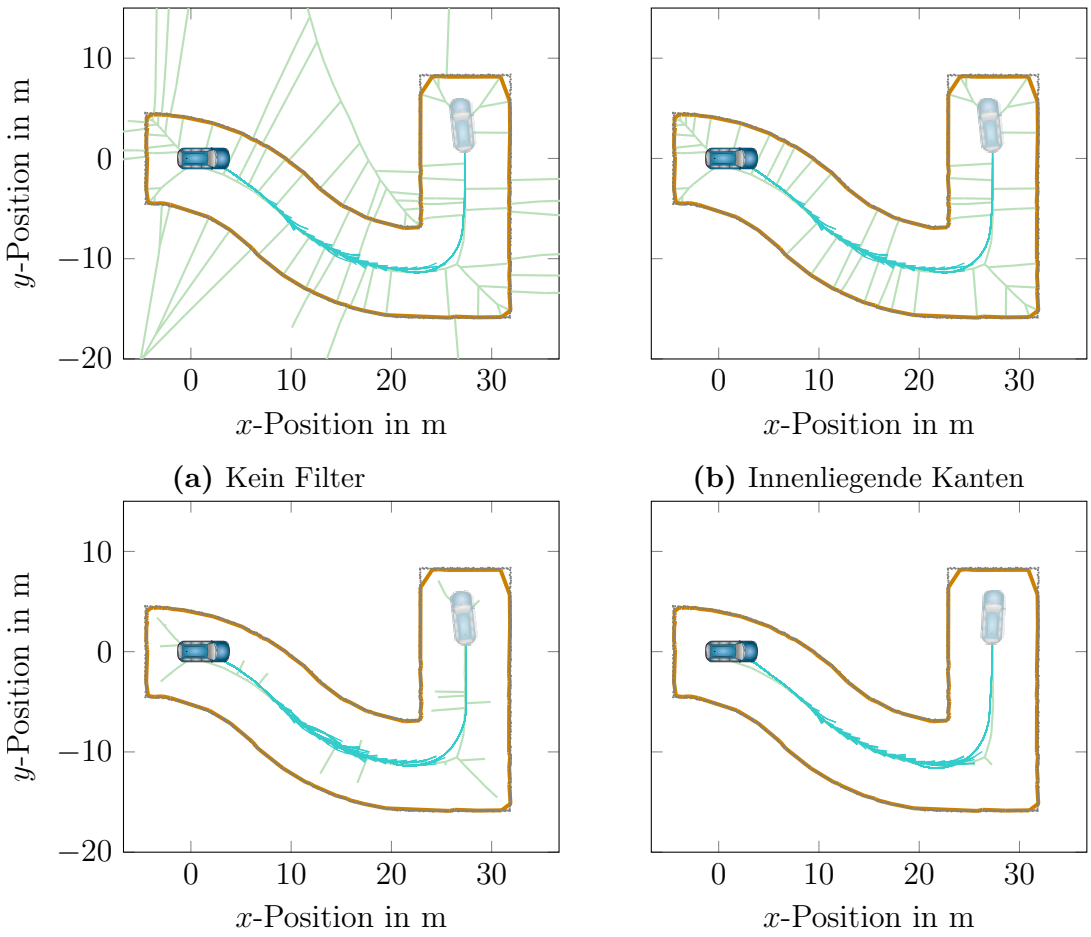
Ein wesentlicher Bestandteil des generischen Hybrid A*-Algorithmus 3.4 ist das in dieser Arbeit eingeführte verallgemeinerte Voronoi-Potential. Dieses basiert im Kern auf einem segmentbasierten Voronoi-Diagramm der räumlichen Beschränkungen und einer anschließenden Filterung von dessen Kanten. Im Rahmen der Beschreibung dieses Prozesses in Abschnitt 3.3 sind dabei zwei Fragen offen geblieben:

1. Welchen Einfluss hat der Filterprozess auf das Ergebnis der Planung (vergleiche Abschnitt 3.3.2)?
2. Welchen qualitativen Beitrag leistet die zeitliche Investition zur Berechnung der (dynamischen) Voronoi-Kanten (vergleiche Abschnitt 3.3.4)?

Im Folgenden werden beide Fragen durch exemplarische numerische Ergebnisse beantwortet.

4.1.3.1 Einfluss der Voronoi-Pfad Filterung

Der in Abschnitt 3.3.2 beschriebene Filterprozess für die Kanten des Voronoi-Diagramms hat zum Ziel, möglichst alle in Richtung des Freibereichspolygons führenden Elemente zu entfernen. Idealerweise bleiben anschließend lediglich die parallel zwischen den Segmenten des Freibereichspolygons liegenden Voronoi-Kanten zurück. Dieses Ziel ist in der Praxis jedoch nicht erreichbar, da der Filterprozess auf einem Schwellenwertprinzip basiert und es damit in der Regel nicht möglich ist, für beliebige Situationen stets nur die exakt richtigen Kanten zu filtern. Im Folgenden



(a) Kein Filter

(b) Innenliegende Kanten

(c) Standard (vergleiche Abbildung 4.1d)

(d) Individueller Filter

Abbildung 4.7: Statischer Voronoi-Pfad für verschiedene Filter in der Situation aus Abbildung 4.1d. Gezeigt werden die resultierende Zielkonfiguration sowie die A*-Expansion durch Algorithmus 3.4.

wird an einem exemplarischen Beispiel kurz diskutiert, welchen konkreten Einfluss die Genauigkeit des Filterns auf die Planung mit dem Hybrid A*-Algorithmus hat.

Abbildung 4.7 zeigt den Suchverlauf der Trajektorienoptimierung von Algorithmus 3.4 für verschiedene Filterstufen des Voronoi-Pfades in der Situation aus Abbildung 4.1d. Betrachtet werden die Ergebnisse für das Auslassen des Filterprozesses, nur Entfernen der äußeren Kanten, das Schwellenwertverfahren aus Definition 3.18 mit dem Parameter aus Tabelle A.1 sowie einem auf das betrachtete Szenario angepasstes, *individuelles* Filtern. Die gezeigten Expansionen sehen dabei für alle Beispiele sehr ähnlich aus. Insbesondere im direkten Vergleich von Abbildung 4.7d mit Abbildungen 4.7a und 4.7b ist optisch kein Einfluss durch die zur Fahrrichtung quer liegenden Voronoi-Kanten erkennbar. Dies bestätigt die Vermutung aus Abschnitt 3.3.2, dass diese Referenzpfade aufgrund der zu berücksichtigenden Fahrdy-

Tabelle 4.4: Einfluss der Filtermethode für den statischen Voronoi-Pfad auf die Lösung des Szenarios aus Abbildung 4.7. Die Approximation der Punktwolke mit dem Freibereichspolygon entspricht in allen Fällen den Ergebnissen aus Tabelle 4.1.

		Kein Filter	**Innenliegend**	**Standard**	**Individuell**
	Abbildung	4.7a	4.7b	4.7c	4.7d
# Elem.	Voronoi-Pfad	289	163	106	62
	Trajektorie	85	85	85	85
A*	Geöffnete Knoten	9513	9513	10 562	9614
	Expandierte Knoten	1011	1011	1123	1026
Zeit in ms	Voronoi-Pfade	1,5	1,8	1,8	1,8
	Zielzustand	48,3	0,5	0,2	0,07
	Trajektorie	7,8	7,8	8,5	8,0

namik nicht gut erreichbar sind und damit kaum innerhalb der Suchiteration wahrgenommen werden.

Tabelle 4.4 stellt die relevanten Kennzahlen der gezeigten Szenarien dar und erlaubt damit einen noch differenzierteren Blick auf deren Unterschiede. Mit stärkerer Filterung ergibt sich erwartungsgemäß eine absteigende Anzahl an Elementen im Voronoi-Pfad. Gleichzeitig ist für alle Probleme die Länge der resultierenden Trajektorie identisch und der Expansionsaufwand sehr ähnlich. Kleine Unterschiede sind hier vor allem auf numerische Effekte zurückzuführen. Insgesamt unterstreichen die Werte damit den Eindruck der vorigen optischen Auswertung. Ein relevanter Effekt ergibt sich jedoch bei Betrachtung der Rechenzeiten. Wird der zusätzliche Filterschritt des Voronoi-Pfades weggelassen, so kann dessen Erstellung entsprechend leicht beschleunigt werden. Durch die große Menge an Voronoi-Kanten außerhalb des Freibereichspolygons ergibt sich in diesem Beispiel dann jedoch eine sehr große Aufwandssteigerung bei der automatischen Erstellung eines Zielzustandes; mit 48,3 ms benötigt sie fast 250 mal so viel Zeit wie bei der Standardfilterung. Im Falle einer individuellen Filterung ist dagegen kaum Suchaufwand erforderlich, sodass hier lediglich 0,07 ms gebraucht werden. Die Rechenzeiten zur Erstellung der Trajektorie selbst variieren nur leicht und stehen in direktem Zusammenhang mit den Unterschieden im A*-Expansionsaufwand.

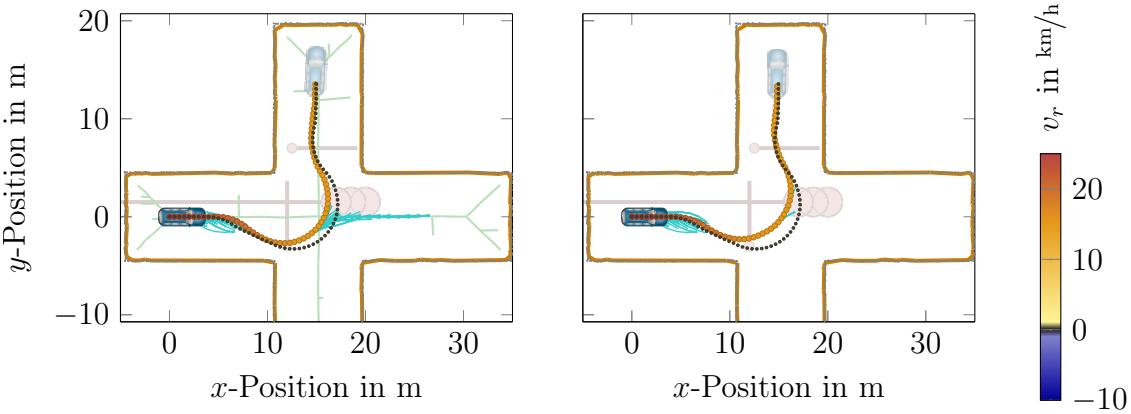
Insgesamt zeigt dieser Abschnitt, dass ein moderates Filtern einen deutlich positiven Effekt hat, da dies insbesondere mit Blick auf die Berechnung der Zielkonfiguration große Vorteile in Bezug auf die Rechenzeit ergibt. Der Schwellenwert sollte dabei jedoch so gewählt werden, dass keine wichtigen innenliegenden Kanten entfernt werden. Dies würde nicht nur das Voronoi-Potentialfeld selbst schwächen, sondern auch bewirken, dass Teilpfade zu ansonsten guten Zielkonfigurationen nicht mehr existieren, vergleiche Abschnitt 3.3.6. Wenn keine genaueren Informationen über die Umgebung vorliegen, kann die halbe Breite des betrachteten Fahrzeuges als gute Referenzgröße für den Filter verwendet werden. Dies garantiert, dass nur Kanten entfernt werden, die zu nicht befahrbaren, engen Passagen gehören.

4.1.3.2 Einfluss der Potentialfunktion

Die simulativen Auswertungen in Abschnitt 4.1.1 zeigen, dass die Berechnung der statischen und dynamischen Voronoi-Pfade als Grundlage des Voronoi-Potentials einen Vergleichsweise großen Aufwand bedeutet. Für sehr einfache statische Probleme wie diejenigen aus den Abbildungen 4.1a bis 4.1c kann sie in der Größenordnung der Trajektorienoptimierung selbst liegen. Im Beispiel aus Abbildung 4.4 ergibt sich aufgrund der feinen Zeitdiskretisierung (siehe Tabelle A.1) und der daraus resultierenden großen Menge von diskreten Zeiten mit beweglichen Objekten sogar eine Berechnungszeit von fast 40 ms. Mit Blick auf die zur Verfügung stehende Gesamtzeit von 100 ms für die taktische Entscheidungsfindung kann der statische Voronoi-Pfad jedoch insgesamt vergleichsweise schnell erstellt werden und hat zusätzlich zum Beispiel auch einen direkten Nutzen zur Identifikation von Zielkonfigurationen (vergleiche Abschnitt 3.3.6). Im Gegensatz dazu erfolgt die wesentlich aufwändigere Berechnung der dynamischen Pfade einzig zur Verwendung im Voronoi-Potential.

Dieser Abschnitt stellt auf Grundlage von zwei Beispielen kurz den Einfluss des zeitabhängigen Voronoi-Potentials auf den Suchverlauf sowie die resultierenden Lösungen dar. Als Vergleichsbasis werden dazu zwei alternative Potentialfelder durch Modifikation der verallgemeinerten Voronoi-Potentials aus Abschnitt 3.3.4 definiert. Die entsprechenden Änderungen sind jeweils textlich hervorgehoben.

1. Das *statische* verallgemeinerte Voronoi-Potential $\Phi_V^{\diamond} \colon \Omega^\circ \times \mathbb{R} \to \mathbb{R}_{\geq 0}$.
   Bei diesem wird nur der statische Voronoi-Pfad, nicht aber die dynamischen Pfade berücksichtigt. Es entspricht ansonsten der verallgemeinerten Voronoi-Potentialfunktion aus Abschnitt 3.3.4 – insbesondere werden auch hier Abstände zu dynamischen Objekten weiter in die Berechnung einbezogen. Die



(a) Statisches Voronoi-Potential $\Phi_V^{\diamond}$ **(b)** Räumliches Potential Φ_D^{*}

Abbildung 4.8: Lösungen von Algorithmus 3.4 für das dynamische links Abbiegen basierend auf den Potentialfunktionen $\Phi_V^{\diamond}$ und Φ_D^{*}. Zum Vergleich ist die Lösung mit dem verallgemeinerten Potential Φ_V^{*} aus Abbildung 4.4 jeweils in schwarz eingezeichnet.

entsprechende Potentialfunktion ergibt sich durch

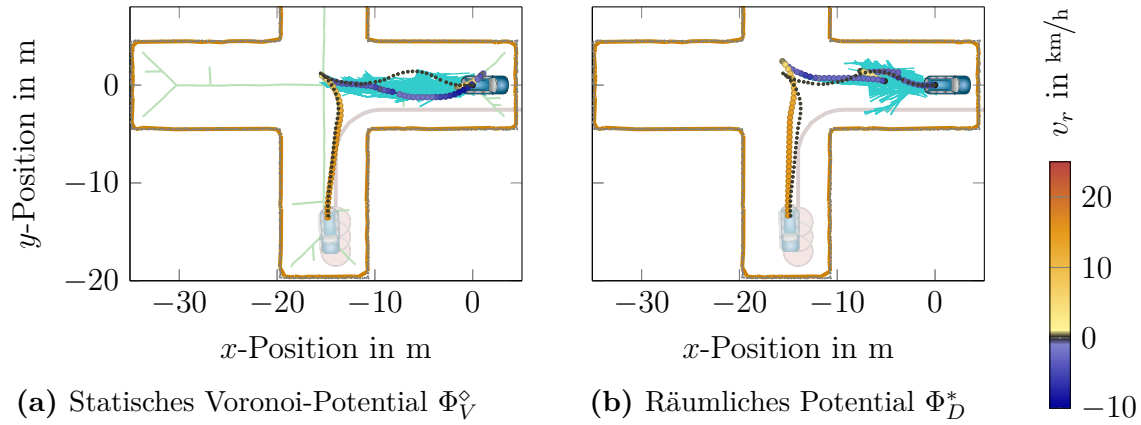
$$\Phi_V^{\diamond} \colon \Omega^{\circ} \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\text{Ego}}^{\circ},\, t) \mapsto \begin{cases} 0, & \text{falls } \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \gamma,\, \Psi^{\square}(t)) \geq \mathrm{d}^{\max}, \\ 1, & \text{falls } \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \Psi^{\square}(t)) \leq 0, \\ \overline{\Phi}_V^{\diamond}(\psi_{\text{Ego}}^{\circ},\, t), & \text{sonst}, \end{cases}$$

mit

$$\overline{\Phi}_V^{\diamond} \colon \Omega^{\circ} \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\text{Ego}}^{\circ},\, t) \mapsto \frac{\alpha}{\alpha + \mathrm{d}^{+}(\psi_{\text{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t))} \cdot \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t),\, \mathcal{R}^{*}) \cdot \frac{\mathrm{d}^{\max} - \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t))}{\mathrm{d}^{\max}}.$$

2. Das Potential der räumlichen Beschränkungen $\Phi_D^{*} \colon \Omega^{\circ} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$.
   Es entspricht der verallgemeinerten Voronoi-Potentialfunktion aus Abschnitt 3.3.4, allerdings ohne Berücksichtigung jeglicher Voronoi-Pfade:

$$\Phi_D^{*} \colon \Omega^{\circ} \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\text{Ego}}^{\circ},\, t) \mapsto \begin{cases} 0, & \text{falls } \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t)) \geq \mathrm{d}^{\max}, \\ 1, & \text{falls } \mathrm{d}(\psi_{\text{Ego}}^{\circ},\, \Psi^{\circ}(t)) \leq 0, \\ \overline{\Phi}_D^{*}(\psi_{\text{Ego}}^{\circ},\, t), & \text{sonst}, \end{cases}$$



(a) Statisches Voronoi-Potential $\Phi_V^{\diamond}$ **(b)** Räumliches Potential Φ_D^{*}

Abbildung 4.9: Lösungen von Algorithmus 3.4 für das dynamische, mehrzügige Abbiegen basierend auf den Potentialfunktionen $\Phi_V^{\diamond}$ und Φ_D^{*}. Zum Vergleich ist die Lösung mit dem verallgemeinerten Potential Φ_V^{*} aus Abbildung 4.5 jeweils in schwarz eingezeichnet.

mit

$$\overline{\Phi}_D^{*}\colon \Omega^{\circ} \times \mathbb{R} \to \mathbb{R}_{\geq 0},$$
$$(\psi_{\mathrm{Ego}}^{\circ},\, t) \mapsto \frac{\alpha}{\alpha + \mathrm{d}^{+}(\psi_{\mathrm{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t))} \cdot 1 \cdot \frac{\mathrm{d}^{\max} - \mathrm{d}(\psi_{\mathrm{Ego}}^{\circ},\, \gamma,\, \Psi^{\circ}(t))}{\mathrm{d}^{\max}}.$$

Beide Varianten stellen im Vergleich zum verallgemeinerten Voronoi-Potential Φ_V^{*} eine Vereinfachung dar, da weniger Aufwand sowohl in die Berechnung der Voronoi-Pfade als auch in die Auswertung dieser im Rahmen der Suchiteration investiert werden muss. Gleichzeitig wird die Potentialfunktion selbst in ihrer Funktion als Referenz für *gute* Konfigurationen des Ego-Fahrzeuges geschwächt. Wird im Fall von Φ_D^{*} nicht einmal der statische Voronoi-Pfad berechnet, entfällt zudem die Möglichkeit, Zielkonfigurationen automatisiert auf deren Grundlage zu bestimmen.

Die Auswirkung dieser alternativen Ansätze wird anhand der beiden Fallbeispiele zum Abbiegen mit dynamischen Hindernissen aus den vorigen Abschnitten in den Abbildungen 4.8 und 4.9 sowie durch Tabelle 4.5 exemplarisch dargestellt. Um sinnvolle Rahmenbedingungen zum Vergleichen der Lösungen zu schaffen, wird die Zielkonfiguration für Φ_D^{*} dabei jeweils aus dem Ansatz mit $\Phi_V^{\diamond}$ übernommen. Werden die beiden Trajektorien aus Abbildung 4.8 mit derjenigen aus Abbildung 4.4 verglichen, sind nur kleine Unterschiede während der Kurvenfahrt in der Mitte der Kreuzung erkennbar: Je detaillierter die Referenz durch die (dynamischen) Voronoi-Pfade vorliegt, desto größer ist das Ausholen zum Ausweichen der anderen beweglichen Objekte und entsprechend länger die resultierende Lösung. Deutlichere Diskrepanzen zwischen den Ansätzen liegen dagegen beim Suchverlauf vor. Während alle Verfahren am Anfang einen erhöhten Suchaufwand benötigen, um dem entgegenkommenden Fahrzeug Platz zu machen, entstehen nur bei den Varianten basierend auf einem

Tabelle 4.5: Einfluss unterschiedlicher Potentialfunktionen auf die Lösungen der Szenarien aus den Abbildungen 4.4 und 4.5 sowie 4.8 und 4.9. Zum qualitativen Vergleich der Trajektorien werden ebenfalls die Anzahl der Richtungswechsel sowie die Länge als Prozesszeit angegeben.

		Einfaches Abbiegen			**Mehrzügiges Abbiegen**		
	Potential	Φ_V^*	$\Phi_V^\diamond$	Φ_D^*	Φ_V^*	$\Phi_V^\diamond$	Φ_D^*
	Abbildung	4.4	4.8a	4.8b	4.5	4.9a	4.9b
# Elem.	Voronoi-Pfad(e)	10 930	143	–	5421	143	–
	Trajektorie	68	64	62	64	81	87
A*	Geöffnete Knoten	19 152	20 721	14 722	87 437	108 958	67 329
	Expandierte Knoten	2187	2290	1725	10 382	13 441	7725
Zeit in ms	Voronoi-Pfade	39,1	3,7	–	14,6	3,6	–
	Trajektorie	19,4	23,3	17,0	103,9	131,9	78,3
	# Richtungswechsel	0	0	0	1	3	5
	Prozessdauer in s	6,9	6,5	6,7	20,6	36,6	41,7

Voronoi-Potential zusätzliche Iterationen in der Mitte der Kreuzung. Durch den nach rechts verlaufenden Voronoi-Pfad entsteht hier jeweils eine *falsche* Referenz, die zunächst einfacher zu verfolgen (da hindernisfrei), aber nicht zielführend ist. Eine solche Fehlleitung tritt durch das Potential Φ_D^* naturgemäß nicht auf, weshalb dies sowohl weniger Iterationen benötigt als auch insgesamt zu deutlich kürzeren Rechenzeiten führt. Im Vergleich der Voronoi-Potentiale ist die Trajektorienberechnung für den Ansatz basierend auf dynamischen Voronoi-Pfaden leicht schneller, benötigt aber insgesamt mehr Zeit, wenn deren Erstellungsdauer ebenfalls berücksichtigt wird.

Generell ist die Aufgabe der Voronoi-Potentialfelder, die Güte einer großen Menge möglicher Konfigurationen zu bewerten und so die Knotenexpansion zu steuern. Im Fall des Problems aus Abbildung 4.4 gibt es jedoch, bis auf die Mitte der Kreuzung, durch die Bewegung der dynamischen Objekte nur sehr eingeschränkte Variationsmöglichkeiten in der Ausgestaltung einer zulässigen Lösung. Dadurch geht der Vorteil der aufwändigeren Varianten basierend auf den Voronoi-Pfaden teilweise verloren. Ein deutlich verändertes Bild ergibt sich jedoch am Beispiel aus Abbildung 4.5,

welches für die alternativen Potentialansätze in Abbildung 4.9 dargestellt ist. Im Unterschied zu vorher gibt es hier viele Möglichkeiten, dem abbiegenden Objekt auszuweichen. Die Lösung in Abbildung 4.9a basierend auf dem statischen Voronoi-Potential setzt dies um, indem kurz zurück- und anschließend in die Ecke vorgesetzt wird. Auf diese Weise wird das Objekt zwar passieren gelassen, es werden allerdings insgesamt zwei zusätzliche Richtungswechsel benötigt. Das Voronoi-freie Potential weicht dagegen leicht auf dem Weg hin zur Kreuzungsmitte aus. Um die Distanz zum Polygon klein zu halten, wird dabei jedoch sogar vier mal die Fahrtrichtung geändert. Dies steht im starken Kontrast zur Referenz durch das verallgemeinerte Voronoi-Potential, welches sehr direkt und ohne zusätzliche Richtungswechsel die Kreuzungsmitte erreicht. Ab diesem Punkt verhalten sich alle Ansätze ähnlich und erreichen zielgerichtet die Endkonfiguration. Insbesondere für Φ_D^* fällt dabei auf, dass dieser letzte Abschnitt im Vergleich zu den anderen Lösungen sehr gradlinig verläuft und so die Hindernisdistanz noch stärker minimiert wird.
Das gehäufte Auftreten der zusätzlichen Richtungswechsel mit den alternativen Potentialansätzen ist ein starker qualitativer Nachteil der resultierenden Lösungen. Sie führen in beiden Fällen zu einer fast doppelt so langen Prozessdauer der Trajektorie von ca. 40 s. Im Kontext eines kompletten Systems zu autonomen Fahren, wie zum Beispiel das OPA^{3}L-System aus Abschnitt 1.2, ergeben sich zudem voraussichtlich Probleme durch die Ausführung einer solchen Referenztrajektorie. Zum einen sind viele Anhalte- und Anfahrvorgänge schwieriger für einen nachgelagerten Regler umzusetzen und zum anderen führen sie vermutlich zu einem deutlich weniger angenehmen Fahrgefühl für potentielle Passagiere.
In Bezug auf den Expansionsaufwand und die Rechenzeit ergeben sich auch hier wieder deutliche Vorteile für das einfachere Potential Φ_D^*. Im Vergleich der Varianten Φ_V^* und $\Phi_V^\diamond$ basierend auf den Voronoi-Pfaden hat in diesem Fall der dynamische Ansatz eine deutlich geringere Laufzeit. Dies deutet darauf hin, dass der Einsatz von Φ_V^* in komplexen Situationen mit vielen Optionen nicht nur zu qualitativ besseren Lösungen führt, sondern diese basierend auf dem dynamischen Voronoi-Pfad auch effizienter berechnet werden können.

Zusammenfassend kann festgehalten werden, dass ein sehr einfaches Potential wie Φ_D^* die Berechnung der Trajektorie in allen gezeigten Beispielen mit wesentlich weniger Iterationen und Zeitaufwand abschließt, sich dabei das Fehlen einer dynamischen Referenz aber auch qualitativ deutlich negativ im Ergebnis widerspiegeln kann.

4.2 Evaluation auf dem Realfahrzeug

Der in dieser Arbeit eingeführte generische Hybrid A*-Algorithmus ist das Kernelement Taktikautonomie des OPA^{3}L-Systems. Diese ist in der Lage, Manöverentscheidungen für autonomen Fahrzeuge in realen urbanen Umgebungen zu berechnen, vergleiche auch Abschnitt 1.2. Dabei wird auf den Komponenten von Algorith-

mus 3.4 aufgebaut, indem eingehende Umgebungsinformationen zum Beispiel durch Freibereichspolygone abstrahiert und anschließend eine Reihe von Handlungsalternativen durch hybride Trajektorienoptimierung berechnet werden. Auf dieser Basis wird schließlich eine Manöverentscheidung durch Auswahl einer der resultierenden Trajektorien getroffen.

Dieser Abschnitt beschreibt die Einzelheiten dieses Vorgehens und diskutiert die Eignung des generischen Hybrid A*-Algorithmus als dessen Grundlage. Dazu werden zunächst die Komponenten der Taktikautonomie von OPA^{3}L in Abschnitt 4.2.1 vorgestellt. Hier wird die Analyse des Voronoi-Pfades an verschiedenen Stellen eine wichtige Rolle spielen, um ein adaptives Verhalten ohne spezielles Vorwissen an die Umgebung umsetzen zu können. Zur Vereinfachung der Auswertung, werden nur Szenarien betrachtet, in der das Ego-Fahrzeug vorwärts fährt. Zudem wird insgesamt ein Fokus auf die wesentlichen Aspekte dieses in seiner Gesamtheit recht komplexen Systems gelegt; Details werden dabei zugunsten einer übersichtlicheren Zusammenfassung in Teilen nur knapp dargestellt, ohne dabei jedoch die Vollständigkeit der Beschreibung zu beschneiden. Abschließend wird der vorgestellte Ansatz durch reale Fahrten des autonomen Forschungsfahrzeuges aus Abbildung 1.4c in den Abschnitten 4.2.2 und 4.2.3 evaluiert.

4.2.1 Taktische Entscheidungen in OPA^{3}L

Die Aufgabe der Taktikautonomie in OPA^{3}L ist es, kontinuierliche Manöverentscheidungen für die unmittelbare Zukunft zu treffen. Das Ziel ist dabei, diese Vorgabe alle 100 ms zu aktualisieren und sie in Form einer Trajektorie dem Modul zur Fahrregelung bereitzustellen. Dieses bestimmt schließlich Steuersignale, um die jeweils aktuelle Entscheidung der Taktik auszuführen. Die Berechnung der taktischen Manöver selbst muss dabei so erfolgen, dass zum einen die langfristigen Vorgaben der Strategieautonomie erfüllt werden und zum anderen sämtliche Informationen über die Umgebung des Ego-Fahrzeuges berücksichtigt sind. Dabei soll beispielsweise die Gefahr einer möglichen Kollision minimiert werden. Zudem sollte die Vorgabe für den nachgelagerten Regler möglichst einfach ausführbar sein. Dies erfordert zum Beispiel, dass aufeinanderfolgende Vorgaben sich nicht häufig stark unterscheiden – bei plötzlichen Änderungen in der Umgebungswahrnehmung ist dies jedoch zwangsläufig nicht immer vermeidbar.

Die zu berücksichtigenden Umgebungsinformationen können in drei Kategorien unterteilt werden, wie in Tabelle 4.6 dargestellt ist. Erstens beinhaltet dies *feste Hindernisse*, welche durch die Kartierung der Bordsensorik (zum Beispiel auf Grundlage der LiDAR-Sensoren) als Punktwolke bereitgestellt werden oder durch eine im Vorfeld erstellte Offline-Karte bekannt sind (zum Beispiel sinnvoll für feste Objekte wie Gebäude oder Schilder). Eine Kollision mit Repräsentanten dieser Kategorie muss zwingend vermieden werden, weshalb sie als *harte* Beschränkung in der Planung zu

Tabelle 4.6: Übersicht zu Informationsquellen, Zeitverhalten und Art der räumlichen Beschränkung für unterschiedliche Informationen über die Umgebung eines autonomen Fahrzeuges zur Berücksichtigung durch die Taktikautonomie in OPA^{3}L

Umgebungsdaten	Quelle	Zeitverhalten	Räumliche Beschränkung
Feste Hindernisse	Offline-Karte, Online-Karte	Statisch	Hart
Fahrspur	Offline-Karte, Bildverarbeitung	Statisch	Weich
Bewegliche Objekte	Objekttracking, V2X	Dynamisch	Hart

berücksichtigen sind.
Dem gegenüber steht die zweite Kategorie der *Spurinformationen*. Sie können ebenfalls durch eine Offline-Karte bereitgestellt oder durch die Verarbeitung von Kameradaten im Umfeld des Fahrzeuges erkannt werden. Um das Fahren in einer Spur zu ermöglichen, sollten diese Informationen – soweit möglich – als Teil der Trajektorienberechnung beachtet werden. Um zum Beispiel harten Hindernissen ausweichen zu können, ist gegebenenfalls auch das Verlassen einer Fahrspur nötig, weshalb sie im Folgenden als *weiche* Beschränkungen aufgefasst werden. Einen Spezialfall stellen durchgezogene Linien dar: Diese können zunächst als harte Hindernisse berücksichtigt, aber in bestimmten Situationen (zum Beispiel im Fall einer Straßenblockade) als weich gekennzeichnet werden.
Die dritte Kategorie betrifft bewegliche Objekte, deren Dynamik ebenfalls durch die Bordsensorik erkannt werden kann (durch LiDAR-Sensoren, Radar oder Kamera) oder zum Beispiel durch V2X Kommunikation übermittelt wird. Wie feste Hindernisse müssen sie als harte Beschränkungen berücksichtigt werden, haben dabei aber ein dynamisches Zeitverhalten.

Die Informationen über die Umgebung bilden zusammen mit einer Zustandsschätzung des Ego-Fahrzeuges sowie der strategischen Routenvorgabe die Basis, auf der die Taktikautonomie ihre Entscheidungen trifft. Der Prozess der Entscheidungsfindung besteht dabei aus insgesamt sechs aufeinanderfolgenden Schritten, welche als eine Erweiterung der Abfolge aus Algorithmus 3.4 aufgefasst werden können: Die Aspekte zur Umgebungsverarbeitung, Zielzustands- und Trajektorienberechnung werden durch eine kriterienbasierte Manöverentscheidung sowie zwei Schritte zur adaptiven Geschwindigkeitsanpassung ergänzt. Generell wird dabei die Zielgeschwindigkeit der Taktikautonomie extern als $v_{\text{set}}^{\text{ext}}$ vorgegeben, zum Beispiel durch die Strategieautonomie oder den Fahrgast; sie wird jedoch situationsbedingt redu-

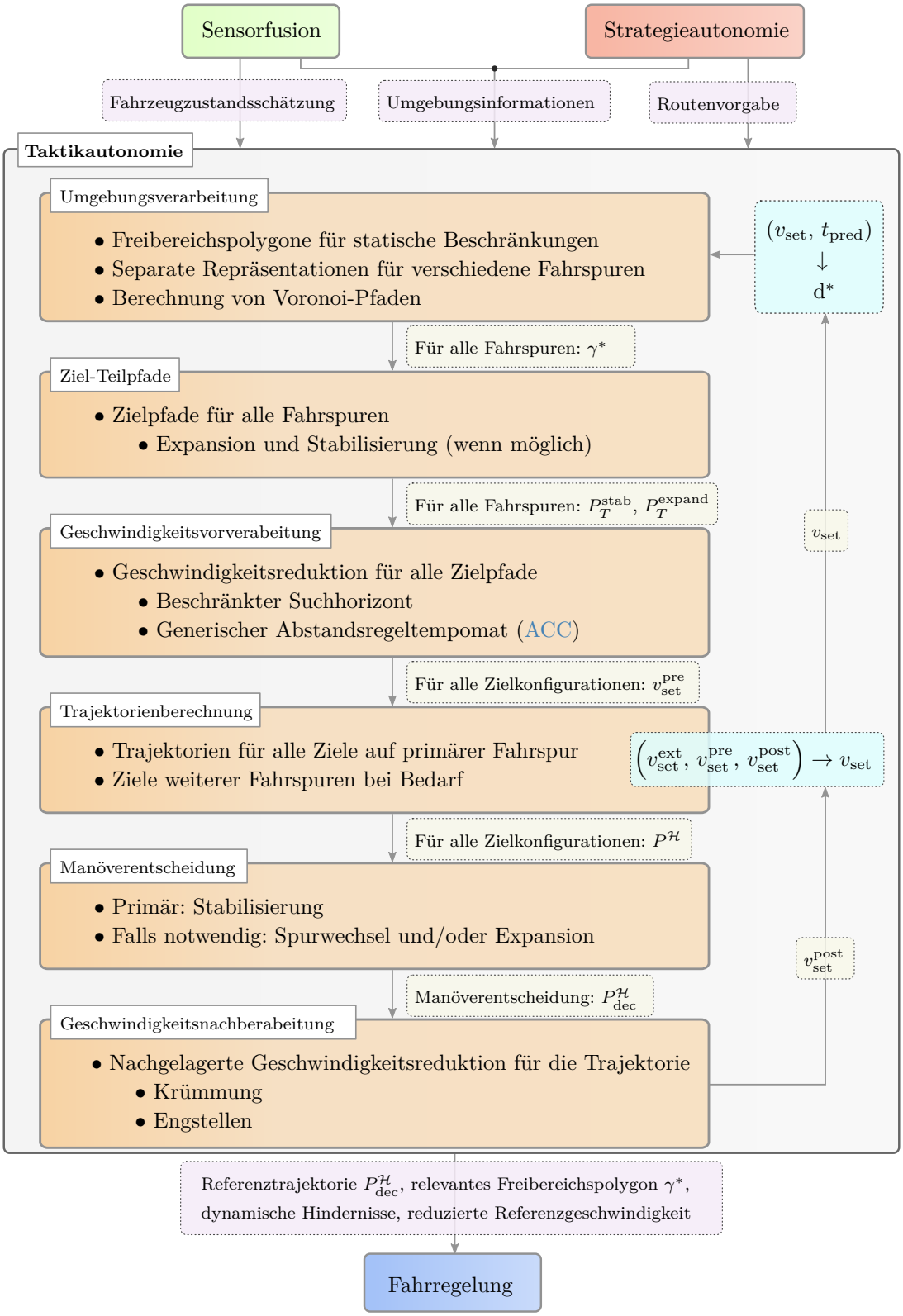


Abbildung 4.10: Verarbeitungsfolge der Taktikautonomie, um eingehende Informationen über Fahrzeugzustand, Fahrzeugumgebung und Routenvorgabe in Manöverentscheidungen in Form einer Trajektorie zu überführen

ziert, was in der angepassten Geschwindigkeitsvorgabe v_{set} resultiert. Zudem berücksichtigt die Taktik einen Prädiktionszeithorizont t_{pred}, welcher beschreibt, wie weit sie ihre Entscheidungen in die Zukunft projiziert. Zusammen mit der adaptiven Geschwindigkeitsvorgabe v_{set} ergibt sich die Prädiktionsdistanz

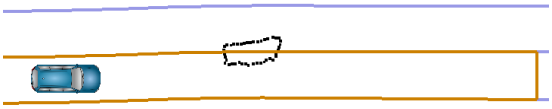
$$\mathrm{d}^* \coloneqq \max(v_{\text{set}} \cdot t_{\text{pred}},\, \mathrm{d}^*_{\text{min}}). \tag{4.1}$$

Diese ist für niedrige Absolutgeschwindigkeiten durch $\mathrm{d}^*_{\text{min}}$ nach unten beschränkt, um beispielsweise auch eine Planung aus dem Stillstand heraus zu ermöglichen. Die Prädiktionsdistanz ist der primäre Einflussfaktor zur Festlegung von situationsabhängigen Expansionstiefen innerhalb der automatischen Berechnung von Freibereichspolygonen und zur Definition von Entfernungen möglicher Zielkonfigurationen. Da weiter entfernte Ziele zu einem höheren Aufwand zur Berechnung der resultierenden Trajektorie führen, vergleiche Abschnitt 4.1.1.1, wirkt d^* damit auch wesentlich auf die benötigte Rechenzeit der Taktik insgesamt ein.

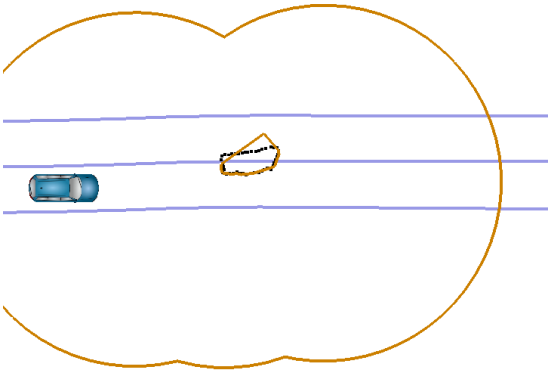
Die Verarbeitungsschritte der Taktikautonomie sind schematisch in Abbildung 4.10 dargestellt und werden im Folgenden kurz beschrieben.

Umgebungsverarbeitung Um die unterschiedlichen Typen von Hindernissen sowie verschiedene Fahrspuren differenziert berücksichtigen zu können, werden eine Reihe von Freibereichspolygonen berechnet, wie in Abbildung 4.11 exemplarisch dargestellt. Dazu wird mit dem entsprechenden Algorithmus 3.1 solange expandiert, bis keine neuen Expansionspunkte gefunden werden können oder die Prädiktionsdistanz d^* erreicht ist. Für eine gegebene Fahrspur werden dabei zunächst die weichen und harten Hindernisse separat durch γ^{soft} und γ^{hard} beschrieben, vergleiche Abbildungen 4.11b und 4.11c. Der Schnitt γ^{merge} entspricht dann dem Bereich, in dessen Inneren sich das Ego-Fahrzeug bewegen soll. Je nachdem, wie weit Objekte in die betrachtete Fahrspur hinein ragen, ist dies jedoch nicht zwingend möglich, wie Abbildung 4.11d zeigt. Um diese Fälle zu adaptiv berücksichtigen, wird γ^{merge} lokal vergrößert, sofern die betroffenen Bereiche nicht durch das Freibereichspolygon γ^{hard} beschränkt sind. Die Stellen, an denen eine Vergrößerung notwendig ist, können generisch identifiziert werden, indem die Kreisüberdeckung des Ego-Fahrzeuges in allen Segmentübergängen des statischen Voronoi-Pfades auf Kollision mit γ^{hard} hin untersucht wird. Im betrachteten Beispiel sind diese kritischen Segmente in Abbildung 4.11e gezeigt. Um diese herum wird der Anteil der weichen Hindernisse in γ^{merge} aufgeweicht und die so gegebenenfalls vergrößerte Fahrspur im erweiterten Freibereichspolygon γ^* zusammengefasst. Dieses stellt nicht nur einen adaptiven Planungsbereich dar, sondern resultiert auch in einem angepassten Voronoi-Pfad, welcher in sicherem Abstand um das Hindernis herum führt. Dies ist in der zugehörigen Abbildung 4.11f zu sehen. Der hier beschriebene Prozess wird für alle betrachteten Fahrspuren durchgeführt, vergleiche Abbildung 4.11g, sodass diese in den folgenden Schritten der Entscheidungsfindung berücksichtigt werden können.

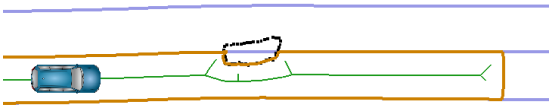
(a) Exemplarische Situation zur Behandlung von *weichen* Spurinformationen und *harten* Hindernisdaten



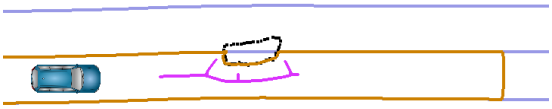
(b) Freibereichspolygon γ^{soft} zur Berücksichtigung der weichen Informationen zur aktuellen Fahrspur



(c) Freibereichspolygon γ^{hard} zur Berücksichtigung harter Hindernisdaten

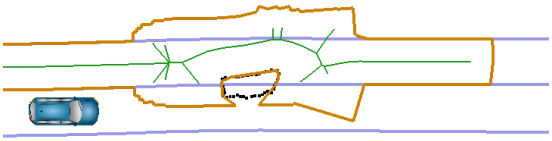


(d) Zusammenfassung der Polygone zu γ^{merge} durch Polygonschnitt von γ^{hard} und γ^{soft}, vergleiche Abbildung 3.12b. Der statische Voronoi-Pfad ist in grün dargestellt.



(e) Konfliktbereich des statischen Voronoi-Pfades aus Abbildung 4.11d in pink

(f) Erweitertes Freibereichspolygon γ^* und zugehöriger Voronoi-Pfad



(g) Ergebnis der Schritte (b) bis (f) für die zweite Fahrspur

Abbildung 4.11: Vorverarbeitungsschritte der Taktikautonomie um unterschiedliche Arten statischer Hindernisse und verschiedene Fahrspuren durch Freibereichspolygone (in orange) darzustellen. Exemplarisch werden hier harte Hindernisse aus der Online-Kartierung in schwarz und die Fahrspuren als weiche Hindernisse in blau gezeigt.

Das erweiterte Freibereichspolygon γ^* und dessen abgeleitete Voronoi-Pfade stellen im Weiteren die Grundlage zur Berechnung des Voronoi-Potentials dar, welches die explorierten Knoten innerhalb des Hybrid A*-Algorithmus bewertet. Im Rahmen der Kollisionsprüfung `IsCollision` aus Abschnitt 3.4 wird jedoch nur das Freibereichspolygon γ^{hard} geprüft. Dadurch werden grundsätzlich auch Fahrzeugzustände zugelassen, die nicht strikt innerhalb von γ^* liegen.

Ziel-Teilpfade Für jede betrachtete Fahrspur werden Ziel-Teilpfade entlang des statischen Voronoi-Pfades von γ^* mit Algorithmus 3.3 berechnet. Hierbei ergibt sich für die Taktikautonomie eine wesentliche Einflussmöglichkeit in Bezug auf das Fahrverhalten des autonomen Ego-Fahrzeugs: Unterscheiden sich

zeitlich aufeinanderfolgende Zielvorgaben stark, so führt dies in letzter Konsequenz zu stark streuenden Referenztrajektorien für den modellprädiktiven Regler. Dieser muss dann tendenziell stärkere Steuersignale ausrechnen, um diese umzusetzen zu können. Ein wesentlich ruhigeres, für den Fahrgast angenehmeres Verhalten wird dagegen erreicht, wenn konsekutive Zielvorgaben räumlich ähnlich sind, solange dies möglich ist. War der letzte Aufruf der Taktikautonomie erfolgreich und ist der Ziel-Referenzpunkt der entsprechenden Trajektorie durch $p_{\mathrm{Ego},T}^{\mathrm{Last}}$ gegeben, dann wird ein solcher *stabilisierender* Teilpfad P_T^{stab} durch die Kostenfunktion

$$f_T^{\mathrm{stab}}(p) \coloneqq \omega_{\mathrm{stab}} \cdot \mathrm{d}(p_{\mathrm{Ego},T}^{\mathrm{Last}},\, p) + \omega_{\mathrm{expand}} \cdot |\mathrm{d}(p_{\mathrm{Ego}},\, p) - \mathrm{d}^*|$$

für einen Punkt $p \in \mathbb{R}^2$ beschrieben. Darin werden zwei Terme kombiniert, die zum einen bewirken, dass in die Nähe von $p_{\mathrm{Ego},T}^{\mathrm{Last}}$ geplant wird, und zum anderen eine Expansion vom aktuellen Referenzpunkt p_{Ego} aus einfordern, sodass der Zielpunkt sich entlang der Routenvorgabe nach vorne bewegt. Beide Terme werden durch die konstanten Gewichtungsparameter ω_{stab}, $\omega_{\mathrm{expand}} \in \mathbb{R}$ gegeneinander abgewogen. Neben f_T^{stab} wird die Suche nach einem Voronoi-Teilpfad durch eine Funktion zur Pfadbeschränkung $\texttt{PathValid}_{\mathrm{stab}}$ charakterisiert. Diese beschränkt die maximale Abweichung zu $p_{\mathrm{Ego},T}^{\mathrm{Last}}$, sodass die Nähe zur letzten Zielkonfiguration und damit die stabilisierende Eigenschaft gewährleistet ist. Des Weiteren wird durch sie garantiert, dass die resultierende Pfadlänge nicht die aktuelle Prädiktionsdistanz d^* überschreitet.

Aufgrund der vorgegebenen Beschränkungen durch $\texttt{PathValid}_{\mathrm{stab}}$ kann es passieren, dass kein zulässiger stabilisierender Voronoi-Teilpfad P_T^{stab} gefunden werden kann. Zudem ist die Berechnung eines solchen Teilpfades generell nur für diejenige Spur sinnvoll, auf der die zuletzt vorgegebene Trajektorie der Taktik lag. Aus diesem Grund wird für jede Spur (zusätzlich) ein *expandierender* Zielpfad P_T^{expand} vorgegeben. Dessen Beschreibung basiert auf dem entsprechenden expandierenden Anteil der Optimierungsfunktion f_T^{stab} und ist damit durch

$$f_T^{\mathrm{expand}}(p) \coloneqq |\mathrm{d}(p_{\mathrm{Ego}},\, p) - \mathrm{d}^*|$$

gegeben. Die Pfadbeschränkungen $\texttt{PathValid}_{\mathrm{expand}}$ reduzieren sich in diesem Fall auf die Berücksichtigung der maximalen Prädiktionsdistanz d^*.

Geschwindigkeitsvorverarbeitung Grundsätzlich versucht die Taktikautonomie zu einer extern vorgegebenen Zielgeschwindigkeit $v_{\mathrm{set}}^{\mathrm{ext}}$ zu planen. Diese kann zum Beispiel durch die Strategieautonomie oder durch einen menschlichen Benutzer vorgegeben sein. Abhängig von den aktuellen Informationen über die Fahrzeugumgebung kann es jedoch notwendig sein, dass die tatsächlich geplante Geschwindigkeit im Vergleich zur Geschwindigkeitsvorgabe reduziert

werden muss. Je nach Ursache kann dies vor der Trajektorienberechnung, basierend auf den im Vorfeld berechneten Teilpfaden, oder im Anschluss daran, basierend auf der tatsächlichen Trajektorie umgesetzt werden.

Für einen gegebenen Ziel-Teilpfad werden in der Geschwindigkeitsvorverarbeitung zwei Aspekte berücksichtigt:

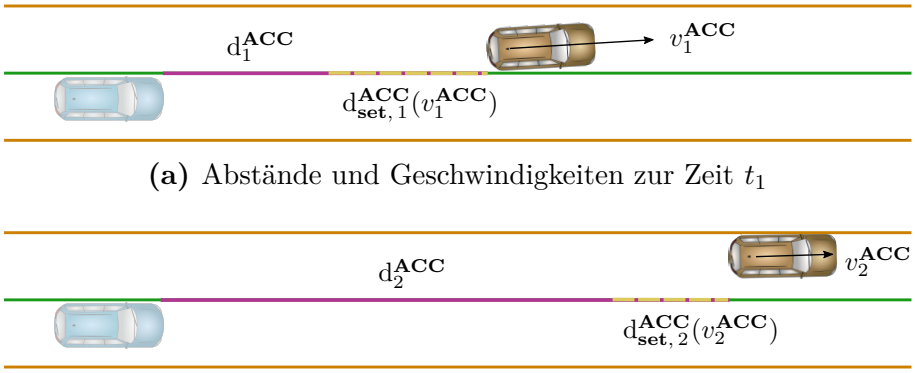
1. *Ist der Pfad wesentlich kürzer als die aktuelle Prädiktionsdistanz* d^**?*
   Dieser Fall tritt in der Regel bei starken statischen Beschränkungen auf, zum Beispiel wenn die betrachtete Fahrspur endet oder durch ein Objekt blockiert ist. Da nicht garantiert ist, dass es eine Ausweichmöglichkeiten gibt, wird für diesen Zielpfad eine reduzierte Zielgeschwindigkeit $v_{\mathrm{set}}^{\mathrm{dist}}$ berechnet, sodass das Fahrzeug unter Berücksichtigung der vorgegebenen maximalen Verzögerung $a_r^{\min}$ noch zum stehen kommen kann.
2. *Gibt es bewegliche Objekte, die entlang des gegebenen Pfades vor dem Ego-Fahrzeug fahren?*
   Sollte die prädizierte Bewegung eines dieser Objekte zu potentiellen Konflikten mit dem Ego-Fahrzeug bei Anwendung der aktuellen Zielgeschwindigkeit führen, wird eine reduzierte Vorgabe $v_{\mathrm{set}}^{\mathrm{ACC}}$ bereitgestellt. Diese garantiert, dass während des Prädiktionszeithorizonts t_{pred} stets ein (adaptiver) Mindestabstand gewährleistet ist, und führt damit zu einem eher konservativen aber auch sicheren Verhalten.
   Zur Berechnung von $v_{\mathrm{set}}^{\mathrm{ACC}}$ wird eine Menge von diskreten Zeitschritten $\{t_1, \ldots,\ t_n = t_{\mathrm{pred}}\}$ betrachtet. Für jeden enthaltenen Zeitpunkt t_i werden dann die Objektgeschwindigkeiten v_i^{ACC} zusammen mit den zugehörigen Abständen $\mathrm{d}_i^{\mathrm{ACC}}$ zwischen dem Objekt und dem Ego-Fahrzeug analysiert, vergleiche Abbildung 4.12. Distanzberechnungen erfolgen dabei entlang des Voronoi-Pfades, wodurch auch bei kurvigen Strecken gute Approximationen erreicht werden. Basierend auf der maximalen Verzögerung $a_r^{\min}$ wird daraus eine Zielgeschwindigkeit $v_{\mathrm{set,\,i}}^{\mathrm{ACC}}$ abgeleitet, sodass das Ego-Fahrzeug zu jeder Zeit einen adaptiven Zielabstand $\mathrm{d}_{\mathrm{set},\,i}^{\mathrm{ACC}}(v_i^{\mathrm{ACC}})$ einhält. Dieser kann zum Beispiel anhand einer festgelegten Zeitlücke t^{ACC} gemäß

$$\mathrm{d}_{\mathrm{set},\,i}^{\mathrm{ACC}} = v_i^{\mathrm{ACC}} \cdot t^{\mathrm{ACC}}$$

   direkt aus der Objektgeschwindigkeit v_i^{ACC} abgeleitet werden. Das Minimum über alle diskreten Zeitpunkte

$$v_{\mathrm{set}}^{\mathrm{ACC}} := \min_{i \in \{1,\ldots,n\}} \left\{ v_{\mathrm{set,\,i}}^{\mathrm{ACC}} \right\}$$

   ergibt die resultierende Geschwindigkeitsvorgabe.
   Das hier beschriebene Vorgehen setzt funktional einen Abstandsregeltempomaten – auch *Adaptive Cruise Control* (ACC) – um und wird in Abschnitt 4.2.3 im Detail vorgestellt.



(a) Abstände und Geschwindigkeiten zur Zeit t_1

(b) Abstände und Geschwindigkeiten zur Zeit t_2

Abbildung 4.12: Exemplarische Geschwindigkeiten v_i^{ACC} eines voraus bewegenden Objektes (in gelb) sowie Abstände von diesem zum Ego-Fahrzeug (in hellblau) zu verschiedenen Zeitpunkten t_i. Gezeigt ist der Voronoi-Pfad (in grün) innerhalb des Freibereichspolygons (in orange). Die zur Berechnung der ACC-Geschwindigkeit $v_{\mathrm{set,\,i}}^{\mathrm{ACC}}$ notwendigen Abstände d_i^{ACC} werden entlang des Voronoi-Pfades bestimmt. Dabei wird stets der aktuelle Zustand des Ego-Fahrzeugs mit dem zur Zeit t_i angenommenen Zustand des Objektes verglichen. Der resultierende Wert beinhaltet zudem den ACC-Zielabstand $d_{\mathrm{set},\,i}^{\mathrm{ACC}}$, welcher von der zum jeweiligen Zeitpunkt vorliegenden Objektgeschwindigkeit abhängt.

Die genannten Betrachtungen werden für jeden Ziel-Teilpfad einzeln umgesetzt, sodass sich für jeweils eine eigene adaptive Zielgeschwindigkeit $v_{\mathrm{set}}^{\mathrm{pre}} := \min(v_{\mathrm{set}}^{\mathrm{dist}}, v_{\mathrm{set}}^{\mathrm{ACC}})$ zur Berücksichtigung in der Trajektorienberechnung ergibt.

Trajektorienberechnung Es werden Trajektorien $P^{\mathcal{H}}$ zu den im Vorfeld bestimmten Zielkonfigurationen gemäß Zeile 5 von Algorithmus 3.4 berechnet. Dabei wird das jeweils zugehörige erweiterte Freibereichspolygon γ^* zur Beschreibung der statischen Beschränkungen verwendet. Die Zielgeschwindigkeit v_{set} ergibt sich mit

$$v_{\mathrm{set}} = \min(v_{\mathrm{set}}^{\mathrm{ext}}, v_{\mathrm{set}}^{\mathrm{pre}}, v_{\mathrm{set}}^{\mathrm{post}})$$

als Minimum der externen $v_{\mathrm{set}}^{\mathrm{ext}}$, der vorverarbeiteten $v_{\mathrm{set}}^{\mathrm{pre}}$ sowie der aus dem letzten Schritt der Taktik nachverarbeiteten Geschwindigkeitsvorgabe $v_{\mathrm{set}}^{\mathrm{post}}$. Um zu verhindern, dass sehr viel Rechenzeit auf einzelne schwer lösbare Probleme verwendet wird, werden Berechnungen nach einer maximalen Anzahl von $\mathbf{O}_{\mathrm{max}}$ geöffneten Knoten abgebrochen.

In den meisten Fällen kann die Taktikautonomie der von der Strategieautonomie empfohlenen Fahrspur folgen. Dies kann genutzt werden, um die Gesamtrechenzeit zu reduzieren, indem die hybride Trajektorienoptimierung vorrangig zu dieser beziehungsweise der aktuell befahrenen Fahrspur durchgeführt wird.

Berechnungen zu allen weiteren Zielkonfigurationen werden dann nur bei Bedarf vorgenommen.

Manöverentscheidung Es folgt eine Auswahl aus den im Vorfeld berechneten Trajektorien als Manöverentscheidung $P_{\mathrm{dec}}^{\mathcal{H}}$. Wie bereits erläutert, wird dabei nach Möglichkeit die zeitlich stabilisierende Lösung basierend auf P_T^{stab} bevorzugt. Sollte diese nicht vorliegen, kann jedoch auch ein Wechsel zur expandierenden Trajektorie derselben Fahrspur vorgenommen werden. Sind alle Trajektorien entlang der aktuellen Spur wesentlich kürzer als die Prädiktionsdistanz d^* oder befindet sich das Fahrzeug nicht auf der durch die Strategie empfohlenen Fahrspur, kann hier auch zugunsten einer Spurwechseltrajektorie entschieden werden.

Dass Lösungstrajektorien zu einzelnen Zielvorgaben nicht zur Manöverentscheidung vorliegen, kann generell zwei Ursachen haben: Entweder konnte zuvor kein Ziel-Teilpfad gefunden werden oder die Trajektorienoptimierung zu einer Konfiguration war nicht erfolgreich. Befindet sich das Fahrzeug in einer besonders kritischen Situation, kann es auch passieren, dass durch die vorangegangenen Schritte überhaupt keine Trajektorie berechnet werden konnte. In diesem Fall ist die Taktik entsprechend nicht in der Lage, eine sichere Handlungsalternative bereitzustellen. Dies kann zum Beispiel bei plötzlich vor dem Fahrzeug auftauchenden Hindernissen auftreten. In einer solchen Situation würde das nachgelagerte Notstopp-Modul ein starkes Bremsmanöver auslösen, um das Ego-Fahrzeug möglichst sicher in den Stillstand zu überführen, vergleiche Abschnitt 1.2.

Geschwindigkeitsnachverarbeitung Basierend auf der Manöverentscheidung in Form der ausgewählten Trajektorie wird erneut eine reduzierte Zielgeschwindigkeit als $v_{\mathrm{set}}^{\mathrm{post}}$ berechnet. Wie im Vorfeld erläutert, werden dabei zwei Aspekte untersucht:

1. *Beinhaltet die Trajektorie eine nicht vernachlässigbare Krümmung?*
   Kurvenfahrten, Ausweichmanöver oder Spurwechsel können – je nach konkreter Szene – partiell vergleichsweise starke Lenkbewegungen erfordern, welche äquivalent zu einem (lokal) kleinen Kurvenradius R der berechneten Trajektorie $P_{\mathrm{dec}}^{\mathcal{H}}$ sind. Um solche Situationen möglichst einfach regelbar und für einen potentiellen Fahrgast möglichst angenehm zu machen, wird die Geschwindigkeitsvorgabe basierend auf R reduziert. Wie schnell das Ego-Fahrzeug fahren soll, kann durch die Vorgabe einer maximalen, absoluten Radialbeschleunigung a_r^{rad} definiert werden, woraus sich die entsprechende Höchstgeschwindigkeit $v_{\mathrm{set}}^{\mathrm{rad}}$ durch

   $$v_{\mathrm{set}}^{\mathrm{rad}} = \sqrt{a_r^{\mathrm{rad}} \cdot R}$$

   ergibt [14].

2. *Verläuft die Trajektorie durch eine Engstelle?*
   Der in dieser Arbeit vorgestellte generische Hybrid A*-Algorithmus versucht per Konstruktion möglichst zentral innerhalb des Freibereichspolygons zu planen; trotzdem können Situationen entstehen, in der das Ego-Fahrzeug beidseitig sehr eng an Hindernissen vorbei fahren muss. Um diese Situationen möglichst sicher zu handhaben und das Sicherheitsgefühl des Fahrgastes zu erhöhen, wird eine reduzierte Zielgeschwindigkeit $v_{\text{set}}^{\text{obstacle}}$ abhängig vom (lokal) zur Verfügung stehenden Platz berechnet. Dieser ergibt sich wiederum durch Analyse des Freibereichspolygons γ^{hard}, welches die harten Hindernisse in der Umgebung des Fahrzeuges abbildet.

Aus den beiden berechneten Geschwindigkeiten $v_{\text{set}}^{\text{rad}}$ und $v_{\text{set}}^{\text{obstacle}}$ werden jeweils effektive Geschwindigkeitsvorgaben generiert, indem sie mit der Distanz zum auslösenden Ereignis – also der Stelle mit hoher Krümmung beziehungsweise der Engstelle – und der vorgegebenen maximalen Verzögerung $a_r^{\min}$ verrechnet werden. Das Minimum dieser Werte ergibt dann die nachverarbeitete Zielgeschwindigkeit $v_{\text{set}}^{\text{post}}$. Diese wird unter anderem für die Trajektorienberechnung zur Verfügung gestellt und kann damit erst im nächsten Aufruf der Taktikautonomie berücksichtigt werden.

Die Entscheidung in Form der Trajektorie wird schließlich an die modellprädiktive Regelung als Referenz zur Ausführung übergeben. Dies wird ergänzt durch das zugehörige Freibereichspolygon sowie die Liste der entsprechend relevanten dynamischen Hindernisse. Sollte die im letzten Schritt der Taktikautonomie berechnete nachverarbeitete Zielgeschwindigkeit $v_{\text{set}}^{\text{post}}$ geringer als die Vorgabe der übergebenen Referenztrajektorie sein, so wird die entsprechend reduzierte Geschwindigkeit ebenfalls bereitgestellt, sodass diese im Rahmen der Regelung direkt berücksichtigt werden kann; innerhalb der Taktik wird sie dagegen erst im nächsten Aufruf zur Definition der Zielgeschwindigkeit v_{set} verwendet, vergleiche Abbildung 4.10.

Durch das präsentierte Verfahren kann die Taktikautonomie adaptiv und ohne spezielle Annahmen an das Szenario sichere Referenztrajektorien bereitstellen. Der resultierende Funktionsumfang wird in den folgenden Abschnitten für zwei autonome Fahrten eines Forschungsfahrzeugs exemplarisch vorgestellt. Alle dabei verwendeten Parameter – sofern sie im Rahmen der Taktikautonomie nicht situationsabhängig berechnet werden – sind in Tabelle A.2 aufgeführt.

4.2.2 Taktische Entscheidungen im urbanen Straßenverkehr

Die Taktikautonomie im Projekt OPA^{3}L wurde mit dem Schwerpunkt auf suburbane Anwendungsbereiche entwickelt. Um deren Potential in diesem Kontext zu beschreiben, stellt dieser Abschnitt eine autonome Fahrt mit dem System aus Abschnitt 1.2 durch das Randgebiet Borgfeld der Stadt Bremen vor, vergleiche Abbildung 4.13. Die

Abbildung 4.13: Satellitenaufnahme eines Teilbereichs des Bremer Stadtteils Borgfeld (© 2022 GeoBasis-DE/BKG) zusammen mit der Positionsschätzung (in rot) des Forschungsfahrzeuges während der in diesem Abschnitt beschriebenen autonomen Fahrt

gefahrene Strecke startet in Punkt **(A)**, verläuft zentral durch den Stadtteil, führt das Fahrzeug durch eine Wendeschleife und anschließend zurück zum Startpunkt. Die im Folgenden präsentierten Ergebnisse resultieren aus einer Fahrt, die das Forschungsfahrzeug vollkommen eigenständig durchgeführt hat – einzig automatisiert getroffene Spurwechselentscheidungen wurden durch einen Knopfdruck des Sicherheitsfahrers am Lenkrad vor der Ausführung bestätigt.

Durch seine Nähe zum Stadtrand weist Borgfeld fast schon ländlichen Charakter auf, wodurch das gewählte Szenario einige spezielle Herausforderungen für ein autonomes System beinhaltet. Um dies zu illustrieren, zeigt Abbildung 4.14 ausgewählte Bilder der Frontkamera des Forschungsfahrzeuges. Die befahrene Strecke verläuft durchgängig zweispurig, sodass diese im Folgenden als *linke* und *rechte* Spur unterschieden werden (aus der jeweiligen Sicht des Ego-Fahrzeugs). Die Straßen kennzeichnen sich zudem dadurch, dass keine Spurmarkierungen auf der Fahrbahn vorhanden sind. In der Folge ist die Spurerkennung mithilfe der Frontkamera erschwert, weshalb die hier gezeigten Experimente auf im Vorfeld aufgenommenen Kartendaten der Straßenverläufe basieren. Diese sind recht unpräzise, sodass Fehler von bis zu einem Meter auftreten können. Die Taktikautonomie muss diese Ungenauigkeiten in den vermessenen Spurverläufen kompensieren, wenn sich diese mit anderen Sensordaten (wie zum Beispiel denjenigen der LiDAR-Sensoren) widersprechen. Solch eine adaptive Anpassung wird ebenfalls benötigt, um auf der Spur positionierte Poller oder Leitbaken zu berücksichtigen, die den Verkehr im Testgebiet an verschiedenen Stellen steuern sollen, siehe Abbildungen 4.14a und 4.14b. Da sich das Forschungsfahrzeug

(a) Unmarkierte Straße, schwer einsehbare Kurve und kleines Hindernis (Poller) **(B)**

(b) Verkehrsberuhigung durch kleine Leitbaken, Poller und Straßenmarkierung **(A)**

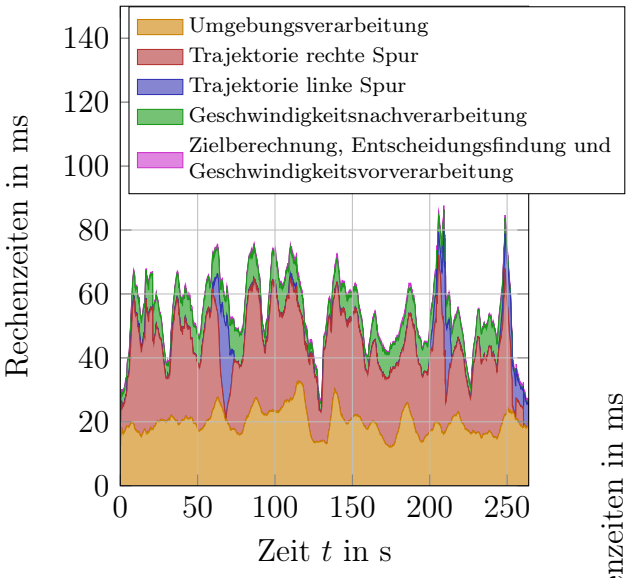
(c) Entgegenkommendes Fahrzeug in schlecht einsehbaren Stelle **(D)**

(d) Parkender Pkw am Straßenrand und entgegenkommendes Fahrzeug **(C)**

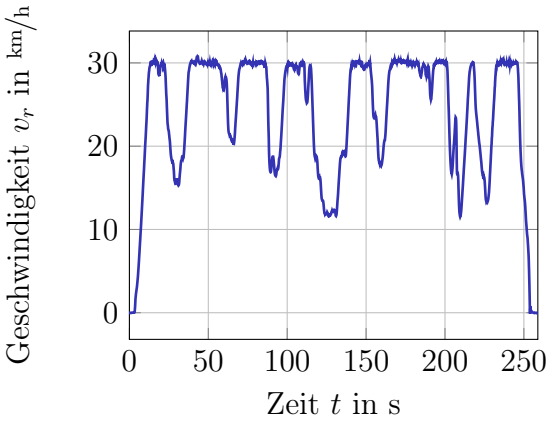
Abbildung 4.14: Aufgezeichnete Bilder der Frontkamera des Forschungsfahrzeuges während der in diesem Abschnitt ausgewerteten autonomen Fahrt. Die jeweiligen Orte sind in der Karte aus Abbildung 4.13 markiert.

auf öffentlichen Straßen bewegt, müssen allgemein zudem andere (statische und dynamische) Verkehrsteilnehmer wie Pkw, Radfahrer oder Fußgänger berücksichtigt werden. Im diskutierten Szenario ergaben sich Situationen mit entgegenkommendem Verkehr, auf der Straße geparkten Fahrzeugen oder einer Kombination aus beidem, wie in den Szenen der Abbildungen 4.14c und 4.14d gezeigt wird.

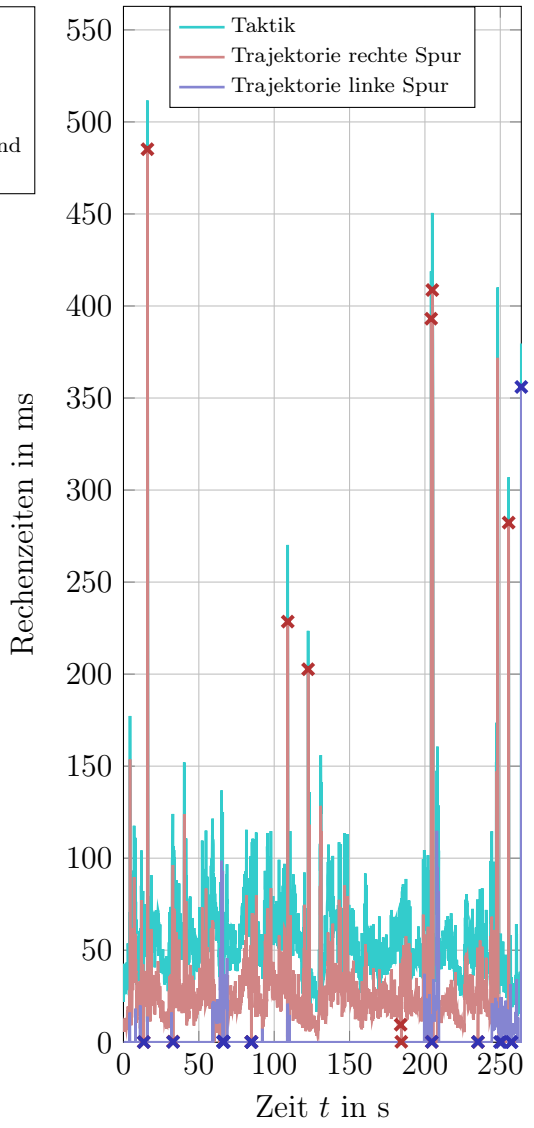
Die betrachtete Strecke hat insgesamt eine Länge von 1,7 km und wurde in 264 s durch das autonome Forschungsfahrzeug abgefahren. Dabei wurde der Hybrid A*-Algorithmus insgesamt 3209 mal aufgerufen, was zu 3178 erfolgreichen Berechnung einer Trajektorie führte. Dies entspricht einer Erfolgsquote von mehr als 99 %. Einzelne fehlgeschlagene Berechnungen können dabei leicht kompensiert werden, da die Taktikautonomie in jedem Schritt verschiedene Handlungsoptionen berechnet, vergleiche Abschnitt 4.2.1. Ursachen für nicht berechenbare Manöver können dabei auf die der Trajektorienberechnung vorangehenden Schritte zurückgeführt werden: hier führen Kombinationen von automatisch berechneten Zielkonfigurationen und Umgebungsdarstellungen zu Aufgaben, die mit dem zugrunde liegenden Einspurmodell sowie den gegebenen Steuerbeschränkungen gar nicht, oder nicht im Rahmen der Beschränkung der maximal zu öffnenden Knoten $\mathbf{O}_{\max}$ gelöst werden können.



(a) Aufsummierte und über 5 s gemittelte Rechenzeiten der Teilschritte der Taktik



(b) Verlauf der geschätzten Fahrzeuggeschwindigkeit, welche auf der gefahrenen Route einer Begrenzung von 30 km/h unterliegt



(c) Unbereinigte Rechenzeiten der Trajektorienberechnung und Taktik. Die Kreuze markieren jeweils, dass keine Trajektorie gefunden werden konnte.

Abbildung 4.15: Rechenzeiten der Taktikautonomie bei der Ausführung auf dem Forschungsfahrzeug während der autonomen Fahrt durch Borgfeld aus Abbildung 4.13. Als Referenz wird ebenfalls die gefahrene Geschwindigkeit gezeigt.

Abbildung 4.15a zeigt die Rechenzeiten aller Teilschritte der Taktikautonomie; die Summe dieser ergibt dann entsprechend die Gesamtlaufzeit der Taktik. Um Tendenzen im Verlauf erkennen zu können, werden die Messwerte über die jeweils vorangegangenen 5 s gemittelt. An den Ergebnissen lässt sich klar erkennen, dass die Schritte zur Umgebungsvorverarbeitung und Trajektorienberechnung den Großteil des benötigten Rechenaufwandes ausmachten. Trajektorien wurden dabei fast überwiegend zur vom Fahrzeug aus rechten (und damit eigenen) Fahrspur ausgerechnet. Insgesamt wurden zweimal – nach ca. 60 s bei **(E)** und nach 210 s bei **(C)** – am Straßenrand parkende Pkw überholt, was jeweils einen kurzzeitigen Wechsel zur linken (entgegenkommenden) Fahrspur notwendig gemacht hat. Die Notwendigkeit, entsprechende zusätzliche Referenztrajektorien auszurechnen, wurde durch die Taktikautonomie automatisch erkannt und an diesen Stellen gezielt ergänzt. Zum Ende des gezeigten Szenarios fuhr das Fahrzeug auf die Verkehrsberuhigung aus Abbildung 4.14b zu. Aufgrund der eingeschränkten Suchweite entlang der eigenen Spur wurden hier ebenfalls Referenztrajektorien zur linken Fahrbahn berechnet; es wurde jedoch kein Spurwechsel mehr durch den Sicherheitsfahrer freigegeben, weshalb das Fahrzeug vor der Verkehrsberuhigung zum Stehen kam.
Die Geschwindigkeitsnachverarbeitung stellt den letzten Teilschritt dar, der in einem erkennbaren Aufwand resultierte. Dies ergab sich vor allem aus der vergleichsweise aufwändigen Prüfung von Engstellen entlang der ausgewählten Trajektorie, vergleiche Abschnitt 4.2.1. Die übrigen Teilschritte waren bezüglich ihrer Rechenzeit vernachlässigbar.

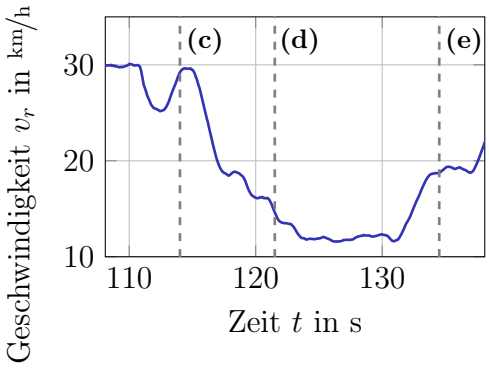
Die Geschwindigkeitsnachverarbeitung wies mit ca. 10 ms einen vergleichsweise konstanten Verlauf auf. Dies galt mit Einschränkung ebenfalls für die Umgebungsvorverarbeitung, deren Rechenzeiten zwar leichte Schwankungen beinhalteten, welche aber stets in der Nähe von 20 ms verliefen. Eine relative starke Streuung ergab sich dagegen für die Trajektorienberechnung, deren gemittelter Aufwand weniger als 10 ms, in der Spitze aber auch bis zu 60 ms ausmachen konnte. Dies unterlag jedoch zwei klaren Trends: Zum einen bestand ein direkter Zusammenhang zur vermeintlichen Komplexität der zu behandelnden Situation. Nahm diese zu, so konnte dies zu längeren Laufzeiten zur Berechnung aller Trajektorien führen, da die einzelnen Aufgaben weniger leicht zu lösen waren und gegebenenfalls zusätzlicher Aufwand zur Berücksichtigung weiterer Fahrspuren entstand. Ein Beispiel hierfür ist das Überholmanöver nach ca. 210 s, welches am Ende dieses Abschnitts diskutiert wird.
Zum anderen gab es eine starke Korrelation mit der aktuellen Fahrzeuggeschwindigkeit, welche zum Vergleich in Abbildung 4.15b gezeigt wird. Am Ende des Szenarios sowie in Kurven, während Spurwechseln oder innerhalb der Wendeschleife reduzierte die Taktikautonomie, wie bereits beschrieben, automatisch die Geschwindigkeitsvorgabe für die Regelung. Gemäß (4.1) hat dies wiederum einen direkten Einfluss auf die Prädiktionsdistanz und damit auf die räumliche Länge der berechneten Trajektorien. Aufgrund der örtlichen (und nicht zeitlichen) Diskretisierung der Knoten des Hybrid A*-Algorithmus, siehe auch Abschnitt 3.4, müssen in diesen Fällen aber

im Schnitt deutlich weniger Knoten expandiert werden, um die Zielkonfiguration zu erreichen. Dies steht schließlich in starkem Zusammenhang zur resultierenden Rechenzeit, wie in Abschnitt 4.1.1 dargestellt.

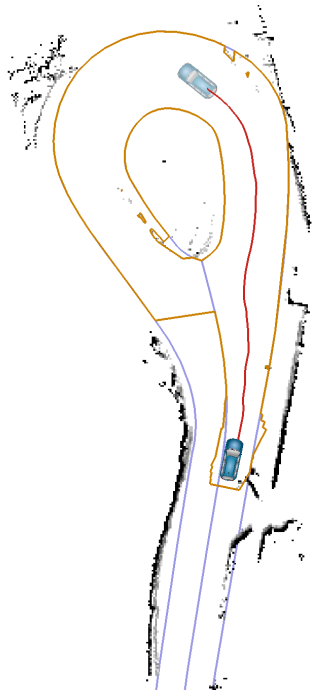
Je nach Geschwindigkeit und Situation lag die gemittelte Zeit zur Entscheidungsfindung durch die Taktikautonomie zwischen ca. 30 ms und 90 ms, und damit entsprechend unter der Zielvorgabe von 100 ms. Wird die nicht-gemittelte Rechenzeit in Abbildung 4.15c betrachtet, ist jedoch gut erkennbar, dass diese Grenze vereinzelt nicht eingehalten wurde: Insgesamt liegen 76 Aufrufe der Taktik über dem genannten Schwellenwert, was einer Quote von 2,9 % entspricht. Die Grafik zeigt ergänzend auch die jeweils benötigten Zeitaufwände der Trajektorienberechnung für beide Spuren, deren Verläufe eine deutliche Korrelation zur Rechenzeit der Taktik insgesamt aufweisen. Dies wird besonders in den neun sehr starken Abweichungen deutlich, bei denen in der Summe zwischen 220 ms und 510 ms benötigt wurden, was in allen Fällen aus langen Laufzeiten der Trajektorienberechnung resultierte. Dabei können sieben dieser Fälle darauf zurückgeführt werden, dass der Hybrid A*-Algorithmus keine Lösung zu der gegebenen Kombination aus räumlichen Beschränkungen und Zielkonfiguration im Rahmen der Beschränkung der maximal zu öffnenden Knoten $\mathbf{O}_{\max}$ finden konnte.
Da die Taktik ihre Trajektorien für ca. sechs Sekunden in die Zukunft berechnet, vergleiche t_{pred} in Tabelle A.2, können diese auch bei leichten Überschreitungen der Rechenzeitanforderung von 100 ms trotzdem noch sinnvoll als Referenz innerhalb der Regelung genutzt werden. Je größer die Abweichung nach oben ist, desto akuter wird jedoch auch die Gefahr, dass wichtige Änderungen in der Umgebungswahrnehmung nicht rechtzeitig in der Berechnung von Steuersignalen berücksichtigt werden und so gegebenenfalls ein Notstopp ausgelöst wird. Um dieses Risiko zu minimieren, sollten zusätzliche Verfahren implementiert werden, um lange Rechenzeiten des Hybrid A*-Algorithmus zu reduzieren beziehungsweise nicht lösbare Aufgaben frühzeitig zu beenden.

Abschließend wird die Trajektorienberechnung und Entscheidungsfindung der Taktikautonomie anhand von zwei exemplarischen Situationen qualitativ ausgewertet. Abbildung 4.16 zeigt dafür zunächst die Durchfahrt der Wendeschleife in der Mitte der Strecke bei **(D)**. Der Straßenverlauf hier weist insgesamt vergleichsweise hohe Krümmungen auf. Dies wurde partiell recht ungenau durch die Offline-Karte der Spurinformationen abgebildet, wie die drei ausgewählten Szenen in den Abbildungen 4.16c bis 4.16e zeigen. Neben dem Inneren der Wendeschleife betraf dies besonders stark auch deren Ein- beziehungsweise Ausfahrt. Durch die Berücksichtigung aller statischen räumlichen Beschränkungen in einem erweiterten Freibereichspolygon, wie in Abbildung 4.11 gezeigt, konnte das Fahrzeug jedoch auch an diesen Stellen sicher geführt werden. So ergab sich bereits bei der Einfahrt eine Verschiebung in Richtung der Straßenmitte, siehe Abbildung 4.16c. Bei der Ausfahrt kam zusätzlich ein Fahrzeug entgegen, welches jedoch ebenfalls deutlich durch die Trajektorie der Taktik berücksichtigt wurde, vergleiche Abbildung 4.16e. Durch die Krümmung

(a) Bild der Frontkamera bei Einfahrt in die Wendeschleife

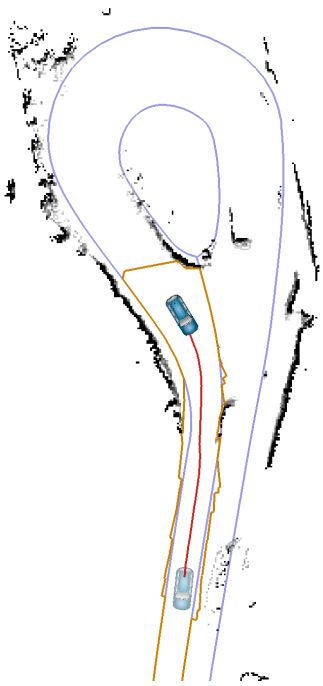


(b) Verlauf der Fahrzeuggeschwindigkeit. Die Markierungen entsprechen den im Folgenden gezeigten Ausschnitten.



(c) $t = 114\,\mathrm{s}$, entspricht Abbildung 4.16a

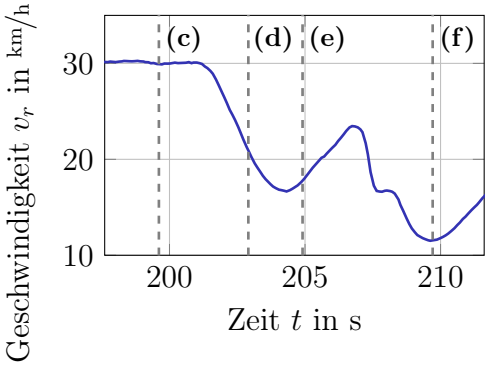
(d) $t = 121{,}5\,\mathrm{s}$



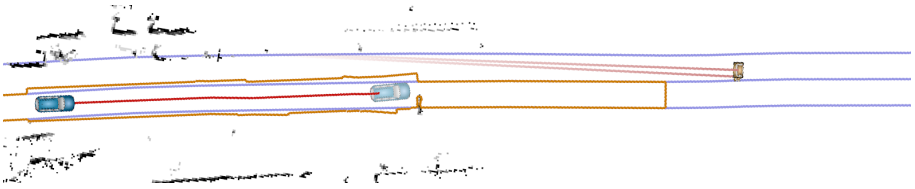
(e) $t = 134{,}5\,\mathrm{s}$, entspricht Abbildung 4.14c

Abbildung 4.16: Exemplarische Ausschnitte der Frontkamera und der Planung des Hybrid A*-Algorithmus sowie der Geschwindigkeitsverlauf des Fahrzeuges bei der autonomen Durchfahrt der Wendeschleife **(D)**. Die Farbgebung entspricht Abbildung 4.11, wobei hier in rot jeweils die Trajektorie der Taktik gezeigt wird, welche zwischen der Schätzung des Fahrzeugzustands und der zugehörigen Zielkonfiguration liegt (repräsentiert durch die beiden Fahrzeugbilder). Der Grauton einer Zelle in der Online-Karte beschreibt die Hindernis-wahrscheinlichkeit an der entsprechenden Stelle, wobei die Wahrscheinlichkeit umso höher eingeschätzt wird, je dunkler diese markiert ist.

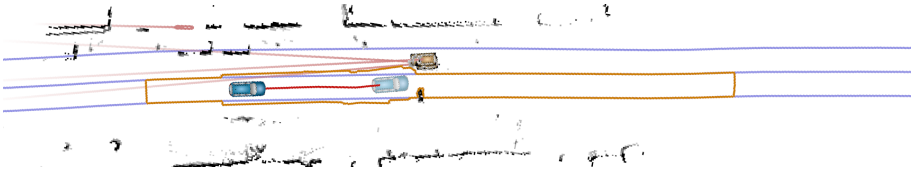
(a) Bild der Frontkamera während das Ego Fahrzeug auf den parkenden Pkw zufährt



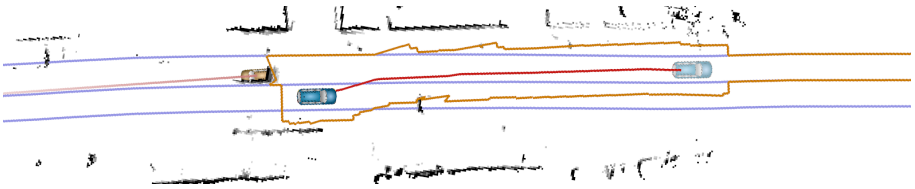
(b) Verlauf der Fahrzeuggeschwindigkeit. Die Markierungen entsprechen den im Folgenden gezeigten Ausschnitten.



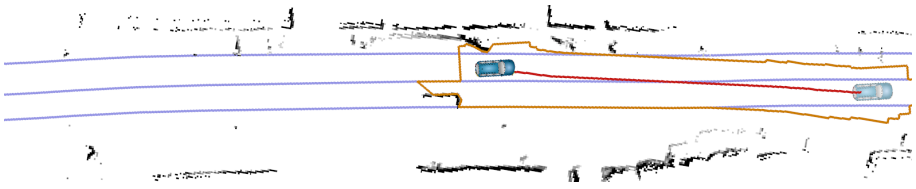
(c) $t = 199{,}6\,\mathrm{s}$, entspricht Abbildung 4.17a



(d) $t = 202{,}9\,\mathrm{s}$, entspricht Abbildung 4.14d



(e) $t = 204{,}9\,\mathrm{s}$



(f) $t = 209{,}7\,\mathrm{s}$

Abbildung 4.17: Exemplarische Behandlung eines entgegenkommenden und parkenden Pkws bei **(C)**, analog zu Abbildung 4.16. Die erkannte Dynamik wird durch ein gelbes Fahrzeugbild zusammen mit Prädiktionen seiner Bewegung in verblassendem orange repräsentiert.

innerhalb der Wendeschleife konnte diese zudem nicht sicher mit der eigentlich geltenden Höchstgeschwindigkeit von 30 km/h durchfahren werden. Dies wurde durch den Schritt zur Geschwindigkeitsnachverarbeitung in der Taktik berücksichtigt, wodurch sich diese zielgerichtet auf 12 km/h reduzierte, vergleiche Abbildung 4.16b. Aufgrund des Zusammenhangs mit der Prädiktionsdistanz aus Gleichung (4.1), wurde diese ebenfalls während der Durchfahrt der Wendeschleife deutlich kleiner, wie in Abbildung 4.16d im Vergleich gut zu sehen ist. Dies erlaubte zum einen, den Prädiktionszeithorizont t_{pred} konstant zu halten und führte zum anderen, mit Blick auf Abbildung 4.15, zu deutlich sinkenden Rechenzeiten.

Andere Anforderungen an die Taktikautonomie ergaben sich in der Situation aus Abbildung 4.17. Hier war die eigene Fahrspur durch einen geparkten Pkw blockiert; zeitgleich wurde ein mögliches Überholmanöver durch ein entgegenkommendes Fahrzeug verhindert. Da dieses durch die Bordsensorik des Fahrzeuges detektiert wurde, lagen bezüglich dessen Größe und geschätzter Bewegung teilweise hohe Unsicherheiten vor, die erst mit der Nähe zum Ego-Fahrzeug deutlich kleiner wurden, vergleiche Abbildungen 4.17c bis 4.17e. Die Szene wurde durch die Taktikautonomie richtig interpretiert, sodass zunächst die Spur gehalten und die Geschwindigkeit langsam abgebaut wurde, wie am Verlauf in Abbildung 4.17b zu sehen ist. Nachdem das entgegenkommende Fahrzeug passiert war, konnte eine Trajektorie zur benachbarten Spur berechnet werden, die (nach Freigabe durch den Sicherheitsfahrer) als Entscheidung ausgewählt und an die Regelung übergeben wurde. Sobald das stehende Fahrzeug erfolgreich umfahren wurde, wählte die Taktik abschließend als Manöver einen erneuten Wechsel zurück zur eigenen Fahrspur aus, siehe Abbildung 4.17f.

Insgesamt zeigen die Ergebnisse, dass die in Abschnitt 4.2.1 beschriebene Taktikautonomie, basierend auf den in dieser Arbeit vorgestellten Algorithmen, in der Lage ist, ohne spezielle Annahmen an die Umgebung sehr adaptive Entscheidungen zur Berücksichtigung aller räumlichen Beschränkungen und zur Anpassung von Geschwindigkeitsvorgaben zu treffen. Dabei kann ein durch die Strategieautonomie vorgegebenes, langfristiges Ziel verfolgt werden. Im Schnitt liegen zudem die benötigten Rechenzeiten deutlich unter der gestellten Anforderung von 100 ms und nur in wenigen einzelnen Situationen darüber. Hierzu sind in zukünftigen, aufbauenden Arbeiten vor allem Möglichkeiten zu finden, um sehr lange Laufzeiten des Hybrid A*-Algorithmus zu reduzieren, insbesondere, wenn unter der gegebenen Kombination aus Beschränkungen und Zielvorgaben (vermutlich) keine Lösung existiert.

4.2.3 Funktionsbeispiel: Abstandsregeltempomat (ACC)

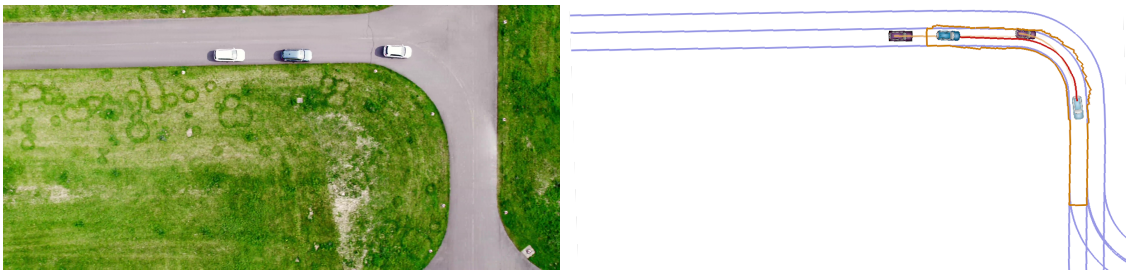
Abschließend wird gezeigt, wie die Geschwindigkeitsvorgaben der Taktikautonomie im Falle von vorausfahrenden Fahrzeugen die Funktion eines Abstandsregeltempomaten (ACC) umsetzen, wie in Abschnitt 4.2.1 beschrieben. Dies wird exemplarischen durch ein Fahrmanöver auf dem Flughafen in Jahnsdorf bei Chemnitz vorgestellt, vergleiche Abbildung 4.18. Hier fuhren mehrere autonom geführte Fahrzeuge

Abbildung 4.18: Satellitenaufnahme eines Teilbereichs des Flughafens in Jahnsdorf bei Chemnitz (© 2022 GeoBasis-DE/BKG) zusammen mit der Positionsschätzung (in rot) des Forschungsfahrzeuges während der in diesem Abschnitt beschriebenen kooperativen autonomen Fahrt. Die Detailbeschreibungen in den Abbildungen 4.20 und 4.19 beziehen sich auf den hier in gelb markierten Bereich.

hintereinander von links auf dem Zubringer in Richtung des Vorfelds, wendeten dort und endeten schließlich wieder am Ausgangsort. Andere Verkehrsteilnehmer oder sonstige Hindernisse mussten nicht berücksichtigt werden. Die Zielgeschwindigkeit aller Fahrzeuge war mit $50\,^{\mathrm{km}}/_{\mathrm{h}}$ festgelegt, welche jedoch insbesondere in Kurven durch das jeweilige autonome System deutlich reduziert werden musste. War dies der Fall, so mussten die hinterher fahrenden Fahrzeuge ebenfalls adaptiv ihre Geschwindigkeit anpassen, um einen sicheren Abstand zum Vorderfahrzeug zu gewährleisten. Vor diesem Hintergrund ist vor allem die erste, zum Vorfeld abknickende und in Abbildung 4.18 markierte Kurve interessant, da diese zu vergleichsweise großen Geschwindigkeitsänderungen führte.

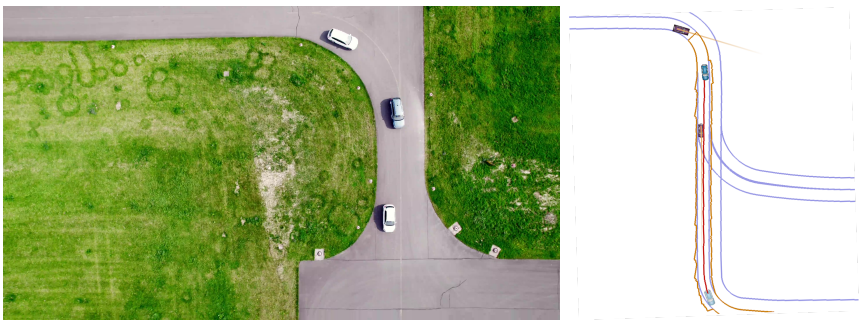
Im Folgenden wird kurz ein Szenario vorgestellt, an dem insgesamt drei autonom geführte Pkw beteiligt waren, siehe Abbildung 4.19. Um die ACC-Funktion exemplarisch darzustellen, beschränkt sich die weitere Diskussion auf die zuvor genannte erste Kurve. Neben dem Ego-Fahrzeug, welches durch das OPA^{3}L-System geführt wurde, waren zwei Fahrzeuge der IAV GmbH beteiligt, welche mit einer separaten Software zu deren autonomer Steuerung ausgestattet waren, vergleiche beispielsweise [83]. Deren Implementierungsdetails sind für die weiteren Betrachtungen im Allgemeinen nicht relevant; alle Systeme hatten aber gemein, dass sie hochfrequent Trajektorien berechneten, welche die jeweils geplante Bewegung für die nächsten paar Sekunden darstellten. Im Rahmen dieses Tests wurden diese Prognosen den jeweils anderen Fahrzeugen durch ein V2X System zur Verfügung gestellt, siehe Abschnitt 1.2, sodass diese eine sehr genaue Informationsgrundlage zur Umsetzung eines Abstandsregeltempomaten hatten. In der gezeigten Konstellation fuhr das Ego-Fahrzeug in der Mitte, sodass – aus dessen Sicht – das hinterherfahrende Fahrzeug für eine ACC-Funktion nicht relevant war. Dieses wird daher in den weiteren Diskussionen vernachlässigt. Zur Referenz der folgenden Betrachtungen ist der Geschwin-

(a) $t = 47{,}6\,\mathrm{s}$



(b) $t = 52{,}4\,\mathrm{s}$

(c) $t = 56{,}3\,\mathrm{s}$



(d) $t = 59{,}4\,\mathrm{s}$

Abbildung 4.19: Exemplarische Ausschnitte der Bewegungen des Ego-Fahrzeugs (blau) sowie von zwei kooperierenden autonomen Fahrzeugen der IAV GmbH davor und dahinter im gezeigten Teilbereich aus Abbildung 4.18. Links ist jeweils eine Bildaufnahme aus der Luft, rechts die zugehörige Wahrnehmung und Planung des Ego-Fahrzeuges zu sehen. Die Farbgebung entspricht dabei Abbildung 4.17, wobei die kooperativen Fahrzeuge in lila gezeichnet werden.

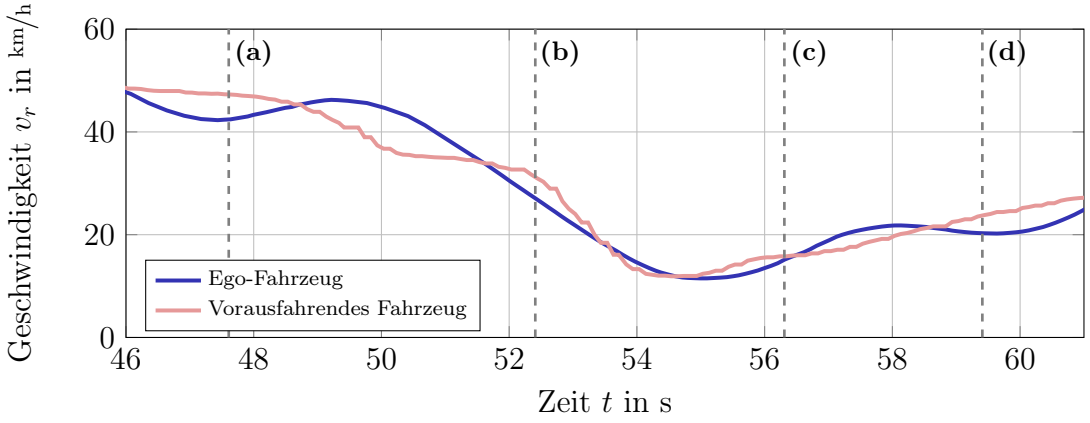


Abbildung 4.20: Verlauf der Fahrzeuggeschwindigkeiten des Ego-Fahrzeugs sowie des vorausfahrenden Fahrzeugs der IAV GmbH beim kooperativen Fahren im gezeigten Teilbereich des Jahnsdorfer Flughafens aus Abbildung 4.18. Die Markierungen entsprechen den in Abbildung 4.19 gezeigten Ausschnitten.

digkeitsverlauf des Ego-Fahrzeugs sowie des vorausfahrenden Fahrzeugs für den hier betrachteten Ausschnitt des Szenarios in Abbildung 4.20 dargestellt.

Abbildung 4.19a zeigt zunächst den Abstand des Ego-Fahrzeugs zum vorausfahrenden Fahrzeug, der sich bei der Fahrt mit 50 km/h auf dem Zubringer einstellte. Dieser betrug hier ca. 10 m*, und setzte sich aus einem allgemeinen (geschwindigkeitsabhängigen) Sicherheitsabstand sowie dem ACC-Zielabstand zusammen. Dieser wiederum basierte auf einer ACC-Zeitlücke von $t^{\mathrm{ACC}} = 0{,}6\,\mathrm{s}$, was bei der gegebenen Geschwindigkeit eine Distanz von $\mathrm{d}_{\mathrm{set}}^{\mathrm{ACC}} = 8{,}3\,\mathrm{m}$ ergab, vergleiche Abschnitt 4.2.1. Im direkten Vergleich zum empfohlenen zeitlichen Mindestabstand auf öffentlichen Straßen von 1 s, beziehungsweise 14 m, bewegten sich die Fahrzeuge damit relativ dicht hintereinander [19]; dies wurde im Rahmen des Testes bewusst so gewählt, um die Reaktionsfähigkeit der beteiligten Systeme zu demonstrieren. Da die Berechnungen der Taktikautonomie des Ego-Fahrzeuges die Bewegung des vorausfahrenden Fahrzeuges mit einbezogen, verläuft die geplante Trajektorie im rechten Teil des Bildes durch dieses hindurch.

In der Phase der Annäherung an die Kurve reduzierten beide Fahrzeuge ihre Geschwindigkeit innerhalb von ca. fünf Sekunden vergleichsweise deutlich auf nur noch 13 km/h. Durch die konservative Berechnung der Referenzgeschwindigkeit, vergleiche Abschnitt 4.2.1, wurde der Abstand in dieser Phase allerdings nur leicht abgebaut, was gut in den Abbildungen 4.19b und 4.19c zu sehen ist. Dies spiegelte sich auch in den gefahrenen Geschwindigkeiten beider Systeme wider, die insgesamt sehr dicht beieinander verliefen. Im Anschluss an die Kurve beschleunigten beide Fahrzeuge dann gleichmäßig, sodass sich auch hier kein abrupt größerer Abstand aufbaute, wie Abbildung 4.19d zeigt.

*zum Vergleich: das blaue Ego-Fahrzeug hat eine Länge von 4,8 m

Durch die sehr genauen Bewegungsinformationen des vorausfahrenden Fahrzeuges konnte die Taktikautonomie damit Zielgeschwindigkeiten vorgeben, die zu keinen ruckartigen Änderungen im Sicherheitsabstand und damit zu einem vergleichsweise angenehmen Fahrgefühl führten. Ein genauer Blick auf Abbildung 4.20 zeigt allerdings, dass die Geschwindigkeit des Ego-Fahrzeuges leichten Schwankungen unterlag und, zum Beispiel kurzzeitig nach Sekunde 47, sogar leicht wieder anstieg, obwohl das Vorderfahrzeug kontinuierlich abbremste. Eine Ursache dieses Verhaltens kann in der Verzögerung liegen, die sich zwischen Vorgabe der Entscheidung durch die Taktik und Ausführung eines zugehörigen Steuerkommandos durch das Fahrzeug einstellt. Dies beinhaltet zum einen die Zeit für die Berechnungen des modellprädiktiven Fahrreglers; diese hängt wesentlich mit der Iterationsdauer des zugehörigen Optimierers zusammen und kann im OPA^{3}L-System in Ausnahmefällen bis zu 200 ms betragen. Zum anderen ergibt sich eine feste Verzögerung bei der Umsetzung von Steuerkommandos durch die Schnittstellen des Fahrzeuges selbst. Während dies für die Lenksignale ca. 200 ms beansprucht, benötigt die Ausführung von Beschleunigungs- und Verzögerungssignalen sogar ungefähr 500 ms.
Für das gezeigte Manöver führten diese Zeitverzögerungen jedoch nur zu einem leichten Schwankungsverhalten und wirkten sich damit nicht merklich negativ auf das Fahrgefühl aus. Da jedoch zu erwarten ist, dass diese Art von Effekten sich bei stärkeren Geschwindigkeitsänderungen intensivieren, sollten zukünftige Arbeiten bekannte Latenzen im System in der Bewegungsplanung der Taktikautonomie berücksichtigen.

Kapitel 5

Resümee

Im Folgenden wird zunächst eine kurze Zusammenfassung der wichtigsten Inhalte und wesentlichsten Ergebnisse dieser Arbeit vorgestellt. Abschließend werden, basierend auf den Erkenntnissen der vorangegangenen Kapitel, verschiedene Aspekte diskutiert, die als Grundlage für eine theoretische und praktische Weiterentwicklung der vorgestellten Algorithmen dienen können.

5.1 Zusammenfassung

Die Automatisierung und Autonomisierung von Fahrzeugen stellt eine bedeutende technische Entwicklung unserer Zeit dar. Insbesondere für Pkw und Lkw wird kontinuierlich an neuen Assistenzsystem gearbeitet, welche das Fahren zum Beispiel sicherer oder effizienter machen. Im Jahr 2022 wurde in diesem Kontext mit dem DRIVE PILOT von Mercedes-Benz weltweit erstmalig ein automatisiertes System der SAE-Stufe 3 zur Marktreife gebracht, welches in Stausituationen auf Autobahnen unter bestimmten Bedingungen die Verantwortung für Führung des Pkws übernehmen kann [67]. Ungeachtet dieser Fortschritte ist die Entwicklung vollkommen autonom fahrender Fahrzeuge weiterhin Gegenstand intensiver Forschung und Entwicklung; insbesondere städtische Szenarien stellen aufgrund der hohen Vielfalt an möglichen, schwer vorhersehbaren Situationen eine große Herausforderung dar. Vor diesem Hintergrund wird im Forschungsprojekt OPA^{3}L unter anderem an der Umsetzung und realen Erprobung neuer algorithmischer Ansätze für autonome Fahrfunktionen in (sub-)urbanen Anwendungsgebieten gearbeitet. Dazu gehört beispielsweise auch eine Komponente zur Umsetzung einer Taktikautonomie. Deren Aufgabe ist das Treffen von unmittelbaren Manöverentscheidungen, um einerseits ein langfristig vorgegebenes Ziel zu erreichen und andererseits das Fahrzeug möglichst sicher und für den Fahrgast angenehm zu führen. Dazu werden zunächst Manöveroptionen

durch einen speziell konstruierten Bewegungsplaner berechnet und anschließend eine Entscheidung durch Auswahl des vermeintlich besten Manövers getroffen.

Die vorliegende Arbeit führte den *generischen Hybrid A*-Algorithmus* als Umsetzung dieses Bewegungsplaners ein. Dieser kann als Erweiterung des originalen Hybrid A*-Verfahrens aus [25] verstanden werden, welcher die Ideen und Eigenschaften des knotenbasierten A*-Suchalgorithmus mit einem dynamischen Fahrzeugmodell koppelt, und so dessen mögliche Bewegungen abbildet. Während in der Literatur lediglich eine qualitative Darstellung des Hybrid A*-Konzeptes existiert, führte diese Arbeit erstmalig eine formalisierte Beschreibung in Algorithmus 2.3 ein. Auch wenn hier ebenfalls die Steuerung eines Pkws als Anwendungsfall betrachtet wurden, so erlaubt die eingeführte Notation die Übertragung der Ideen auf beliebige dynamische Systeme. Mit Satz 2.39 konnten zudem Voraussetzungen genannt werden, mit denen sich die theoretischen Resultate bezüglich Optimalität des ordinären A*-Algorithmus ebenfalls auf den Hybrid A* übertragen; insbesondere war hier die Endlichkeit der betrachteten hybriden Knotenmenge zu nennen.

Aufbauend auf dieser Formalisierung wurde im weiteren Verlauf die Verallgemeinerung zum generischen Hybrid A*-Algorithmus vorgestellt, die im Kern auf drei Komponenten beruht: Erstens wurde die Modellierung von statischen Beschränkungen für die Planung durch Freibereichspolygone vorgestellt, wodurch die Abstrahierung verschiedenster Hindernisarten ermöglicht wurde. Da diese sehr effizient und adaptiv erstellt werden können, konnte zudem gezeigt werden, dass schon bei moderaten Datenmengen wesentliche Geschwindigkeitsvorteile im Rahmen der Kollisionsüberprüfung entstehen. Freibereichspolygone bewirken zudem eine Einschränkung an den räumlichen Suchhorizont, wodurch sie zur Endlichkeit der hybriden Knotenmenge beitragen und damit eine wichtige Voraussetzung für die Vollständigkeit und Optimalität des Suchalgorithmus darstellen.
Die zweite Erweiterung betraf die Berücksichtigung dynamischer Komponenten in der Umgebung des betrachteten Ego-Fahrzeuges. Dazu wurde vorgestellt, wie auch die prädizierte Bewegung von beweglichen Objekten in der Kollisionsüberprüfung von hybriden Knoten beachtet wird. Als wesentliche Voraussetzung dafür wurde ein Modell der Systemdynamik des Fahrzeuges verwendet, welches, im Gegensatz zum einfachen Hybrid A*-Verfahren, ebenfalls Geschwindigkeitsinformationen beinhaltet.
Die letzte und umfassendste Erweiterung bezieht sich auf die Berechnung von Voronoi-Pfaden und deren Einsatz in einem verallgemeinerten Voronoi-Potentialfeld. Im Gegensatz zur nachgelagerten Einbindung, wie in [25] vorgestellt, wird in dieser Arbeit das Voronoi-Potential direkt innerhalb des Suchalgorithmus verwendet. Dies vereinfachte nicht nur das Gesamtverfahren, sondern führte auch zu reduzierten Rechenzeiten aufgrund von starken Synergieeffekten in der Überprüfung von Kollisionen zwischen explorierten Knoten und der Fahrzeugumgebung. Die Generierung der Voronoi-Pfade selbst wurde dabei sehr effizient basierend auf der Umgebungsbeschreibung durch Freibereichspolygone umgesetzt. Darauf aufbauend wurde das Voronoi-Potential zudem so verallgemeinert, dass ebenfalls die Bewegung dynami-

scher Objekte abgebildet werden kann. Um die Erstellung der dafür notwendigen dynamischen Voronoi-Pfade auch bei eingeschränkter Rechenkapazität möglich zu machen, wurde mit Algorithmus 3.2 schließlich ein Verfahren eingeführt, mit dem das zugrunde liegende Voronoi-Diagramm äußerst effizient an sich lokal verändernde Hindernisinformationen angepasst wird. Die Eigenschaft von Voronoi-Pfaden als abstandsmaximierende Referenz zu den modellierten räumlichen Beschränkungen konnte schließlich auch außerhalb des Voronoi-Potentials als Grundlage für eine Umgebungsanalyse und als Entscheidungshilfe dienen. Ein Beispiel dazu wurde in Algorithmus 3.3 vorgestellt, mit dem automatisiert Fahrzeugkonfigurationen als Teil des Zielzustandes für das Hybrid A*-Suchverfahren bestimmt werden können.

Der generische Hybrid A* wurde schließlich als Kombination der genannten Verfahren in Algorithmus 3.4 für den Anwendungsfall des autonomen Fahrens beschrieben. Dieser wurde so konzipiert, dass er auch für unstrukturierte Umgebungsdaten und ohne besonderes Vorwissen Trajektorien berechnen kann, die sich vor allem durch hohe Sicherheit kennzeichnen. Anhand von simulierten Szenarien unterschiedlicher Komplexität wurde die Leistungsfähigkeit dieses Verfahrens ausführlich qualitativ und quantitativ untersucht. Dabei konnte gezeigt werden, dass sich selbst komplizierte Probleme in der Regel in deutlich unter 100 ms lösen lassen. Insbesondere wurde in diesem Kontext auch dargestellt, dass die (aufwändigere) Verwendung des verallgemeinerten Voronoi-Potentials gegenüber einfacheren Modellierungsansätzen einen spürbar positiven Einfluss auf die Qualität der berechneten Lösungen haben kann – insbesondere in komplexen Situationen.

Abschließend wurde vorgestellt, wie der generische Hybrid A*-Algorithmus zu einem Modul für taktische Entscheidungen in einem autonom fahrenden Fahrzeug verallgemeinert werden kann. Dieses wurde im Rahmen des OPA^{3}L-Systems auf einem realen Forschungsfahrzeug eingesetzt und anhand von zwei unterschiedlichen Fahrten – im normalen Straßenverkehr eines Bremer Vororts sowie auf einem Flughafengelände – ausgewertet. Die Ergebnisse zeigen deutlich, dass die in dieser Arbeit präsentierten Algorithmen in der Lage waren, die Umgebung des autonomen Fahrzeugs stets sinnvoll zu analysieren und so sichere und komfortable Entscheidungen zu treffen. Dies galt insbesondere auch in vergleichsweise schwierigen Situationen, in denen das Ego-Fahrzeug sich an die Bewegung anderer Verkehrsteilnehmer anpassen musste. Eine wesentliche Komponente stellte in diesem Zusammenhang wieder die Auswertung von Voronoi-Pfaden dar, durch welche Engstellen entlang des Straßenverlaufs oder Abstände zu vorausfahrenden Fahrzeugen sehr robust erkannt und berücksichtigt werden konnten. Insgesamt wurden zudem auch in diesen Realversuchen die Rechenzeitanforderung von 100 ms im Mittel eingehalten. Vor allem in Ausnahmesituationen, in welchen durch den Hybrid A*-Algorithmus keine Lösung berechnet werden konnte, ergaben sich jedoch zum Teil sehr hohe Abweichungen von diesem Zielwert; diese traten allerdings selten genug auf, um keinen spürbaren Einfluss auf das Fahrverhalten des autonomen Pkws zu nehmen.

Insgesamt zeigen sowohl die simulativen Ergebnisse als auch der erfolgreiche Einsatz im realen Straßenverkehr, dass der in dieser Arbeit vorgestellte generische Hybrid A*-Algorithmus hohen Anforderungen an Effizienz und Adaptivität genügt.

5.2 Ausblick

Während der Entwicklung und Auswertung der in dieser Arbeit vorgestellten Algorithmen wurden an verschiedenen Stellen Potentiale für weitere Verbesserungen angedeutet. Dies betraf vor allem die folgenden Aspekte:

1. Theoretische und algorithmische Erweiterung des Algorithmus 3.2 zum lokalen Voronoi-Update.
2. Mögliche Verbesserungen durch eine genauere Fahrzeugmodellierung.
3. Behandlung von Laufzeitspitzen des Hybrid A*-Algorithmus.

Der Algorithmus 3.2 zum lokalen Voronoi-Update stellte als Grundlage für die effiziente Berechnung dynamischer Voronoi-Pfade einen zentralen Beitrag dieser Arbeit dar. In Abschnitt 3.3.5 wurde er basierend auf der Tatsache motiviert, dass nur eine örtlich begrenzte Menge an Eingabedaten zur Definition einer bestimmten Kante eines Voronoi-Diagramms beiträgt. Diese Erkenntnis wurde schließlich ausgenutzt, um Schritte abzuleiten, mit denen sich lokale Änderungen der Voronoi-Kanten beim Hinzufügen weiterer Datenpunkte berechnen lassen. Es wurde exemplarisch die Korrektheit des vorgestellten Verfahrens gezeigt; weiterführende Arbeiten sollten jedoch einen formalen Korrektheitsbeweis liefern und Voraussetzungen beziehungsweise Grenzen der Methode benennen. Zudem könnte der Algorithmus inhaltlich erweitert werden, um zum Beispiel auch das Entfernen von Datenpunkten zu behandeln.
Die deutliche Effizienzsteigerung des Ansatzes bei geringer Parallelisierung wurde anhand eines Testszenarios mit einem beweglichen Objekt numerisch gezeigt. Für einen hohen Grad an parallelen Prozessen ergab sich allerdings ein leichter Mehraufwand. Zudem wurde die Erwartung benannt, dass sich bei einer großen Menge an zu aktualisierenden Voronoi-Kanten (zum Beispiel bei vielen dynamischen Objekten) ebenfalls Rechenzeitnachteile ergeben könnten. Um hier eine a-priori Abschätzung treffen zu können, sollte der Algorithmus 3.2 weiterführend analysiert werden. Ein mögliches Resultat könnte eine Heuristik sein, nach der situationsbedingt abgeschätzt wird, ob sich der Einsatz des lokalen Voronoi-Updates gegenüber der kompletten Neuberechnung des Voronoi-Diagramms lohnt.

Punkt 2 ergab sich zum Beispiel aus den Erkenntnissen der numerischen Auswertung in Kapitel 4. Dabei zeigten zum einen die Ergebnisse in der Simulation, dass die

durch den Hybrid A*-Algorithmus berechneten Trajektorien zwar zu glatten Verläufen in der Position führen, insbesondere die Steuerwerte jedoch sehr unkontinuierlich verlaufen können, vergleiche Abschnitt 4.1.1.1. Dies konnte sich im kleineren Ausmaß auch auf von diesen abhängige Zustandsvariablen übertragen. Zum anderen ergab sich aus den Beobachtungen im Zusammenhang mit dem Forschungsfahrzeug in Abschnitt 4.2.3, dass zum Beispiel nicht berücksichtigte Zeitverzögerungen des Gesamtsystems im Rahmen der Planung in Schwankungen der gefahrenen Geschwindigkeit resultieren können. Ein letzter Kritikpunkt betrifft direkt das verwendete Einspurmodell. Wie in Abschnitt 3.1 beschrieben, ist dieses generell nur für geringe Querbeschleunigungen anwendbar, was die Einsatzbereiche des vorgestellten Verfahrens einschränkt.
Alle genannten Aspekte könnten durch eine Erweiterung der Kinematikmodellierung behandelt werden. Eine einfache Verbesserung in diesem Kontext wird zum Beispiel in [31] zur Berechnung von optimierten Trajektorien mit Methoden der nichtlinearen Optimierung vorgestellt. Dabei werden zusätzliche Ableitungsebenen in den Steuergrößen eingeführt, um so die eigentlich relevanten Steuerwerte (als deren Integrale) zu glätten. Zudem werden bekannte Zeitverzögerungen in den Steuerungen durch zusätzliche Differentialgleichungen in Form von PT1-Gliedern approximiert [99]. Um den Einsatzbereich des Modells auf insgesamt höhere Geschwindigkeiten zu erweitern, könnten zudem Komponenten zur Modellierung von auf die Reifen wirkenden Kräften wie in [84] hinzugefügt werden.
In allen Fällen würde das modellierte System durch zusätzliche Zustandsvariablen vergrößert werden, was beim Einsatz in einem Hybrid A*-Algorithmus zu entsprechend höherdimensionalen Knoten führen kann. Dies lässt einen gesteigerten Suchaufwand erwarten, sodass entsprechende Erweiterungen der Fahrzeugmodellierung in weiterführenden Arbeiten genau untersucht und sorgsam abgewogen werden sollten.

Eine weitere Beobachtung der numerischen Auswertung aus Kapitel 4 sind die in Punkt 3 genannten vereinzelten Laufzeitspitzen des hybriden Suchverfahrens. Diese traten sowohl bei komplexen, dynamischen Problemen in der simulationsbasierten Auswertung auf, vergleiche Abschnitt 4.1.1.2, als auch vereinzelt im Fall von nicht lösbaren Problemen auf dem Realfahrzeug, siehe Abschnitt 4.2.2. Eine Verbesserung in diesen Spezialfällen könnte durch eine Adaption der Konzepte aus der Klasse der *Anytime A**-Algorithmen auf den hybriden Rahmen umgesetzt werden. Deren Kernidee ist es, die A*-Suche zunächst in einer deutlich vereinfachten Variante der betrachteten Aufgabe durchzuführen, um möglichst schnell ein zulässiges (aber gegebenenfalls suboptimales) Ergebnis zu erhalten. In weiteren Iterationen wird die Vereinfachung schrittweise reduziert, bis schließlich das eigentliche Problem gelöst wird. Sollte zum Beispiel die verfügbare Rechenzeit vorzeitig erreicht werden, so steht zumindest das bis dahin beste Ergebnis zur Verfügung. Insbesondere stellt dieses Vorgehen auch einen Ansatz dar, um die mögliche Unlösbarkeit eines gegebenen Problems frühzeitig zu erkennen. Eine konkrete Implementierungsoption stellt

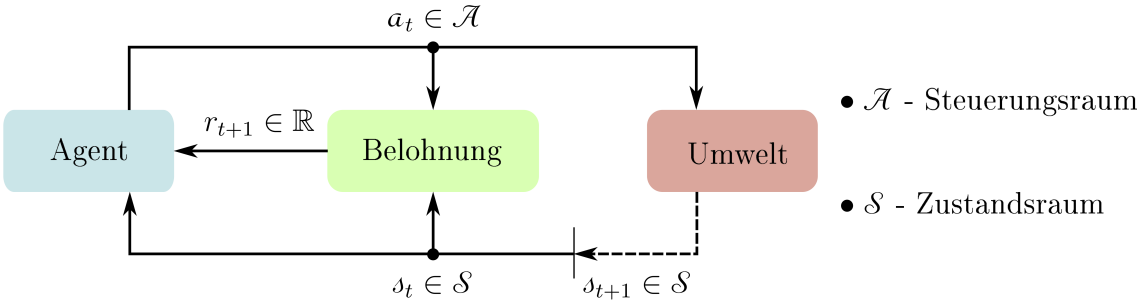


Abbildung 5.1: Wechselwirkung der drei Hauptelemente des Reinforcement Learning, aus [30]

der *gewichtete Anytime A**-Algorithmus dar [42]. Dieser basiert auf der Erkenntnis aus Abschnitt 2.4.1, nach der eine suboptimale Lösung in reduzierter Anzahl von Iterationen durch Skalierung der Heuristik gefunden werden kann. Ein alternatives Anytime-Verfahren könnte aus dem Ansatz aus Abschnitt 4.1.1.2 abgeleitet werden. Danach würde zunächst mit einer deutlich gröberen Diskretisierung der hybriden Knoten gerechnet werden, um diese im weiteren Verlauf schrittweise zu verfeinern. Beide Vorschläge würden zu einer Steigerung des Aufwands zum Lösen einzelner Probleme führen, dafür aber Möglichkeiten bereitstellen, unlösbare oder sehr aufwändige Fälle besser zu handhaben.

Beim Einsatz des Hybrid A*-Algorithmus in einer festen Anwendung (zum Beispiel dem autonomen Fahren) könnten bestimmte Aspekte der betrachteten Suchprobleme auch datenbasiert durch ein Modell gelernt werden. Dieses könnte, abhängig von den verfügbaren Umgebungsinformationen und der jeweiligen Suchiteration, Prognosen über die Lösbarkeit der betrachteten Aufgabe treffen. Eine weitere Möglichkeit wäre, als Teil der Suchverfahrens die Auswahl der zu expandierenden Knoten zu steuern und so die Anzahl der nötigen Iterationen zu verringern. Dieser Ansatz könnte unter anderem durch den Einsatz des *Reinforcement Learning* verfolgt werden, vergleiche Abbildung 5.1. Bei diesem wird ein *Agent* trainiert sich so zu verhalten, dass seine Aktionen zu einer positiven Interaktion mit seiner *Umwelt* führen. Die Bewertung dieser Wechselbeziehung erfolgt durch eine *Belohnungsfunktion* [95]. Beim Einsatz in einem (hybriden) A*-Algorithmus könnte der Agent beispielsweise als Teil der Heuristikfunktion zur Bewertung eines Knoten im Kontext der vorhandenen Umgebungsinformationen beitragen. Die Umwelt würde dann das A*-Verfahren selber darstellen, welches, basierend auf den jeweils erwarteten Gesamtkosten der Knoten, zur Expansion des nächsten Elementes führt. Die Belohnungsfunktion könnte schließlich die notwendige Menge an Iterationen bestrafen, um das Verfahren auf diese Weise insgesamt zu beschleunigen. Das Reinforcement Learning führt in der Regel zu Agenten, die durch ihre Aktionen auch sehr langfristige Ziele verfolgen und so auch im Rahmen von tausenden von Iterationen eines Suchverfahrens eingesetzt werden können. Zudem wurde bereits gezeigt, dass ähnliche Ansätze im Kontext von Lösungsroutinen für nichtlineare Optimierungsprobleme zu aussichts-

reichen Resultaten führen [56]. Somit stellt die Übertragung auf den generischen Hybrid A*-Algorithmus eine vielversprechende Forschungsrichtung als Fortsetzung dieser Arbeit dar.

Anhang A

Verwendete Hyperparameter

Im Folgenden werden die in den numerischen Auswertungen verwendeten Hyperparameter dargestellt. Dabei zeigt Tabelle A.1 die in den simulationsbasierten Experimenten genutzten Werte aus Kapitel 3 sowie Abschnitt 4.1. Tabelle A.2 listet die Parameter der realen Evaluationsfahrten aus Abschnitt 4.2 auf.

Tabelle A.1: Verwendete Hyperparameter der simulationsbasierten Experimente aus Kapitel 3 sowie Abschnitt 4.1

Beschreibung	Bezeichnung	Wert	Einheit
Generierung von Freibereichspolygonen			
Expansionsweite	κ	40,0	m
Lokale Expansionstiefe	κ_ρ	24,0	m
Expansionsdistanz	ξ	12,0	m
Voronoi-Potential			
Distanz-Schwellenwert für Voronoi-Pfade	$\delta_{\mathcal{R}}$	1,0	m
Parallele Kerne für dynamische Voronoi-Pfade	–	4	–
Potentialverlauf	α	1000,0	–
Maximal berücksichtigte Distanz	$\mathrm{d}^{\max}$	4,0	m
Diskretisierungen			
Räumliche Diskretisierung	δ_p	0,5	m
Orientierungsdiskretisierung	δ_φ	0,1	rad
Geschwindigkeitsdiskretisierung	δ_{v_r}	0,5	$\mathrm{m/s}$
Zeitdiskretisierung	δ_t	0,3	s
Diskrete Beschleunigungen	$\overline{U}_{a_r}$	$\{-1{,}2,\ -0{,}6,\ 0,\ 0{,}6,\ 1{,}2\}$	$\mathrm{m/s^2}$
Diskrete Lenkwinkel	$\overline{U}_{\delta_f}$	$\{-0{,}55,\ -0{,}275,\ 0,\ 0{,}275,\ 0{,}55\}$	rad

Beschreibung	Bezeichnung	Wert	Einheit
Ego-Fahrzeug			
Radstand	l	2,786	m
Länge	Δ_l	4,767	m
Breite	Δ_b	2,083	m
Mittelpunkte der Kreisüberdeckung	$\psi_{\text{Ego}}^{(1)}$	$(-0{,}039,\ 0{,}0)$	m
(relativ zur Hinterachse)	$\psi_{\text{Ego}}^{(2)}$	$(0{,}983,\ 0{,}0)$	m
	$\psi_{\text{Ego}}^{(3)}$	$(2{,}005,\ 0{,}0)$	m
	$\psi_{\text{Ego}}^{(4)}$	$(2{,}728,\ 0{,}0)$	m
Radien der Kreisüberdeckung	ψ_{Ego}^{R}	1,022	m
Kantenkosten des hybriden Graphen			
Minimale Geschwindigkeit für Kantenkosten	$v_{\text{set, min}}$	5,0	$\mathrm{km/h}$
Gewicht Kantenkosten Distanz	ω_p	0,25	–
Gewicht Kantenkosten Geschwindigkeit	ω_v	1,0	–
Gewicht Kantenkosten Voronoi-Potential	$\omega_{\mathcal{R}}$	2,0	–

Tabelle A.2: Änderungen und Ergänzungen in den verwendeten Hyperparametern im Vergleich zu Tabelle A.1 für die realen Evaluationsfahrten aus Abschnitt 4.1

Beschreibung	Bezeichnung	Wert	Einheit
Generierung von Freibereichspolygonen (abhängig von aktueller Prädiktionsdistanz d^*)			
Expansionsweite	κ	d^*	m
Lokale Expansionstiefe	κ_ρ	$\mathrm{d}^* \cdot 3/5$	m
Expansionsdistanz	ξ	$\mathrm{d}^* \cdot 3/10$	m
Diskretisierungen			
Diskrete Beschleunigungen	$\overline{U}_{a_r}$	$\{-1{,}8,\ 0,\ 1{,}8\}$	m/s^2
Kantenkosten des hybriden Graphen			
Gewicht Kantenkosten Distanz	ω_p	1,0	–
Taktikautonomie			
Prädiktionszeithorizont	t_{pred}	6,0	s
Minimale Prädiktionsdistanz	$\mathrm{d}^*_{\mathrm{min}}$	20,0	m
Maximale Radialbeschleunigung	a_r^{rad}	0,4	m/s^2
Stabilisierende Zielpfadgewichtung	ω_{stab}	1,0	–
Expandierende Zielpfadgewichtung	ω_{expand}	3,0	–
Maximale Anzahl geöffneter Knoten	$\mathbf{O}_{\mathrm{max}}$	50 000	–

Literaturverzeichnis

[1] 5G Automotive Vision. The 5G Infrastructure Public Private Partnership, 2015. White Paper.

[2] J. Agirrebeitia et al.: A new APF strategy for path planning in environments with obstacles. *Mechanism and Machine Theory*, Bd. 40(6), S. 645–658, 2005.

[3] H. An und J. il Jung: Decision-Making System for Lane Change Using Deep Reinforcement Learning in Connected and Automated Driving. *Electronics*, Bd. 8(5), S. 543–555, 2019.

[4] S. G. Anavatti, S. L. Francis und M. Garratt: Path-planning modules for Autonomous Vehicles: Current status and challenges. In *2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation*, S. 205–214. IEEE, 2015.

[5] S. Aradi: Survey of Deep and Reinforcement Learning and for Motion and Planning of Autonomous and Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, Bd. 23(2), S. 740–759, 2022.

[6] A. Armand, D. Filliat und J. Ibañez-Guzman: Ontology-Based Context Awareness for Driving Assistance Systems. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, S. 227–233. IEEE, 2014.

[7] M. Atagoziyev, K. W. Schmidt und E. G. Schmidt: Lane Change Scheduling for Autonomous Vehicles. In *14th IFAC Symposium on Control in Transportation Systems*. 2016.

[8] J. Becker et al.: Bosch's Vision and Roadmap Toward Fully Autonomous Driving. In G. Meyer und S. Beiker, Hrsg., *Road Vehicle Automation*, S. 49–59. Springer, 2014.

[9] S. Beiker: Deployment Scenarios for Vehicles with Higher-Order Automation. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 193–211. Springer Berlin Heidelberg, 2016.

[10] S. Beiker: Implementation of an Automated Mobility-on-Demand System. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 277–295. Springer Berlin Heidelberg, 2016.

[11] R. E. Bellman: On a Routing Problem. *Quarterly of Applied Mathematics*, Bd. 16(1), S. 87–90, 1958.

[12] K. Berntorp und S. D. Cairano: Joint Decision Making and Motion Planning for Road Vehicles Using Particle Filtering. In *2016 IFAC Symposium on Advances in Automotive Control*, Bd. 49, S. 175–181. Elsevier BV, 2016.

[13] T. Blechmann: Boost.Heap, 2016. URL https://www.boost.org/doc/libs/1_63_0/doc/html/heap.html. Abgerufen am 15.07.2021.

[14] M. Blundell und D. Harty: *The Multibody Systems Approach to Vehicle Dynamics*. Elsevier, 2. Aufl., 2015.

[15] Gesetz zum autonomen Fahren tritt in Kraft. Bundesministerium für Digitales und Verkehr, 2021. URL https://www.bmvi.de/SharedDocs/DE/Artikel/DG/gesetz-zum-autonomen-fahren.html. Abgerufen am 02.10.2022.

[16] Forschung für autonomes Fahren. Bundesministerium für Wirtschaft und Klimaschutz, 2020. URL https://www.bmwk.de/Redaktion/DE/Schlaglichter-der-Wirtschaftspolitik/2020/03/kapitel-1-5-forschung-fuer-autonomes-fahren.html. Abgerufen am 27.09.2022.

[17] M. Buehler, K. Iagnemma und S. Singh: *The 2005 DARPA Grand Challenge: The Great Robot Race*, Bd. 36. Springer Publishing Company, Incorporated, 1. Aufl., 2007.

[18] M. Buehler, K. Iagnemma und S. Singh: *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, Bd. 56. Springer Publishing Company, Incorporated, 1. Aufl., 2009.

[19] Sicherheitsabstand: Welchen Abstand schreibt die StVO vor? Bußgeldkatalog, 2022. URL https://www.bussgeldkatalog.org/sicherheitsabstand/?utm_source=google&utm_medium=cpc_search&utm_term=sicherheitsabstandauto&utm_content=bussgeldkatalog.org-search&utm_campaign=c-441889508&gclid=CjwKCAjwquWVBhBrEiwAt1KmwjCMcxrZHMeDR1AszxYPrTjDwt85fDrEC5oW-Ge_H-qWvZU3z-7rvhoC7rAQAvD_BwE. Abgerufen am 12.09.2022.

[20] T. H. Cormen et al.: *Algorithmen - Eine Einführung*. Oldenbourg Verlag München, 4. Aufl., 2009.

[21] P. de Beaucorps et al.: Decision-making for automated vehicles at intersections adapting human-like behavior. In *IEEE Intelligent Vehicles Symposium*. 2017.

[22] R. Dechter und J. Pearl: Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM*, Bd. 32(3), S. 505–536, 1985.

[23] E. D. Dickmanns: Vehicles Capable of Dynamic Vision. In *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence*, Bd. 2, S. 1577–1592. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[24] E. W. Dijkstra: A note on two problems in connexion with graphs. *Numerische Mathematik 1*, S. 269–271, 1959.

[25] D. Dolgov et al.: Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *The International Journal of Robotics Research*, Bd. 29(5), S. 485–501, 2010.

[26] A. Dosovitskiy et al.: CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, S. 1–16. 2017.

[27] L. E. Dubins: On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, Bd. 2(3), S. 497–516, 1957.

[28] J. Erickson: The Jordan Polygon Theorem. Vorlesungsskript, 2009.

[29] C. Ericson: *Real-Time Collision Detection*. CRC Press, 2005.

[30] A. Folkers: *Steuerung eines autonomen Fahrzeugs durch Deep Reinforcement Learning*. Springer Spektrum, Wiesbaden, 1. Aufl., 2019.

[31] A. Folkers, M. Rick und C. Büskens: Controlling an Autonomous Vehicle with Deep Reinforcement Learning. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019.

[32] A. Folkers, M. Rick und C. Büskens: Time-Dependent Hybrid-State A* and Optimal Control for Autonomous Vehicles in Arbitrary and Dynamic Environments. In *Proceedings of the 21st IFAC World Congress*, Bd. 53, S. 15 077–15 083. 2020.

[33] A. Folkers et al.: The OPA^{3}L System and Testconcept for Urban Autonomous Driving. In *Proceedings of the 25th IEEE International Conference on Intelligent Transportation Systems*. 2022.

[34] L. R. Ford: Network flow theory. The Rand Corporation, 1956. Paper P-923.

[35] S. Fortune: A Sweepline Algorithm for Voronoi Diagrams. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, S. 313–322. ACM, New York, NY, USA, 1986.

[36] E. Fraedrich und B. Lenz: Societal and Individual Acceptance of Autonomous Driving. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 621–640. Springer Berlin Heidelberg, 2016.

[37] M. Fredman et al.: The pairing heap: A new form of self-adjusting heap. *Algorithmica*, Bd. 1(1), S. 111–129, 1986.

[38] M. L. Fredman und R. E. Tarjan: Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the Association for Computing Machinery*, Bd. 34(3), S. 596–615, 1987.

[39] T. M. Gasser: Fundamental and Special Legal Questions for Autonomous Vehicles. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 523–551. Springer Berlin Heidelberg, 2016.

[40] E. Gilbert, D. Johnson und S. Keerthi: A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, Bd. 4(2), S. 193–203, 1988.

[41] T. Gu et al.: Automated Tactical Maneuver Discovery, Reasoning and Trajectory Planning for Autonomous Driving. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, S. 5474–5480. 2016.

[42] E. A. Hansen und R. Zhou: Anytime Heuristic Search. *Journal of Artificial Intelligence Research*, Bd. 28, S. 267–297, 2007.

[43] P. E. Hart, N. J. Nilsson und B. Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, Bd. 4(2), S. 100–107, 1968.

[44] D. Heinrichs: Autonomous Driving and Urban Land Use. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 213–231. Springer Berlin Heidelberg, 2016.

[45] D. A. Hennessy und D. L. Wiesenthal: Traffic congestion, driver Stress, and driver aggression. *Aggressive Behavior*, Bd. 25(6), S. 409–423, 1999.

[46] C.-J. Hoel, K. Wolff und L. Laine: Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, S. 1563–1569. 2020.

[47] R. Horowitz, C.-W. Tan und X. Sun: An Efficient Lane Change Maneuver for Platoons of Vehicles in an Automated Highway System. Techn. Ber., California Path Program, Institute of Transportation Studies, University of California, Berkeley, 2004.

[48] B. Houska, H. J. Ferreau und M. Diehl: ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, Bd. 32(3), S. 298–312, 2011.

[49] J. Hromkovič: *Theoretische Informatik: Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Krytographie*. Springer Vieweg, 5. Aufl., 2014.

[50] J. Ji et al.: Path Planning and Tracking for Vehicle Collision Avoidance Based on Model Predictive Control With Multiconstraints. *IEEE Transactions on Vehicular Technology*, Bd. 66(2), S. 952–964, 2017.

[51] A. Johnson: Clipper-an open source freeware library for clipping and offsetting lines and polygons, 2014. URL http://www.angusj.com/delphi/clipper.php. Abgerufen am 11.10.2021.

[52] A. Kanaris, E. B. Kosmatopoulos und P. A. Loannou: Strategies and spacing requirements for lane changing and merging in automated highway systems. *IEEE Transactions on Vehicular Technology*, Bd. 50(6), S. 1568–1581, 2001.

[53] C. Katrakazas et al.: Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, Bd. 60, S. 416–442, 2015.

[54] M. Knauer und C. Büskens: Real-Time Optimal Control using TransWORHP and WORHP Zen. In G. Fasano und J. D. Pintér, Hrsg., *Modeling and Optimization in Space Engineering*, Bd. 144, S. 211–232. Springer Optimization and Its Applications, 2019.

[55] S. Kolski et al.: Autonomous Driving in Structured and Unstructured Environments. In *2006 IEEE Intelligent Vehicles Symposium*, S. 558–563. 2006.

[56] R. Kuhlmann: Learning to steer nonlinear interior-point methods. *EURO Journal on Computational Optimization*, Bd. 7(4), S. 381–419, 2019.

[57] D.-K. Kye, S.-W. Kim und S.-W. Seo: Decision Making for Automated Driving at Unsignalized Intersection. In *Proceedings of the 15th International Conference on Control, Automation and Systems*. Institute of Control, Robotics and Systems - ICROS, 2015.

[58] S. M. LaValle: Rapidly-Exploring Random Trees: A New Tool for Path Planning. Techn. Ber., Department of Computer Science, Iowa State University, 1998.

[59] U. Lee et al.: Local path planning in a complex environment for self-driving car. In *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*, S. 445–450. IEEE, 2014.

[60] B. Lenz und E. Fraedrich: New Mobility Concepts and Autonomous Driving: The Potential for Change. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 173–191. Springer Berlin Heidelberg, 2016.

[61] S. Li et al.: Dynamic Trajectory Planning and Tracking for Autonomous Vehicle With Obstacle Avoidance Based on Model Predictive Control. *IEEE Access*, Bd. 7, S. 132 074–132 086, 2019.

[62] M. Likhachev und D. Ferguson: Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles. *The International Journal of Robotics Research*, Bd. 28(8), S. 933–945, 2009.

[63] M. Likhachev, G. Gordon und S. Thrun: ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *Proceedings of (NeurIPS) Neural Information Processing Systems*, S. 767–774. 2003.

[64] P. Lin: Why Ethics Matters for Autonomous Cars. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 69–85. Springer Berlin Heidelberg, 2016.

[65] A. D. Luca, G. Oriolo und C. Samson: Feedback Control of a Nonholonomic Car-like Robot. In *Robot Motion Planning and Control*, Kap. 4. Springer, 1998.

[66] C. Meerpohl, M. Rick und C. Büskens: Free-space Polygon Creation based on Occupancy Grid Maps for Trajectory Optimization Methods. In *10th IFAC Symposium on Intelligent Autonomous Vehicles*, Bd. 52, S. 368–374. Elsevier BV, 2019.

[67] Introducing DRIVE PILOT: An Automated Driving System for the Highway. Techn. Ber., Mercedes-Benz Research and Development North America, 2019.

[68] T. Mercy, R. V. Parys und G. Pipeleers: Spline-Based Motion Planning for Autonomous Guided Vehicles in a Dynamic Environment. *IEEE Transactions on Control Systems Technology*, Bd. 26(6), S. 2182–2189, 2018.

[69] J. Nilsson et al.: Lane Change Maneuvers for Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, Bd. 18(5), S. 1087–1096, 2017.

[70] M. W. Otte: A Survey of Machine Learning Approaches to Robotic Path-Planning, 2009.

[71] T. Ottmann und P. Widmayer: *Algorithmen und Datenstrukturen*. Springer Vieweg, 6. Aufl., 2017.

[72] B. Paden et al.: A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles*, Bd. 1(1), S. 33–55, 2016.

[73] M. Pavone: Autonomous Mobility-on-Demand Systems for Future Urban Mobility. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 387–402. Springer Berlin Heidelberg, 2016.

[74] S. Pettie: Towards a Final Analysis of Pairing Heaps. In *46th Annual IEEE Symposium on Foundations of Computer Science*, S. 174–183. 2005.

[75] P. Polack et al.: The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *IEEE Intelligent Vehicles Symposium (IV)*, S. 812–818. 2017.

[76] L. Priese und K. Erk: *Theoretische Informatik: Eine umfassende Einführung*. Springer Vieweg, 4. Aufl., 2018.

[77] A. H. Qureshi et al.: Motion Planning Networks. In *2019 International Conference on Robotics and Automation (ICRA)*, S. 2118–2124. 2019.

[78] F. W. Rauskolb et al.: Caroline: An Autonomously Driving Vehicle for Urban Environments. In M. Buehler, K. Iagnemma und S. Singh, Hrsg., *The DARPA Urban Challenge*, S. 441–508. Springer, Berlin, Heidelberg, 2009.

[79] J. A. Reeds und L. A. Shepp: Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, Bd. 145(2), S. 367–393, 1990.

[80] P. Resende und F. Nashashibi: Real-time dynamic trajectory planning for highly automated driving in highways. In *13th International IEEE Conference on Intelligent Transportation Systems*, S. 653–658. 2010.

[81] M. Rick et al.: Autonomous Driving Based on Nonlinear Model Predictive Control and Multi-Sensor Fusion. In *10th IFAC Symposium on Intelligent Autonomous Vehicles*, Bd. 52, S. 182–187. Elsevier BV, 2019.

[82] P. Riekert und T. E. Schunck: Zur Fahrmechanik des gummibereiften Kraftfahzeugs. *Ingenieur Archiv*, Bd. 11, S. 210–224, 1940.

[83] R. Ritschel, N. Schwarz und R. Voßwinkel: An Advanced Cruise Control Like MPC Approach for Cooperative Vehicle Movement Planning. In *Proceedings of the 6th International Symposium on Future Active Safety Technology toward Zero Accidents*. 2021.

[84] R. Ritschel et al.: Nonlinear Model Predictive Path-Following Control for Highly Automated Driving. *Proceedings of the 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV*, Bd. 52(8), S. 350–355, 2019.

[85] Die häufigsten Unfallursachen. Bundesministerium für Verkehr und digitale Infrastruktur, 2020. URL https://www.runtervomgas.de/unfallursachen/artikel/die-haeufigsten-unfallursachen.html. Abgerufen am 11.02.2021.

[86] S. J. Russell und P. Norvig: *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1. Aufl., 1995.

[87] A. Serban, E. Poll und J. Visser: A Standard Driven Software Architecture for Fully Autonomous Vehicles. *Journal of Automotive Software Engineering*, Bd. 1(1), S. 20–33, 2020.

[88] Z. Shiller: *Off-Line and On-Line Trajectory Planning*, Bd. 29, S. 29–62. Springer, 2015.

[89] S. Singh: Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey. Report DOT HS 812 50, National Highway Traffic Safety Administration, Washington, DC, 2018.

[90] L. Sommer et al.: AO-Car: transfer of space technology to autonomous driving with the use of WORHP. In *Proceedings of the 7th International Conference on Astrodynamics Tools and Techniques*. 2018.

[91] W. Song, G. Xiong und H. Chen: Intention-Aware Autonomous Driving Decision-Making in an Uncontrolled Intersection. *Mathematical Problems in Engineering*, 2016.

[92] P. Souères und J.-D. Boissonnat: *Robot Motion Planning and Control*, Kap. Optimal trajectories for nonholonomic mobile robots, S. 93–169. Springer-Verlag, Berlin, 1998.

[93] Containers library, 2021. URL https://en.cppreference.com/w/cpp/container. Abgerufen am 15.07.2021.

[94] J. A. Sussmann und G. Tang: Shortest paths for the reeds-shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. Techn. Ber., 1991.

[95] R. S. Sutton und A. G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, 2010.

[96] A. Sydorchuk: The Boost.Polygon Voronoi Library, 2012. URL https://www.boost.org/doc/libs/1_75_0/libs/polygon/doc/voronoi_main.htm. Abgerufen am 11.10.2021.

[97] Tesla: Autopilot, 2021. URL https://www.tesla.com/de_DE/autopilot. Abgerufen am 18.02.2021.

[98] S. Ulbrich et al.: Towards a Functional System Architecture for Automated Vehicles. Techn. Ber., 2017.

[99] H. Unbehauen: *Regelungstechnik I: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme.* 15. Vieweg+Teubner Verlag Wiesbaden, 2008.

[100] C. Urmson et al.: Autonomous Driving in Urban Environments: Boss and the Urban Challenge. In M. Buehler, K. Iagnemma und S. Singh, Hrsg., *The DARPA Urban Challenge*, Bd. 56. Springer, 2009.

[101] B. R. Vatti: A Generic Solution to Polygon Clipping. *Communications of the ACM*, Bd. 35(7), S. 56–63, 1992.

[102] J. Vuillemin: A data structure for manipulating priority queues. *Communications of the ACM*, Bd. 21(4), S. 309–315, 1978.

[103] W. Wachenfeld et al.: Use Cases for Autonomous Driving. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 9–37. Springer Berlin Heidelberg, 2016.

[104] Y. Wang et al.: Dynamic Trajectory Planning of Autonomous Lane Change at Medium and Low Speeds Based on Elastic Soft Constraint of the Safety Domain. *Automotive Innovation*, Bd. 3(1), S. 73–87, 2020.

[105] J. R. Ward et al.: Extending Time to Collision for probabilistic reasoning in general traffic scenarios. *Transportation Research Part C: Emerging Technologies*, Bd. 51, S. 66–82, 2015.

[106] Waymo: We're building the World's Most Experienced Driver, 2021. URL https://waymo.com/. Abgerufen am 18.02.2021.

[107] 2020 Autonomous Vehicle Technology Report. Wevolver, 2020.

[108] T. Winkle: Safety Benefits of Automated Vehicles: Extended Findings from Accident Research for Development, Validation and Testing. In M. Maurer et al., Hrsg., *Autonomous Driving: Technical, Legal and Social Aspects*, S. 335–364. Springer Berlin Heidelberg, 2016.

[109] M. T. Wolf und J. W. Burdick: Artificial potential functions for highway driving with collision avoidance. In *2008 IEEE International Conference on Robotics and Automation*, S. 3731–3736. 2008.

[110] J. Ziegler et al.: Making Bertha Drive – An Autonomous Journey on a Historic Route. *IEEE Intelligent Transportation Systems Magazine*, Bd. 6(2), S. 8–20, 2014.