Universität Rostock Traditio et Innovatio

Fakultät für Informatik und Elektrotechnik
Institut für Automatisierungstechnik

Beitrag zur Bewegungsplanung autonomer Wasserfahrzeuge

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Robert Damerius, geb. am 10.08.1991 in Bergen auf Rügen
aus Rostock

Rostock, 21. Januar 2025

https://doi.org/10.18453/rosdok_id00005155

eingereicht: 21.01.2025
verteidigt: 28.05.2025
Gutachter: Prof. Dr.-Ing. Torsten Jeinsch
Universität Rostock
Prof. Dr.-Ing. habil. Andreas Rauh
Carl von Ossietzky Universität Oldenburg

Kurzfassung

In dieser Arbeit wird ein neuartiges Verfahren zur Bewegungsplanung traversierfähiger, autonomer Wasserfahrzeuge vorgestellt, wobei das Manövrieren in der Nähe unbewegter Hindernisse im Fokus steht. Die Aufgabe der Bewegungsplanung ist die fortlaufende Berechnung kollisionsfreier Trajektorien von einem initialen Bewegungszustand zu einer vorgegebenen Zielpose. Dabei werden alle unbewegten Hindernisse sowie die geometrische Form des Fahrzeuges mit konvexen Polygonen beschrieben.

Das Verfahren basiert auf dem RRT*-Algorithmus und verfolgt einen sequenziellen Ansatz, bei dem das Planungsproblem in zwei Teilprobleme zerlegt wird. Zuerst wird ein kollisionsfreier Pfad ermittelt, der anhand seiner Länge, dem Abstand zu Hindernissen und der Längsbewegung des Fahrzeuges entlang dieses Pfades optimiert wird. Anschließend wird eine Untermenge des resultierenden Pfades für die nachfolgende Bewegungsplanung verwendet, um eine dynamisch realisierbare und kollisionsfreie Trajektorie zu generieren. Die Berechnung einer Trajektorie beruht auf der numerischen Integration eines geschlossenen Regelkreises, in dem ein nichtlineares Bewegungsmodell des Wasserfahrzeuges verwendet wird. Um Nichtlinearitäten auf Geschwindigkeitsebene zu kompensieren, wird ein Regelsystem auf Basis der exakten Linearisierung entworfen. Damit das Fahrzeug zur Laufzeit auf neu detektierte Hindernisse reagieren kann, wird eine Methode für das wiederholte Neuplanen von Trajektorien vorgestellt, bei der die Rechenzeiten für die Pfad- und Bewegungsplanung berücksichtigt werden. Mithilfe einer Warmstart-Prozedur wird ein zuvor berechneter Pfad für die aktuelle Berechnung genutzt. Die entwickelten Methoden werden anhand von Simulationen quantitativ analysiert und auf einem realen Wasserfahrzeug appliziert und erprobt.

Abstract

This thesis presents a novel method for motion planning of fully-actuated autonomous surface vehicles, focusing on maneuvering in the vicinity of non-moving obstacles. The task of motion planning is to continuously calculate collision-free trajectories from an initial motion state to a given goal pose. All non-moving obstacles and the geometric shape of the vehicle are described by convex polygons.

The method is based on the RRT* algorithm and follows a sequential approach in which the planning problem is decomposed into two sub-problems. First, a collision-free path is determined, which is optimized based on its length, the distance to obstacles and the longitudinal movement of the vehicle along this path. Afterwards, a subset of the resulting path is used for motion planning in order to generate a dynamically feasible and collision-free trajectory. The calculation of a trajectory is based on the numerical integration of a closed-loop control system in which a non-linear motion model of the surface vehicle is used. In order to compensate for non-linearities at the velocity level, a control system based on feedback linearization is designed. To enable the vehicle to react to newly detected obstacles at runtime, a method for the online replanning of trajectories is presented in which the computing times of the path and motion planning are taken into account. A warm start procedure is presented in order to replan a path on the basis of a previously calculated path. The proposed methods are analyzed using simulations and applied and tested on a real surface vehicle.

Wo ein Weg ist, ist auch ein Wille.

Inhaltsverzeichnis

Akronyme

AABB	Axis-Aligned Bounding Box
ASV	Autonomous Surface Vehicle
BSH	Bundesamt für Seeschifffahrt und Hydrographie
COLREG	Convention on the international regulations for preventing collisions at sea
DoF	Freiheitsgrad (engl. *degrees of freedom*)
DP	Dynamische Positionierung
ECEF	Earth Centered, Earth Fixed
GNC	Guidance, Navigation & Control
GNSS	Globales Navigationssatellitensystem
IENC	Inland Electronic Navigational Chart
INS	Inertiales Navigationssystem
LUT	Lookup Table
MPC	Modellprädiktive Regelung (engl. *model predictive control*)
NED	North, East, Down
OBB	Oriented Bounding Box
OCP	Optimalsteuerungsproblem (engl. *optimal control problem*)
PRM	Probabilistic Roadmap
RK4	Klassisches Runge-Kutta-Verfahren
RRT	Rapidly-Exploring Random Tree
RTK	Echtzeitkinematik (engl. *real-time kinematic*)
SD100	Szenarien-Datensatz bestehend aus 100 Szenarien
VWFS	Vermessungs-, Wracksuch- und Forschungsschiff
WGS84	World Geodetic System (1984)
WSV	Wasserstraßen- und Schifffahrtsverwaltung des Bundes

Symbolverzeichnis

$\boldsymbol{A}_{box}$	Transformationsmatrix für das Sampling einer Box um ein Pfadsegment während der Bewegungsplanung
$\boldsymbol{A}_c$	Systemmatrix des Eingangsfilters
$\boldsymbol{A}_\eta$	Systemmatrix für das linearisierte System der Posenregelung
$\boldsymbol{A}_\tau$	Systemmatrix des Kraftmodells
$\boldsymbol{B}$	Eingangsmatrix des Geschwindigkeitsmodells
$\boldsymbol{b}_{box}$	Translation für das Sampling einer Box um ein Pfadsegment während der Bewegungsplanung
$\boldsymbol{B}_c$	Eingangsmatrix des Eingangsfilters
$\boldsymbol{B}_\eta$	Eingangsmatrix für das linearisierte System der Posenregelung
$\boldsymbol{B}_\tau$	Eingangsmatrix des Kraftmodells
b_x, b_y	Tuningparameter für das Festlegen der Größe des Suchgebietes für die Pfadplanung
$\boldsymbol{C}(\boldsymbol{\nu})$	Coriolismatrix, enthält zusätzliche Massen
c_μ	Kostenterm für den Abstand zu Hindernissen
$c(\boldsymbol{q}_0, \boldsymbol{q}_1)$	Kostenwert für den Übergang von Konfiguration $\boldsymbol{q}_0$ zu Konfiguration $\boldsymbol{q}_1$
c_ρ	Kostenterm für die Pfadlänge
c_v	Kostenterm für die Querbewegung zur Pfadrichtung
$\boldsymbol{D}(\boldsymbol{\nu})$	Dämpfungsmatrix
D_ψ	Maximaler Winkelabstand von Samples zum gegebenen Teilpfad während der Bewegungsplanung
D_{xy}	Maximaler Positionsabstand von Samples zum gegebenen Teilpfad während der Bewegungsplanung
$\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$	Einheitsvektoren in x-, y- bzw. z-Richtung
$\boldsymbol{F}$	Koeffizientenmatrix für nichtlineare Terme des Geschwindigkeitsmodells
$f_\varepsilon(x, y)$	Skalarfeld zur Beschreibung von Kostenwerten für den Abstand zu Hindernissen
$\boldsymbol{f}(\boldsymbol{\nu})$	Vektorfunktion für das nichtlineare Geschwindigkeitsmodell
g	Länge der Seitenkante einer Gitterzelle für die LUT des Skalarfeldes f_ε
$g(\Delta\psi)$	Gewichtungsfunktion der Winkeldifferenz für den Kostenterm c_v
g_{max}	Periode für das *goal sampling* während der Pfad- bzw. Bewegungsplanung
$H(k, p)$	Halton-Sequenz für den Iterationsschritt k und Primzahl p
h	Höhe über dem Referenzellipsoid nach WGS84
$\boldsymbol{I}_{n\times n}$	Einheitsmatrix der Dimension $n \times n$
$\boldsymbol{0}_{m\times n}$	Nullmatrix der Dimension $m \times n$
$\boldsymbol{J}$	Jacobi-Matrix
$\boldsymbol{K}_\eta$	Verstärkungsmatrix für die Posenregelung
k_{max}	Maximale Anzahl der Pseudozufallszahlen, die im Vorfeld mithilfe der Halton-Sequenz generiert werden
L	Weglänge einer zweidimensionalen Geraden
L_{max}	Maximale Weglänge des Teilpfades $\boldsymbol{\Pi}_t$ für die Bewegungsplanung
m_g	Modulofaktor, der angibt, für welche Gitterzellen der LUT das Skalarfeldes f_ε berechnet wird

$\boldsymbol{M}$	Trägheitsmatrix, enthält zusätzliche Massen
N	Drehmoment um die z^b-Achse
n	Kardinalität des Baums $\mathcal{T}$
N_c	Kommandiertes Drehmoment um die z^b-Achse
n^m_{max}	Maximale Anzahl an Knoten innerhalb des Baums $\mathcal{T}$ während der Bewegungsplanung
N_{max}	Maximales Drehmoment aufgrund der Stellgrößenbeschränkung
N_{min}	Minimales Drehmoment aufgrund der Stellgrößenbeschränkung
N_{th}	Obere Grenze für den Betrag des Drehmoments für das Erreichen einer Zielpose
$\boldsymbol{n}(\boldsymbol{\nu})$	Vektor aus nichtlinearen Geschwindigkeitstermen
n^p_{max}	Maximale Anzahl an Knoten innerhalb des Baums $\mathcal{T}$ während der Pfadplanung
$\Delta\boldsymbol{p}$	Zweidimensionale Positionsdifferenz
Δp_{max}	Maximaler Positionsfehler für die Approximation einer Kurve durch einen Polygonzug während des Kollisionstests
$\boldsymbol{q}$	Konfiguration
$\boldsymbol{q}_G$	Zielkonfiguration
$\boldsymbol{q}_I$	Startkonfiguration
r	Drehrate um die Hochachse des körperfesten Koordinatensystems
$\boldsymbol{R}^n_b(\psi)$	Rotationsmatrix für $\mathbb{R}^2$
$r_{p,min}$	Minimaler körperfester Positionsradius für das *line-of-sight*-basierte *Guidance*-Gesetz
r_{th}	Obere Grenze für den Betrag der Drehrate für das Erreichen einer Zielpose
$r_{x,max}, r_{y,max}$	Maximale körperfeste Radien für das *line-of-sight*-basierte *Guidance*-Gesetz
r_x, r_y, r_ψ	Körperfeste Radien für das *line-of-sight*-basierte *Guidance*-Gesetz
$\boldsymbol{S}$	Steuerbarkeitsmatrix
t	Zeit
$\boldsymbol{T}^n_b(\psi)$	Transformationsmatrix für $SE(2)$
t_c	Rechenzeit
t_{conv}	Konvergenzzeit
$\bar{t}_{conv}$	Mittlere Konvergenzzeit
t_ϵ	Zeit zur Berücksichtigung der tatsächlichen Rechenzeit der sequenziellen Bewegungsplanung
T_{f1}, T_{f2}, T_{f3}	Zeitkonstanten für das Eingangsfilter
t^m_{max}	Maximale Rechenzeit für die Bewegungsplanung
t^p_{max}	Maximale Rechenzeit für die Pfadplanung
t_{new}	Startzeitpunkt einer neu geplanten Trajektorie $\boldsymbol{\xi}_{new}$
t_{now}	Aktueller Zeitpunkt
T_s	Abtastzeit
t_ξ	Startzeitpunkt der Trajektorie $\boldsymbol{\xi}$
T_X, T_Y, T_N	Zeitkonstanten für das lineare Kraftmodell erster Ordnung
u	Längsgeschwindigkeit in körperfesten Koordinaten
$\boldsymbol{u}_\tau$	Eingangsvektor des nichtlinearen Zustandsraummodells für die Posenregelung
u_{th}	Obere Grenze für den Betrag der Längsgeschwindigkeit für das Erreichen einer Zielpose
$\boldsymbol{u}$	Eingangsvektor
v	Quergeschwindigkeit in körperfesten Koordinaten
V_{box}	Volumen einer Box um ein Pfadsegment während des Samplings innerhalb der Bewegungsplanung

V	Menge aller Knoten aus einer Nachbarschaft
v_{th}	Obere Grenze für den Betrag der Quergeschwindigkeit für das Erreichen einer Zielpose
w_α	Wichtungsfaktor für Quer- und Rückwärtsbewegungen im Kostenterm c_v
w_β	Stauchungsfaktor der Gewichtungsfunktion $g(\Delta\psi)$ für den Kostenterm c_v
w_ψ	Wichtungsfaktor für die Winkelkomponente der Distanzmetrik
w_v	Wichtungsfaktor für den Kostenterm, der die Querbewegung entlang eines Pfades bestraft
X	Längskraft
X_c	Kommandierte Längskraft
x	x-Position in Nordrichtung
$\boldsymbol{x}$	Zustandsvektor
Δx_{th}	Obere Grenze für den Betrag der x-Positionsdifferenz für das Erreichen einer Zielpose
$\boldsymbol{x}_\nu$	Zustandsvektor des nichtlinearen Zustandsraummodells auf Geschwindigkeitsebene
X_{max}	Maximale Längskraft aufgrund der Stellgrößenbeschränkung
X_{min}	Minimale Längskraft aufgrund der Stellgrößenbeschränkung
X_{th}	Obere Grenze für den Betrag der Längskraft für das Erreichen einer Zielpose
Y	Querkraft
Y_c	Kommandierte Querkraft
y	y-Position in Ostrichtung
Δy_{th}	Obere Grenze für den Betrag der y-Positionsdifferenz für das Erreichen einer Zielpose
$\boldsymbol{y}_\nu$	Ausgangsvektor des nichtlinearen Zustandsraummodells auf Geschwindigkeitsebene
Y_{max}	Maximale Querkraft aufgrund der Stellgrößenbeschränkung
Y_{min}	Minimale Querkraft aufgrund der Stellgrößenbeschränkung
Y_{th}	Obere Grenze für den Betrag der Querkraft für das Erreichen einer Zielpose
$\boldsymbol{z}$	Transformierter Zustandsvektor
α	Tuningparameter für das Wichten des Skalarfeldes f_ε gegenüber anderen Termen der Kostenfunktion
β	Tuningparameter für den exponentiellen Abfall des Skalarfeldes f_ε
$\delta(x,y)$	Kleinster Abstand des Punktes $(x,y)^{\mathrm{T}}$ zu allen Polygonen
$\boldsymbol{\varepsilon}$	Strecke zwischen zwei Konfigurationen
$\boldsymbol{\eta}$	Pose des Fahrzeuges, gegeben im erdfesten Koordinatensystem
$\Delta\boldsymbol{\eta}^b$	Posendifferenz in körperfesten Koordinaten
$\Delta\boldsymbol{\eta}^n$	Posendifferenz in Navigationskoordinaten
$\Gamma : \mathcal{C} \mapsto \mathbb{R}^3$	Abbildung des Konfigurationsraums auf einen äquivalenten euklidischen Raum
γ	Konstante zur Beschränkung der Schrittweite im RRT*-Algorithmus
$\boldsymbol{\kappa}_i^b$	i-ter Punkt in körperfesten Koordinaten, an dem c_μ ausgewertet wird
λ	Geografischer Längengrad nach WGS84
λ_{max}	Obere Beschränkung der Schrittweite λ_s
λ_s	Schrittweite, Länge der Strecke zwischen zwei Konfigurationen
$\mu(\mathcal{C}_{free})$	Volumen des freien Konfigurationsraums
$\boldsymbol{\nu}$	Geschwindigkeitsvektor des Fahrzeuges bezüglich des erdfesten Koordinatensystems, gegeben im körperfesten Koordinatensystem
$\boldsymbol{\Omega}$	Geografischer Ursprung für das NED-Koordinatensystem

φ	Geografischer Breitengrad nach WGS84
$\mathbf{\Pi}$	Pfad
$\mathbf{\Pi}_r$	Referenzpfad
$\mathbf{\Pi}_t$	Teilpfad mit reduzierter Länge
ψ	Heading-Winkel
$\Delta\psi_{max}$	Maximale Winkeländerung für die Approximation der konvexen Hülle während des Kollisionstests
ψ_ε	Winkel einer zweidimensionalen Gerade
$\Delta\psi_{th}$	Obere Grenze für den Betrag der Heading-Winkeldifferenz für das Erreichen einer Zielpose
ρ	Distanzmetrik für $SE(2)$
$\sigma(\psi)$	Funktion zur Abbildung des Argumentes auf den Winkelbereich $[-\pi, +\pi)$
$\boldsymbol{\sigma}(\boldsymbol{q})$	Funktion zur Abbildung der Winkelkomponenten des Argumentes auf den Winkelbereich $[-\pi, +\pi)$
$\boldsymbol{\tau}$	Kräfte-Momenten-Vektor, gegeben im körperfesten Koordinatensystem
$\boldsymbol{\tau}_c$	Stellgröße, kommandierter Kräfte-Momenten-Vektor
$\boldsymbol{\xi}$	Trajektorie
$\boldsymbol{\xi}_{new}$	Neu geplante Trajektorie
$\boldsymbol{\xi}_{old}$	Zuvor geplante Trajektorie
ζ_d	Volumen der Einheitskugel in $\mathbb{R}^d$
$\mathcal{A}(\boldsymbol{x}_\nu)$	Vektorfunktion für die Kompensation nichtlinearer Effekte des Geschwindigkeitsmodells in der exakten Linearisierung
$\mathcal{B}$	Eingangsmatrix für die Kompensation nichtlinearer Effekte des Geschwindigkeitsmodells in der exakten Linearisierung
$\mathcal{C}$	Konfigurationsraum
$\mathcal{C}_{free}$	Teilmenge des Konfigurationsraums, der nicht zu Kollisionen führt
$\mathcal{C}_{obs}$	Teilmenge des Konfigurationsraums, der zu Kollisionen führt
$\mathcal{H}$	Halbraum
$\mathcal{O}$	Geometrische Form von Hindernissen, bestehend aus konvexen Polygonen
$\mathcal{P}$	Konvexes Polygon
$\mathcal{S}$	Sample-Set bestehend aus einer endlichen Menge von Konfigurationen
$\mathcal{T}$	Baum
$\mathcal{V}$	Geometrische Form des Fahrzeuges, bestehend aus konvexen Polygonen
$\mathcal{V}(\boldsymbol{q})$	Mit $\boldsymbol{q}$ transformierte geometrische Form des Fahrzeuges
$\mathcal{V}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$	Exakte geometrische Fahrzeugform entlang einer Kante $\boldsymbol{\varepsilon}$ von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$
$\mathcal{V}^{\diamondsuit}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$	Approximation von $\mathcal{V}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch eine endliche Anzahl konvexer Polygone
$\mathcal{V}_{\boldsymbol{\kappa}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$	Exakte geometrische Fahrzeugform entlang einer Kurve $\boldsymbol{\kappa}$ von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$
$\mathcal{V}^{\diamondsuit}_{\boldsymbol{\kappa}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$	Approximation von $\mathcal{V}_{\boldsymbol{\kappa}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch eine endliche Anzahl konvexer Polygone
$\mathcal{W}$	Welt
$\mathcal{X}$	Phasenraum
$\mathcal{X}_{free}$	Teilmenge des Phasenraums, der nicht zu Kollisionen führt
$\mathcal{X}_{obs}$	Teilmenge des Phasenraums, der zu Kollisionen führt
$\mathbb{N}_0$	Menge natürlicher Zahlen inklusive der 0
$\mathbb{R}^n$	n-dimensionaler euklidischer Raum
$SE(2)$	Spezielle euklidische Gruppe für eine zwei-dimensionale Welt; Menge aller Translationen aus $\mathbb{R}^2$ und Rotationen aus $SO(2)$
$SO(2)$	Drehgruppe; Menge orthogonalen Matrizen aus $\mathbb{R}^{2\times2}$ mit Determinante $+1$

1. Einleitung

Die Entwicklung autonomer Wasserfahrzeuge (engl. *Autonomous Surface Vehicle* (ASV)) ist ein zentrales Forschungsfeld, das sowohl für die Wissenschaft als auch für die Industrie von wachsender Bedeutung ist. Mit der zunehmenden Automatisierung im maritimen Sektor eröffnen sich neue Möglichkeiten, um die Effizienz, Sicherheit und Nachhaltigkeit von Anwendungen in der See- und Binnenschifffahrt zu verbessern. Das steigende internationale Verkehrsaufkommen in Verbindung mit stetig größer werdenden Schiffen stellt die Sicherheit in Häfen und engen Wasserstraßen vor hohe Herausforderungen [1]. Gleichzeitig erfordert der zunehmende Fachkräftemangel in der maritimen Branche [2] neue Lösungen, die mit weniger Personal künftigen Anforderungen gerecht werden. Hochautomatisierte oder in ferner Zukunft auch autonome Fahrzeuge, die eigenständig komplizierte Aufgaben sicher und zuverlässig bewältigen, können einen Beitrag leisten. Die Voraussetzung dafür ist, dass ein Fahrzeugsystem selbst Entscheidungen ohne das aktive Eingreifen des Menschen treffen kann. Eine Schlüsselkomponente ist die Bewegungsplanung, mit der ein ASV zukünftige Bewegungsaktionen berechnet, um eine festgelegte Zielpose zu erreichen. Für ein automatisiertes Manövrieren in der Nähe von Hindernissen wie beispielsweise bei An- und Ablegevorgängen ist das Planen kollisionsfreier Trajektorien essenziell.

Das Problem der Bewegungsplanung für ASVs lässt sich folgendermaßen beschreiben: Ausgehend vom aktuellen Bewegungszustand des Wasserfahrzeuges ist eine Trajektorie gesucht, die diesen Anfangszustand mit einer gegebenen Zielpose verbindet. Entlang der resultierenden Trajektorie müssen Kollisionen des Wasserfahrzeuges mit Hindernissen ausgeschlossen sein, wobei die geometrische Form des Fahrzeuges zu berücksichtigen ist. Gleichzeitig muss die Trajektorie dem dynamischen Bewegungsverhalten des Fahrzeuges entsprechen und die Stellgrößen müssen innerhalb gegebener Beschränkungen liegen. Durch das Minimieren einer Kostenfunktion ist zudem eine bestmögliche Trajektorie zu berechnen.

Die Problemstellung der Bewegungsplanung autonomer Fahrzeuge findet sich auch in anderen Branchen. Im Automotive-Bereich ist seit 2016 ein massives Forschungsinteresse an selbstfahrenden Autos zu erkennen [3]. Zu den Forschungsthemen gehört beispielsweise das Planen kollisionsfreier Pfade im urbanen Straßenverkehr [4, 5, 6]. In der Raumfahrt müssen Rover auf anderen Planeten oder Monden in unbekanntem Terrain selbstständig navigieren und den Weg zu einem Zielpunkt finden [7, 8, 9]. Die aktuell existierenden Verfahren aus diesen Branchen lassen sich jedoch nicht problemlos auf maritime Systeme übertragen. Ein Grund ist, dass sich das Bewegungsverhalten eines Wasserfahrzeuges wesentlich von dem eines Landfahrzeuges unterscheidet. Zudem haben externe Störungen wie Wind und Strömung einen signifikanten Einfluss auf die Bewegung von Wasserfahrzeugen [10]. Daher ist eine Modifikation existierender Verfahren notwendig.

Das Ziel der Arbeit ist die Entwicklung eines Verfahrens zur Bewegungsplanung traversierfähiger ASVs. Der Anwendungsbereich beschränkt sich auf Manövriervorgänge in sicherheitskritischen Bereichen wie beispielsweise beim An- und Ablegen. Das Planungsproblem wird in zwei Teilprobleme zerlegt, die sequenziell gelöst werden. Mit einem sampling-basierten Planungsalgorithmus wird zuerst ein optimaler kollisionsfreier Pfad berechnet, der eine gegebene Startpose mit einer festgelegten Zielpose verbindet. Dieser Pfad wird in der nachfolgenden Bewegungsplanung genutzt, um eine dynamisch realisierbare und kollisionsfreie Trajektorie in dessen Umgebung zu generieren. Für die Trajektorienberechnung wird ein geschlossener Regelkreis numerisch integriert, wobei ein nichtlineares Bewegungsmodell als Prozessmodell verwendet wird. Durch den Einsatz der exakten Linearisierung [11, 12] werden Nichtlineari-

täten der Regelstrecke auf Geschwindigkeitsebene kompensiert. Für den praktischen Einsatz der Bewegungsplanung wird eine Methode vorgestellt, bei der unter Berücksichtigung der Rechenzeit der Pfad- und Bewegungsplanung fortlaufend neue Trajektorien erzeugt werden. Diese Vorgehensweise ist notwendig, damit das Fahrzeugsystem zur Laufzeit die aktuell detektierten Hindernisse verarbeiten kann. Ein zuvor berechneter Pfad wird mithilfe einer Warmstart-Prozedur für das aktuelle Pfadplanungsproblem genutzt.

Die vorliegende Arbeit ist in acht Kapitel gegliedert. Eine grafische Übersicht der nachfolgenden Kapitel ist in Abbildung 1.1 dargestellt. In Kapitel 2 wird auf fundamentale Begriffe und mathematisch relevante Grundlagen eingegangen. Anschließend wird in Kapitel 3 der aktuelle Stand der Technik beleuchtet. Es werden verschiedene Verfahren für das Lösen von Planungsproblemen vorgestellt, unter anderem der RRT*-Algorithmus, welcher in dieser Arbeit von zentraler Bedeutung ist. In Kapitel 4 wird die Problemstellung der Bewegungsplanung für Wasserfahrzeuge herausgearbeitet. Dabei werden die existierenden Verfahren aus Kapitel 3 bezüglich gestellter Anforderungen analysiert und offene Fragestellungen aufgezeigt, aus denen ein Lösungsansatz abgeleitet wird. Das entwickelte Verfahren zur sequenziellen Bewegungsplanung wird in Kapitel 5 erläutert. Es setzt sich aus einer Pfadplanung mit einer nachgelagerten Bewegungsplanung zusammen, die in einem Online-Algorithmus kombiniert werden. Kapitel 6 untersucht das Verfahren in der Simulation, um den Einfluss verschiedener Tuningparameter auf die resultierende Lösung zu analysieren. Im Anschluss wird in Kapitel 7 die Applikation des Verfahrens für ein reales Wasserfahrzeug beschrieben. Das Kapitel 8 fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Entwicklungen autonomer Wasserfahrzeuge.

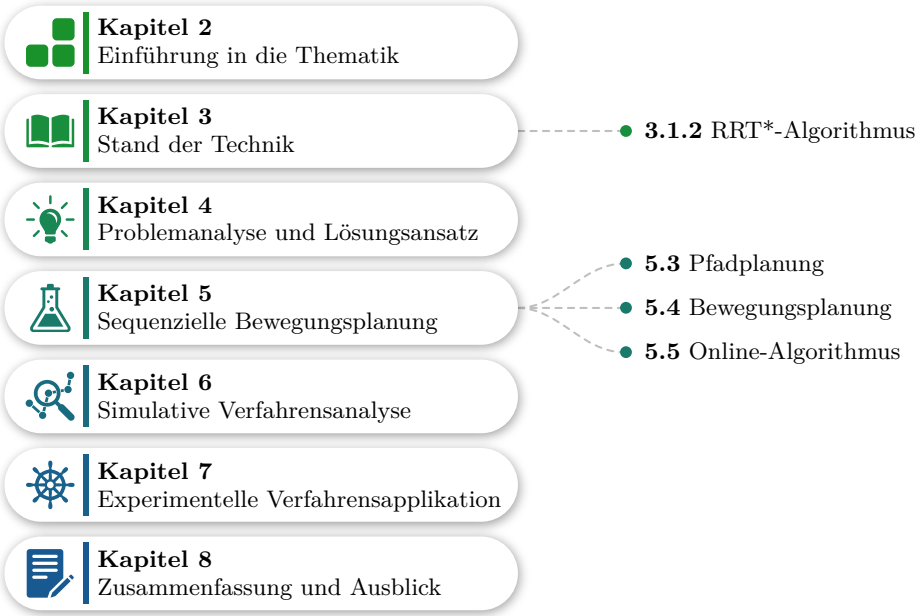


Abbildung 1.1.: Grafische Übersicht der nachfolgenden Kapitel.

2. Einführung in die Thematik

In diesem Kapitel werden fundamentale Begriffe erklärt und mathematisch relevante Grundlagen zusammengefasst. Neben der Definition für Pfad und Trajektorie, die unterschiedliche Bedeutungen im Sinne der Bewegungsplanung haben, wird der Konfigurationsraum und die geometrische Modellierung von Hindernissen vorgestellt. Zudem wird auf notwendige Begriffe der Graphentheorie eingegangen.

2.1. Koordinatensysteme

Für die Positions- und Lagebestimmung eines ASVs werden unterschiedliche Koordinatensysteme verwendet. Das erdfeste Koordinatensystem (***e-frame***) entspricht einem kartesischen Koordinatensystem, welches seinen Ursprung im Erdmittelpunkt hat. In Abbildung 2.1 ist das *e-frame* auf der linken Seite dargestellt. Die z^e-Achse entspricht der Erdrotationsachse und die x^e-Achse verläuft durch den Schnittpunkt der Äquatorebene mit dem Nullmeridian. Dieses Koordinatensystem wird auch mit *Earth Centered, Earth Fixed* (ECEF) bezeichnet [13]. Die Position auf der Erdoberfläche wird in Kugelkoordinaten angegeben. Dazu wird die Erdfigur durch ein Referenzellipsoid angenähert. Das gängigste geodätische Referenzsystem ist das *World Geodetic System* von 1984 (WGS84). Die geografische Breite φ gibt die nördliche oder südliche Entfernung eines Punktes vom Äquator in Form eines Winkelmaßes an. Die geografische Länge λ entspricht dem Azimuthwinkel und gibt die östliche oder westliche Entfernung eines Punktes vom Nullmeridian an. Die Höhe über dem Referenzellipsoid wird mit h bezeichnet.

Das Navigationskoordinatensystem (***n-frame***) ist ein kartesisches Koordinatensystem, welches seinen Ursprung an einem beliebigen Punkt auf der Erdkugel, mit Ausnahme der Pole, hat. Die x^n- und y^n-Achse zeigen nach Norden bzw. Osten. Beide Achsen bilden eine Tangentialebene an einem festgelegten geografischen Ursprung auf dem Ellipsoid. Die z^n-Achse zeigt in Richtung des Erdmittelpunktes. Daher wird dieses Koordinatensystem auch mit *North, East, Down* (NED) bezeichnet. Auf der linken Seite der Abbildung 2.1 ist die Tangentialebene, deren Ursprung sich bei $\mathbf{\Omega} = (\varphi, \lambda, h)^T$ befindet, dargestellt. Das zugehörige kartesische Navigationskoordinatensystem ist auf der rechten Seite abgebildet.

Viele Größen, die Einfluss auf das Bewegungsverhalten eines ASVs haben, lassen sich gut in in körperfesten Koordinaten beschreiben. Dazu gehören beispielsweise die durch Antriebe erzeugten Kräfte und Momente. Aus diesem Grund wird ein körperfestes Koordinatensystem (***b-frame***) definiert, welches gegenüber dem *n-frame* um den Heading-Winkel ψ des Fahrzeuges verdreht ist. Die x^b-Achse zeigt in Vorausrichtung und die y^b-Achse zeigt nach Steuerbord. Wie beim NED-System zeigt die z^b-Achse ebenfalls nach unten. Entlang dieser Achsen werden die translatorischen Geschwindigkeiten u, v sowie die rotatorische Geschwindigkeit r definiert. Gleiches gilt für die Kräfte X und Y sowie für das Moment N. Die translatorischen Größen lassen sich mithilfe der Rotationsmatrix

$$\boldsymbol{R}_b^n(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \tag{2.1}$$

vom *b-frame* in das *n-frame* transformieren [13].

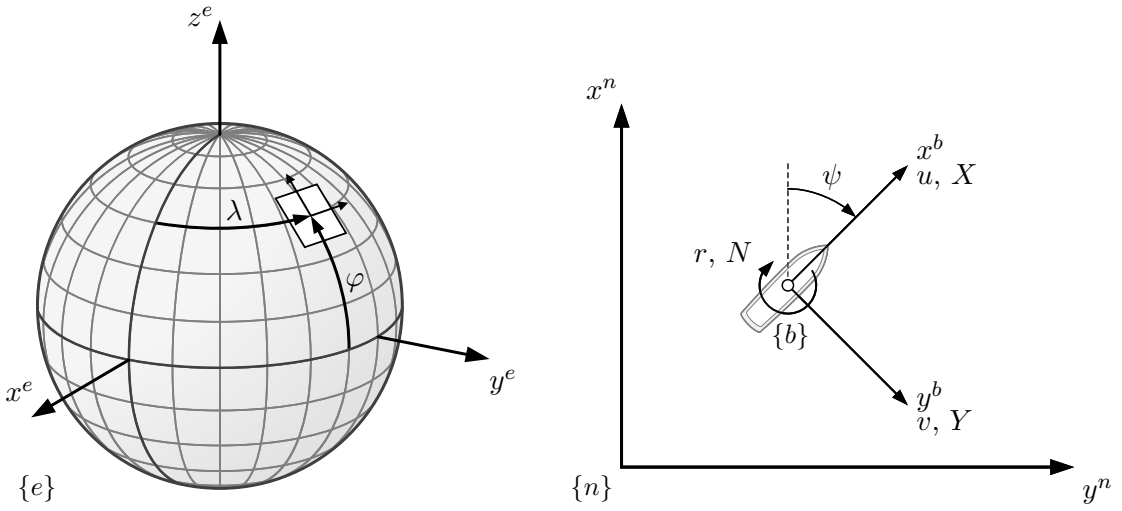


Abbildung 2.1.: Schematische Darstellung des Navigationskoordinatensystems als Tangentialebene auf dem Referenzellipsoid sowie des körperfesten Koordinatensystems für ein Wasserfahrzeug mit drei Freiheitsgraden.

2.2. Grundlegende Begriffe

Das Bewegungsverhalten eines ASVs lässt sich mit einer gewöhnlichen nichtlinearen Differenzialgleichung der Form

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.2}$$

modellieren, wobei $\boldsymbol{x}(t) = (\boldsymbol{\eta}^{\mathrm{T}}(t), \boldsymbol{\nu}^{\mathrm{T}}(t))^{\mathrm{T}}$ der Zustandsvektor und $\boldsymbol{u}(t)$ der Eingangsvektor zum Zeitpunkt t ist. Der Zustandsvektor enthält die Position und die Lage des Fahrzeuges $\boldsymbol{\eta}(t)$ sowie dessen Geschwindigkeiten $\boldsymbol{\nu}(t)$. Die Kombination von Position und Lage wird auch als Pose bezeichnet. Im Eingangsvektor sind die Stellgrößen des Fahrzeuges enthalten. Die Anzahl der verwendeten Freiheitsgrade und somit die Dimensionen von $\boldsymbol{\eta}$, $\boldsymbol{\nu}$ und $\boldsymbol{u}$ hängen von der Anwendung ab und spielen für eine allgemeine Darstellung zunächst keine Rolle. Eine Aneinanderreihung mehrerer Posen ergibt den Pfad

$$\boldsymbol{\Pi} = \{\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \cdots, \boldsymbol{\eta}_N\} \tag{2.3}$$

ohne zeitlichen Bezug. Das bedeutet, dass die einzelnen Posen $\boldsymbol{\eta}_i$ keinem Zeitpunkt zugeordnet werden können. Ein Pfad entspricht somit einer geordneten Liste von N Posen oder Wegpunkten. Für die geometrische Darstellung eines Pfades können die einzelnen Posen in der entsprechenden Reihenfolge durch gerade Linien miteinander verbunden werden. Im Gegensatz zum Pfad bezeichnet eine Trajektorie eine zeitabhängige Folge von Zustands- und zugehörigen Eingangsgrößen. Werden die Zustands- und Eingangsgrößen nur zu zeitdiskreten Punkten betrachtet, ist eine Trajektorie mit

$$\boldsymbol{\xi} = \{(\boldsymbol{x}_1, \boldsymbol{u}_1), (\boldsymbol{x}_2, \boldsymbol{u}_2), \cdots, (\boldsymbol{x}_N, \boldsymbol{u}_N)\} \tag{2.4}$$

definiert, wobei $\boldsymbol{x}_k = \boldsymbol{x}(t = kT_s)$ und $\boldsymbol{u}_k = \boldsymbol{u}(t = kT_s)$ für $k = 1, \ldots, N$ gilt. Hierbei ist $T_s > 0$ die Abtastzeit. In dieser Definition ist der Anfangszustand $\boldsymbol{x}_0$ und der Anfangseingangsvektor $\boldsymbol{u}_0$ zum Zeitpunkt $t = 0$ nicht Teil der Trajektorie. Diese Art der Definition ist hilfreich, um mehrere Teiltrajektorien zu einer vollständigen Trajektorie zusammenfassen zu können.

Für die Planung von Trajektorien gibt es zwei Begriffe, die oft synonym verwendet werden: Trajektorienplanung (engl. *trajectory planning* oder auch *trajectory generation*) und Bewegungsplanung (engl. *motion planning*). In der Literatur findet sich der Begriff der Trajektorienplanung jedoch häufiger bei Anwendungen, in denen Hindernisse keine Rolle spielen [14]. Derartige Verfahren zielen oftmals nur auf die Erzeugung einer dynamisch realisierbaren Trajektorie

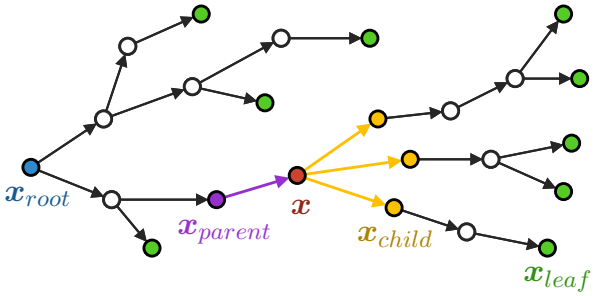


Abbildung 2.2.: Beispiel eines gewurzelten Baums.

ab, also einer Trajektorie, für welche die differenzielle Beschränkung der Form (2.2) gilt. Die Trajektorienplanung kann in diesem Fall als Teilmenge der Bewegungsplanung gesehen werden. Bei der Bewegungsplanung ist das Berücksichtigen von Hindernissen von zentraler Bedeutung. Aus diesem Grund wird in der vorliegenden Arbeit der Begriff der Bewegungsplanung favorisiert.

Die Bewegungsplanung hat die Aufgabe, einen bestmöglichen Plan zu finden. Im Allgemeinen ist ein Plan eine Abfolge von Aktionen, wobei eine Aktion den Übergang von einem Zustand in einen anderen Zustand beschreibt. Im Kontext von Planungsalgorithmen umfasst der Begriff Zustand die Gesamtheit aller Informationen zur vollständigen Beschreibung der momentanen Eigenschaften des Systems [15]. Im Falle der Bewegungsplanung eines Fahrzeuges ist dieser Zustand gleichbedeutend mit dem Bewegungszustand des Fahrzeuges. Ausgehend von einem Zustand werden durch unterschiedliche Aktionen verschiedenen benachbarte Zustände erreicht. Es entsteht ein Netzwerk von Zuständen, die über Aktionen miteinander verbunden sind. Dieses Netzwerk lässt sich als Graph beschreiben. Ein Graph besteht aus Knoten und Kanten, welche die Knoten verbinden. Ein Knoten entspricht folglich einem Zustand und eine Kante einer Aktion für den Übergang zwischen zwei Zuständen. Für diese Arbeit sind Bäume als spezielle Art von Graphen relevant. Ein Baum ist ein zusammenhängender, kreisfreier Graph. Das bedeutet, dass alle Knoten indirekt miteinander verbunden sind, ohne dass sich Schleifen oder Zyklen bilden [16]. Im Folgenden wird von gewurzelten Bäumen ausgegangen, die einen Wurzelknoten $\boldsymbol{x}_{root}$ besitzen, von dem aus alle weiteren Knoten erreicht werden können. In Abbildung 2.2 ist ein Beispiel für einen gewurzelten Baum dargestellt. Aus Sicht des roten Knotens $\boldsymbol{x}$ werden den benachbarten Knoten bestimmte Namen zugeordnet. Ein vorangegangener Knoten wird als Elternknoten $\boldsymbol{x}_{parent}$ bezeichnet. Bis auf den Wurzelknoten haben alle Knoten genau einen Elternknoten. Ein nachfolgender Knoten ist hingegen ein Kindknoten $\boldsymbol{x}_{child}$, wobei jeder Knoten beliebig viele Kindknoten haben kann. Knoten, die keine Kindknoten haben, werden als Blätter des Baums definiert und sind in der Abbildung 2.2 als grüne Blattknoten $\boldsymbol{x}_{leaf}$ dargestellt. Die Anzahl aller Knoten innerhalb eines Graphen wird als Mächtigkeit oder auch Kardinalität bezeichnet.

2.3. Konfigurationsraum und geometrische Modellierung von Hindernissen

In diesem Abschnitt wird der Konfigurationsraum sowie eine Möglichkeit zur geometrischen Modellierung von Hindernissen auf Basis konvexer Polygone beschrieben. Für weiterführende Informationen zu dieser Thematik wird auf die Literatur [15] verwiesen.

Gegeben sei ein Fahrzeug, welches sich in der Welt $\mathcal{W}$ bewegt. Für den Anwendungsfall autonomer Oberflächenfahrzeuge gilt $\mathcal{W} = \mathbb{R}^2$, da sich das Fahrzeug nur in einer zweidimensionalen Ebene bewegt. In dieser Welt kann das Fahrzeug beliebige Konfigurationen

$$\boldsymbol{q} = (x,\ y,\ \psi)^{\mathrm{T}}$$

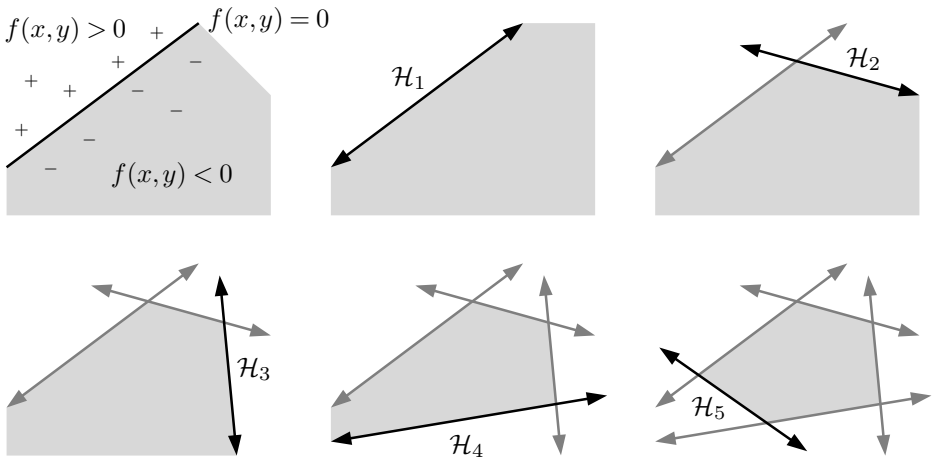


Abbildung 2.3.: Beschreibung konvexer Polygone durch Schnittmengen von Halbräumen.

annehmen, wobei $(x,y)^{\mathrm{T}} \in \mathbb{R}^2$ die Position in einem zweidimensionalen Raum und ψ den Heading-Winkel des Fahrzeuges beschreibt. Alle möglichen Ausrichtungen des Fahrzeuges mit ψ werden in der speziellen orthogonalen Gruppe $SO(2)$ zusammengefasst. Die Kombination aller möglichen Translationen aus $\mathbb{R}^2$ und Rotationen aus $SO(2)$ ergeben die spezielle euklidische Gruppe $SE(2) = \mathbb{R}^2 \times SO(2)$. Der Konfigurationsraum entspricht der Menge aller Konfigurationen und ist demnach mit

$$\mathcal{C} = SE(2)$$

gegeben. Somit kann der Konfigurationsraum als eine Menge aller Posen verstanden werden. Einige dieser Posen führen zur Kollision des Fahrzeuges mit Hindernissen.

Für die mathematische Repräsentation von Hindernissen werden Polygone verwendet. Polygone sind zweidimensionale Polytope, die mathematisch sehr gut handhabbar sind. Die Kante eines Polygons wird mit der impliziten Geradengleichung

$$f(x,y) = ax + by + c = 0 \tag{2.5}$$

beschrieben, die einen Raum in zwei Hälften teilt. Alle Punkte auf der einen Seite der Geraden ergeben einen negativen Funktionswert und alle Punkte auf der anderen Seite der Geraden ergeben einen positiven Funktionswert. Bilden a und b einen Normaleneinheitsvektor, dann entspricht $|f(x,y)|$ dem minimalen Abstand des Punktes $(x,y)^{\mathrm{T}}$ von der Geraden. Mithilfe der impliziten Geradengleichung wird der Halbraum

$$\mathcal{H}_i = \{(x,y) \in \mathcal{W} \mid f_i(x,y) \leq 0\}$$

definiert. Die Schnittmenge von m Halbräumen resultiert in einem m-seitigen konvexen Polygon

$$\mathcal{P} = \mathcal{H}_1 \cap \mathcal{H}_2 \cap \cdots \cap \mathcal{H}_m \ ,$$

wie in Abbildung 2.3 dargestellt ist. Die Eckpunkte eines Polygons werden auch Vertices genannt. Ein konvexes Polygon ist dadurch charakterisiert, dass alle Punkte auf einer Linie zwischen zwei beliebigen Punkten innerhalb des Polygons ebenfalls im Polygon liegen. Demnach gilt für ein konvexes Polygon

$$\begin{gathered} \boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathcal{P} \\ \lambda \boldsymbol{p}_1 + (1-\lambda)\boldsymbol{p}_2 \in \mathcal{P} \ \forall \ \lambda \in [0,1] \ . \end{gathered}$$

Durch die Vereinigung mehrerer konvexer Polygone lassen sich nicht-konvexe Polygone

$$\mathcal{O} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \cdots \cup \mathcal{P}_n$$

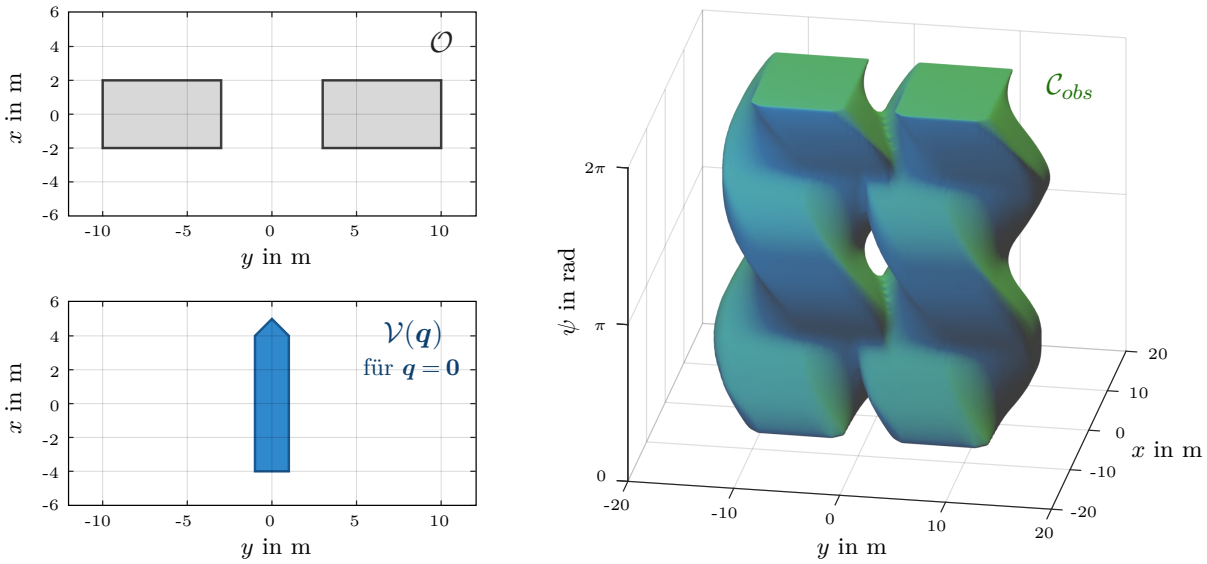


Abbildung 2.4.: Grafische Darstellung der Teilmenge $\mathcal{C}_{obs}$ (rechts) an einem Beispiel mit zwei rechteckigen Hindernissen $\mathcal{O}$ (links oben) und einem fünf-eckigen Fahrzeugpolygon $\mathcal{V}(\boldsymbol{q})$ (links unten).

und somit beliebige Hindernisstrukturen abbilden. Auf die gleiche Weise wird die geometrische Form des Fahrzeuges $\mathcal{V}$ durch ein oder mehrere konvexe Polygone beschreiben. Werden alle Vertices des Fahrzeugpolygons mit einer beliebigen Konfiguration $\boldsymbol{q} \in \mathcal{C}$ transformiert, so ergibt sich $\mathcal{V}(\boldsymbol{q})$. Alle Konfigurationen, die zu einer Kollision führen, werden in der Teilmenge

$$\mathcal{C}_{obs} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{V}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}$$

zusammengefasst. In Abbildung 2.4 ist $\mathcal{C}_{obs}$ an einem Beispiel grafisch illustriert. Auf der linken Seite sind sowohl die Hindernisse $\mathcal{O}$ als auch das Fahrzeugpolygon $\mathcal{V}(\boldsymbol{q})$ für $\boldsymbol{q} = \mathbf{0}$ abgebildet. Auf der rechten Seite ist die Menge aller Konfigurationen als dreidimensionaler Körper gezeigt, die zu einer Überschneidung von $\mathcal{V}(\boldsymbol{q})$ mit $\mathcal{O}$ führen. Eine Methode zur Überprüfung der Überschneidung zweier konvexer Polygone ist in [15] beschrieben. Ein Pfadplanungsalgorithmus muss einen Lösungspfad hervorbringen, welcher zwei Konfigurationen miteinander verbindet, wobei dieser Lösungspfad vollständig im freien Konfigurationsraum $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ liegen muss. Anhand der graphischen Darstellung aus Abbildung 2.4 ist zu erkennen, dass das Fahrzeug das Hindernis zwischen den zwei Rechtecken für Heading-Winkel nahe 0 bzw. 2π und π durchfahren kann. Je nach Form von Fahrzeug und Hindernissen können beliebig komplizierte Strukturen für $\mathcal{C}_{obs}$ entstehen.

Der Konfigurationsraum kann erweitert werden, indem nicht nur die Pose des Fahrzeuges, sondern auch die Geschwindigkeiten berücksichtigt werden. In diesem Fall wird der Konfigurationsraum $\mathcal{C}$ zu dem Phasenraum $\mathcal{X}$ erweitert, der die Menge aller möglichen Zustände enthält. Die Schwierigkeit liegt dann in der Beschreibung von $\mathcal{X}_{obs}$, also der Menge an Zuständen, die zur Kollision führen. In jedem Fall gilt $\mathcal{C}_{obs} \subseteq \mathcal{X}_{obs}$. Für ein Schiff, welches sich beispielsweise in einem Zustand $\boldsymbol{x}$ befindet, in dem es auf ein Hindernis zufährt und nicht mehr ausweichen kann, gilt zwar $\boldsymbol{q} \in \mathcal{C}_{free}$ jedoch nicht $\boldsymbol{x} \in \mathcal{X}_{free}$.

3. Stand der Technik

Dieses Kapitel beleuchtet den aktuellen Stand der Forschung und Technik. Es werden die gängigsten Methoden zur Pfad- und Bewegungsplanung vorgestellt. Abschließend werden zahlreiche Entwicklungen aus den Bereichen der Luft- und Raumfahrt, des Automobilsektors und der maritimen Branche mit dem Schwerpunkt der Pfad- und Bewegungsplanung aufgezeigt.

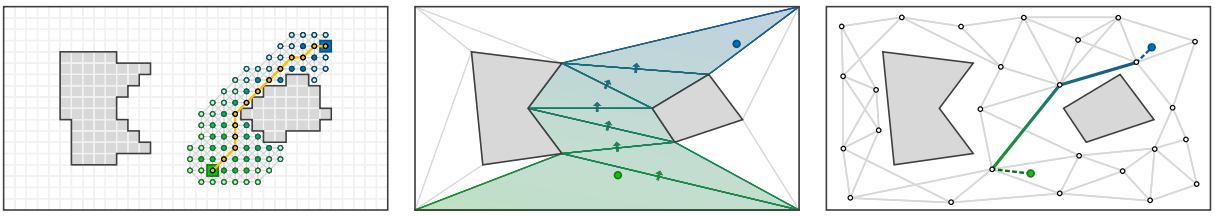
3.1. Methoden zur Pfad- und Bewegungsplanung

Das Problem der Pfad- und Bewegungsplanung ist PSPACE-schwer [17, 18]. Das bedeutet, dass es unwahrscheinlich oder sogar unmöglich ist, dass derartige Planungsprobleme jemals durch einen effizienten Algorithmus in polynomieller Zeit gelöst werden können - also in einer Zeit, die mit der Problemgröße nicht stärker wächst als eine Polynomfunktion. Es existieren Ansätze zum effizienten Lösen bestimmter Planungsprobleme, die jedoch aufgrund ihrer hohen Rechenzeit praktisch nicht einsetzbar sind [15]. Daher wurden zahlreiche Methoden entwickelt, die das Problem für spezielle Problemklassen so gut wie möglich in begrenzter Rechenzeit lösen können, indem sie gewisse Kompromisse bezüglich der Lösung eingehen. Die Methoden lassen sich nach unterschiedlichen Gesichtspunkten klassifizieren. Im Folgenden werden die bekanntesten Methoden vorgestellt.

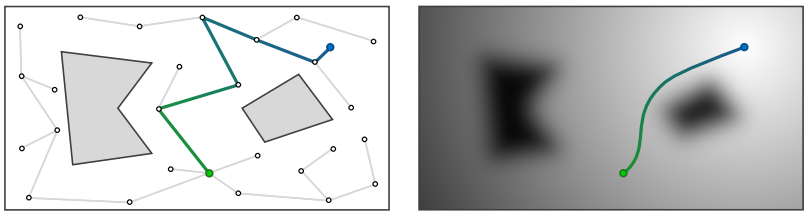
Gitter-basierte Methoden zerlegen den Konfigurationsraum in ein Gitternetz aus typischerweise gleichgroßen Zellen. Durch eine bestimmte Aktion gelangt das Fahrzeug von einer Zelle in die benachbarte Zelle. Sowohl der Start- als auch der Zielpunkt sind bestimmten Gitterzellen zugeordnet. Durch Hindernisse sind einige Gitterzellen belegt und können nicht erreicht werden. Die Verbindungen zwischen freien Gitterzellen bilden einen Graphen. Mithilfe von Suchalgorithmen aus der Graphentheorie wird ein optimaler Pfad vom Start- zum Zielpunkt gefunden. Die bekanntesten Algorithmen sind A* [19] und D* [20]. Abbildung 3.1a zeigt ein Beispiel. Ein Nachteil gitter-basierter Methoden ist, dass die Gitterauflösung einen entscheidenden Einfluss auf die Performance hat. Zudem können hoch-dimensionale Probleme aufgrund der systematischen Suche nicht praktikabel gelöst werden [14].

Geometrische Methoden verzichten auf ein Gitter. Lässt sich der Konfigurationsraum analytisch, beispielsweise mit Polygonen, beschreiben, so kann ein geometrisches Modell für $\mathcal{C}_{free}$ erstellt werden. Der freie Raum wird dann in Zellen zerlegt. Ähnlich wie bei gitter-basierten Methoden entsteht aus den benachbarten Zellen ein Graph, der wiederum systematisch durchsucht wird. Die Zerlegung von $\mathcal{C}_{free}$ in Zellen gibt der Methode ihren Namen: *cell decomposition*. Beispiele finden sich in [21], [22] und [23]. In Abbildung 3.1b ist die Methode schematisch dargestellt. Für einen Punkt in einem zweidimensionalen Raum mit polygonalen Hindernissen ist das problemlos möglich. Wird der Punkt jedoch durch ein drehbares Polygon ersetzt, findet sich keine analytische Beschreibung mehr [15]. Voronoi-Diagramme [24] und *visibility graphs* [25] sind weitere geometrische Methoden, die hier nicht näher erläutert werden.

Sampling-basierte Methoden bilden eine weitere wichtige Klasse von Methoden zum Lösen von Planungsproblemen. Ein bedeutender Vorteil ist dabei, dass keine explizite Modellierung des Konfigurationsraums notwendig ist. Stattdessen wird der Konfigurationsraum durch eine große Anzahl zufälliger Samples erkundet. Dabei genügt eine *black-box*-Funktion,



(a) A* (gitter-basiert) (b) cell decomposition (c) probabilistic roadmap



(d) rapidly-exploring random tree (e) Potenzialfeld

Abbildung 3.1.: Schematische Darstellung unterschiedlicher Algorithmen zur Pfadplanung zwischen Startpunkt (grün) und Zielpunkt (blau).

die bestimmt, ob eine gesampelte Konfiguration gültig, also beispielsweise kollisionsfrei, ist oder nicht. Im Gegensatz zu den oben genannten Methoden, die auf einer systematischen Suche basieren, sind sampling-basierte Methoden nicht vollständig. Das bedeutet, sie können keine Aussage darüber treffen, ob eine Lösung existiert oder nicht. Aus diesem Grund gibt es den schwächeren Begriff der probabilistischen Vollständigkeit [15]. Existiert eine Lösung und geht die Anzahl der Samples gegen unendlich, so geht die Wahrscheinlichkeit dafür, dass die Lösung gefunden wird, gegen 1. Zu den bekanntesten Methoden gehören *probabilistic roadmap* (PRM) [26] und *rapidly-exploring random tree* (RRT) [27]. Bei PRM wird in einer Vorberechnung zunächst der freie Konfigurationsraum durch eine endliche Anzahl an Samples abgetastet, die schließlich in einem oder mehreren Graphen miteinander verbunden werden. Wird nun ein Paar aus Start- und Zielpunkt vorgegeben, kann unter Anwendung des zuvor berechneten Graphen schnell eine Lösung gefunden werden, indem beide Punkte dem Netzwerk hinzugefügt werden und eine Pfadsuche innerhalb des Graphen durchgeführt wird, wie in Abbildung 3.1c gezeigt. Das ist besonders dann vorteilhaft, wenn viele Kombinationen aus Start- und Zielpunkt untersucht werden müssen und sich der Konfigurationsraum nicht verändert [28]. Im Gegensatz zu PRM baut RRT einen Baum ausgehend vom Startpunkt auf, indem zufällige Konfigurationen gesampelt und dem bestehenden Baum hinzugefügt werden. Ein Beispiel ist in Abbildung 3.1d dargestellt. RRT hat sich aufgrund seiner Flexibilität und einfachen Umsetzung in der Praxis bewährt und gilt als *state-of-the-art* für das Lösen von Planungsproblemen [29, 30]. Die Methode wird in Abschnitt 3.1.1 detailliert vorgestellt.

Die **Potenzialfeld-Methode** bietet eine weitere Möglichkeit zur Bewegungsplanung [31, 32]. Die grundsätzliche Idee ist dabei, dass der Zielpunkt durch eine Senke in einem Potenzialfeld dargestellt wird. Hindernisse werden durch zusätzliche hohe Potenziale abgebildet. Für jeden Punkt im Raum lässt sich der Gradient bestimmen. Das Minimum wird erreicht, indem fortlaufend Schritte in Richtung des stärksten Gefälles ausgeführt werden. In Abbildung 3.1e ist dazu ein Beispiel gezeigt. Einer der entscheidenden Nachteile dieser Methode ist, dass ein gefundenes Minimum möglicherweise nicht dem Ziel entspricht. Der Lösungspfad endet unter Umständen in einem lokalen Minimum [33].

Das Planungsproblem lässt sich auch als **Optimierungsproblem** [34] formulieren. Sei $\boldsymbol{x}_0$ der Anfangszustand zum Zeitpunkt t_0 und $\boldsymbol{x}_f$ der Endzustand zum Zeitpunkt t_f, dann

ist die Stellgröße $\boldsymbol{u}(t)$ gesucht, welche eine gegebene Kostenfunktion J unter einer Reihe von Nebenbedingungen minimiert.

$$\begin{aligned} \min_{\boldsymbol{u}(t),t_f} \quad & J(t_f, \boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \text{u.d.N.} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{max} \\ & \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_f \\ & \boldsymbol{h}(\boldsymbol{x}(t)) \leq \boldsymbol{0} \end{aligned} \tag{3.1}$$

Das dynamische Bewegungsverhalten wird durch die differenzielle Beschränkung $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$ berücksichtigt. Die Nebenbedingung $\boldsymbol{u}_{min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{max}$ stellt sicher, dass die Eingangsgrößen in einem erlaubten Bereich liegen. Gleichzeitig sind Anfangs- und Endzustand durch die zwei Nebenbedingungen $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ und $\boldsymbol{x}(t_f) = \boldsymbol{x}_f$ gegeben. Hindernisse müssen durch eine nichtlineare Nebenbedingung $\boldsymbol{h}(\boldsymbol{x}(t)) \leq \boldsymbol{0}$ beschrieben werden. Es entsteht ein nichtlineares Optimierungsproblem, welches mit entsprechenden Solvern gelöst werden muss. Dabei gibt es keine Garantie dafür, dass eine Lösung gefunden wird und diese das globale Minimum darstellt [35]. Zudem hängt der Erfolg von guten Startwerten ab [36].

Für das Lösen des Optimalsteuerungsproblems (engl. *optimal control problem* (OCP)) gibt es viele mögliche Methoden, die ihrerseits nach unterschiedlichen Gesichtspunkten klassifiziert werden können. Eine Übersicht zu verschiedenen Methoden findet sich in [37] und [38]. Für ausführliche Informationen zum Lösen von OCPs sei auf [34] und [39] verwiesen. In [40] werden die gängigsten Methoden zum numerischen Lösen von Optimierungsproblemen beschrieben.

3.1.1. Rapidly-Exploring Random Tree

Der RRT-Algorithmus gehört zur Klasse der sampling-basierten Verfahren. Einer der Vorteile des Algorithmus ist, dass dieser nicht für ein spezielles Problem ausgelegt ist, sondern vielmehr als ein Framework gesehen werden kann, welches erst durch die Implementierung bestimmter *black-box*-Funktionen die spezifische Funktionalität zum Lösen eines konkreten Problems erhält. Aus diesem Grund gibt es zahlreiche Erweiterungen dieser Methode, wobei eine Erweiterung in den meisten Fällen nur ausgewählte Funktionalitäten in den *black-box*-Funktionen realisiert [41]. Das Anwendungsgebiet des RRT liegt aufgrund seiner Generalisierbarkeit nicht nur in der Pfad- und Bewegungsplanung von Fahrzeugen und Roboterarmen, sondern kommt beispielsweise auch bei der Analyse von Proteinfaltungen zum Einsatz [42].

Im Nachfolgenden wird die grundsätzliche Funktionsweise des Algorithmus erläutert. Gegeben ist eine Startkonfiguration $\boldsymbol{q}_I$ sowie eine Zielkonfiguration $\boldsymbol{q}_G$. Die genaue Struktur von $\mathcal{C}_{obs}$ ist nicht bekannt, jedoch existiert eine boolesche Funktion $\Phi : \mathcal{C} \mapsto \{\textsc{True}, \textsc{False}\}$, mit deren Hilfe ermittelt werden kann, ob eine Konfiguration in $\mathcal{C}_{obs}$ liegt. Der Algorithmus verwaltet eine Baumstruktur aus Konfigurationen. Durch mehrfaches Iterieren wächst der Baum um zufällig generierte Konfigurationen. Die Methode ist in Algorithmus 1 beschrieben. Darüber hinaus ist die Funktionsweise einer Iteration in der Abbildung 3.2 illustriert. In den Teilabbildungen wird aus Gründen der Anschaulichkeit von einem Baum ausgegangen, der bereits durch mehrere vorangegangene Iterationen gewachsen ist. Außerdem ist $\boldsymbol{q}_G$ nicht abgebildet, da dieser Knoten für das Hinzufügen neuer Knoten zunächst nicht von Interesse ist.

Zu Beginn wird der Algorithmus initialisiert. Dabei wird $\boldsymbol{q}_I$ als Wurzelknoten des Baums $\mathcal{T}$ gesetzt (Zeile 1). Der Lösungsknoten $\boldsymbol{q}_{solution}$ wird ebenfalls mit $\boldsymbol{q}_I$ initialisiert (Zeile 2). Anschließend erfolgt eine definierte Anzahl an Iterationen (Zeile 3). In Abbildung 3.2 ist der aktuelle Baum zu Beginn einer Iteration als Schritt (0) dargestellt. In Zeile 4 wird eine zufällige Konfiguration $\boldsymbol{q}_{rand}$ erzeugt. Aus dem Baum wird der Knoten $\boldsymbol{q}_{near}$ ausgewählt, der $\boldsymbol{q}_{rand}$ am

Algorithmus 1: Rapidly-Exploring Random Tree (RRT)

```
1:  T.Init(q_I);
2:  q_solution ← q_I;
3:  for k = 1 to K do
4:      q_rand ← Sample();
5:      q_near ← NearestNeighbor(T, q_rand);
6:      (q_new, ε) ← Steer(q_near, q_rand);
7:      if IsFeasible(ε) then
8:          T.Insert(q_near, q_new);
9:          if GoalReached(q_new, q_G) then
10:             q_solution ← q_new;
11:         end
12:     end
13: end
14: Π ← T.GetBranch(q_solution);
```

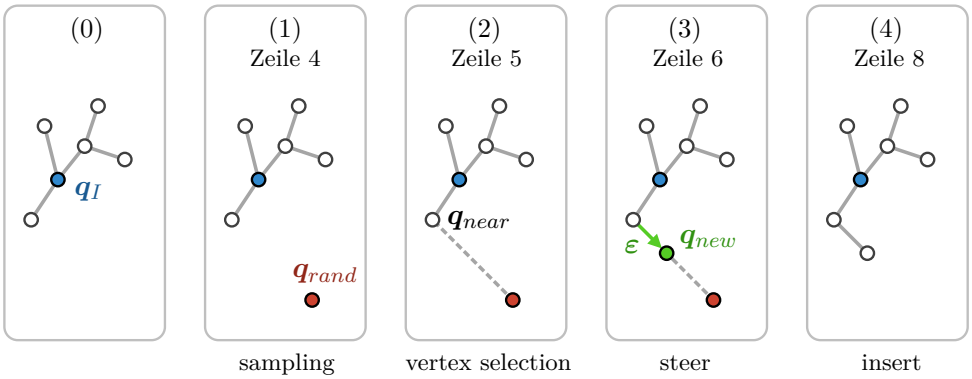


Abbildung 3.2.: Grafische Darstellung einer Iteration des RRT-Algorithmus.

dichtesten ist (Zeile 5). Anschließend wird der Baum von $\boldsymbol{q}_{near}$ aus in Richtung des zufälligen Samples $\boldsymbol{q}_{rand}$ mithilfe einer *Steer*-Funktion erweitert (Zeile 6). Nach [18] muss $\boldsymbol{q}_{rand}$ dabei nicht exakt erreicht werden. Der Baum wird lediglich in Richtung $\boldsymbol{q}_{rand}$ erweitert. Das Ergebnis der *Steer*-Funktion ist eine neue Konfiguration $\boldsymbol{q}_{new}$, die in Zeile 8 zu $\mathcal{T}$ hinzugefügt wird, falls die Teilstrecke $\boldsymbol{\varepsilon}$ von $\boldsymbol{q}_{near}$ zu $\boldsymbol{q}_{new}$ gültig ist (Zeile 7). Dabei ist $\boldsymbol{q}_{near}$ der Elternknoten von $\boldsymbol{q}_{new}$. In Zeile 9 wird untersucht, ob $\boldsymbol{q}_{new}$ in der Nähe von $\boldsymbol{q}_G$ liegt. Die Wahl eines Zielgebietes anstelle einer exakten Konfiguration ist hierbei sinnvoll, da es unwahrscheinlich ist, eine gegebene Zielkonfiguration exakt durch zufälliges Sampling zu erreichen. Ist ein Lösungsknoten $\boldsymbol{q}_{solution}$ aus der Umgebung von $\boldsymbol{q}_G$ gefunden, dann wird der Lösungspfad $\boldsymbol{\Pi}$ des Planungsproblems bestimmt, indem von $\boldsymbol{q}_{solution}$ rekursiv über alle Elternknoten bis zum Wurzelknoten gegangen wird (Zeile 14).

3.1.2. Optimal Rapidly-Exploring Random Tree

Wie bereits erwähnt, gibt es zahlreiche Erweiterungen des RRT-Algorithmus. Die bedeutendste Erweiterung ist RRT*. Es kann gezeigt werden, dass Algorithmus 1 keine optimale Lösung hervorbringen kann [43]. Ist eine Lösung gefunden, wie in Abbildung 3.1d gezeigt, so wird diese durch weiteres Iterieren nicht verbessert. Aus diesem Grund wurde eine Variante entwickelt, bei welcher der Baum durch die Umstrukturierung von Kanten optimiert wird. Jedem Knoten wird

Algorithmus 2: Optimal Rapidly-Exploring Random Tree (RRT*)

1: $\mathcal{T}$.Init($\boldsymbol{q}_I$);
2: $\boldsymbol{q}_{solution} \leftarrow \boldsymbol{q}_I$;
3: **for** $k = 1$ **to** K **do**
4: $\quad \boldsymbol{q}_{rand} \leftarrow$ Sample();
5: $\quad \boldsymbol{q}_{near} \leftarrow$ NearestNeighbor($\mathcal{T}, \boldsymbol{q}_{rand}$);
6: $\quad (\boldsymbol{q}_{new}, \boldsymbol{\varepsilon}) \leftarrow$ Steer($\boldsymbol{q}_{near}, \boldsymbol{q}_{rand}$);
7: $\quad$ **if** IsFeasible($\boldsymbol{\varepsilon}$) **then**
8: $\quad\quad V \leftarrow$ Near($\mathcal{T}, \boldsymbol{q}_{new}$);
9: $\quad\quad \boldsymbol{q}_{parent} \leftarrow$ FindBestParent($V, \boldsymbol{q}_{near}, \boldsymbol{q}_{new}$); `// see algorithm 3`
10: $\quad\quad \mathcal{T}$.Insert($\boldsymbol{q}_{parent}, \boldsymbol{q}_{new}$);
11: $\quad\quad$ Rewire($\mathcal{T}, V \setminus \boldsymbol{q}_{parent}, \boldsymbol{q}_{new}$); `// see algorithm 4`
12: $\quad\quad$ **if** GoalReached($\boldsymbol{q}_{new}, \boldsymbol{q}_G$) **then**
13: $\quad\quad\quad \boldsymbol{q}_{solution} \leftarrow \boldsymbol{q}_{new}$;
14: $\quad\quad$ **end**
15: $\quad$ **end**
16: **end**
17: $\boldsymbol{\Pi} \leftarrow \mathcal{T}$.GetBranch($\boldsymbol{q}_{solution}$);

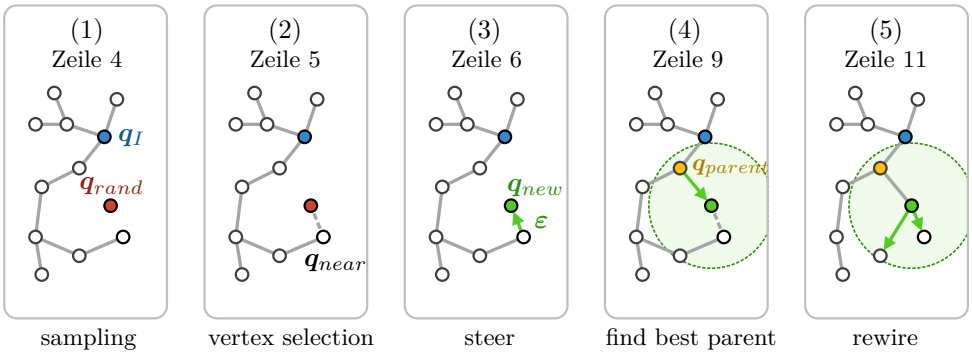


Abbildung 3.3.: Grafische Darstellung einer Iteration des RRT*-Algorithmus.

dabei ein positiver Kostenwert zugeordnet, der die Kosten vom Wurzelknoten $\boldsymbol{q}_I$ zum entsprechenden Knoten entlang des zugehörigen Zweiges des Baums angibt. Das kann beispielsweise die Summe der Kantenlängen sein. Durch die Einführung eines Kostenwertes für jeden Knoten ist es im Vergleich zum RRT-Algorithmus möglich, nach optimalen Verbindungen im Sinne eines minimalen Kostenwertes zu suchen. In Algorithmus 2 ist das Verfahren dargestellt und in Abbildung 3.3 ist das Prinzip einer Iteration des RRT*-Algorithmus grafisch illustriert. Die Zeilen 1-7 sowie die Zeilen 10 und 12-17 entsprechen dem RRT Algorithmus und die Zeilen 8, 9 und 11 stellen die Erweiterung dar. In Zeile 8 werden zunächst alle Knoten aus der Nachbarschaft des neu hinzugefügten Knotens $\boldsymbol{q}_{new}$ ermittelt. Die Menge aller Nachbarknoten wird mit V bezeichnet und ist in der Abbildung 3.3 durch einen gestrichelten Kreis markiert. In Zeile 9 wird der beste Elternknoten $\boldsymbol{q}_{parent}$ für $\boldsymbol{q}_{new}$ aus V ermittelt. Anschließend kann die neue Verbindung von $\boldsymbol{q}_{parent}$ zu $\boldsymbol{q}_{new}$ dem Baum hinzugefügt werden. In Zeile 11 wird untersucht, ob $\boldsymbol{q}_{new}$ selbst ein besserer Elternknoten für andere Knoten aus $V \setminus \boldsymbol{q}_{parent}$ sein kann. Ist dies der Fall, werden die Kanten des Baums umstrukturiert.

Die *FindBestParent*-Prozedur ist in Algorithmus 3 dargestellt. In dieser Prozedur wird untersucht, ob $\boldsymbol{q}_{new}$ über einen anderen Knoten aus V besser erreicht werden kann, als über $\boldsymbol{q}_{near}$. In Zeile 2 wird der Elternknoten mit $\boldsymbol{q}_{near}$ initialisiert. Anschließend werden alle verbleibenden Knoten aus V untersucht (Zeile 3). In Zeile 4 wird der Kostenwert c_i für die

potenzielle Verbindung zwischen $\boldsymbol{q}$ und $\boldsymbol{q}_{new}$ berechnet. Hierbei gibt die *Cost*-Funktion den Gesamtkostenwert eines Knotens an und $c(\boldsymbol{q}_0, \boldsymbol{q}_1)$ kennzeichnet den Kostenwert zwischen zwei Knoten $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$. In Zeile 5 wird die Kante $\boldsymbol{\varepsilon}$ mithilfe der *Steer*-Prozedur errechnet. Stimmt die resultierende Konfiguration $\boldsymbol{q}_s$ mit $\boldsymbol{q}_{new}$ überein und ist die Verbindung beider Konfigurationen gültig, so wird $\boldsymbol{q}$ als bester Elternknoten gewählt, falls die Gesamtkosten zu $\boldsymbol{q}_{new}$ über diesen Elternknoten geringer sind (Zeile 6-7). In diesem Fall wird der Kostenwert für $\boldsymbol{q}_{new}$ aktualisiert (Zeile 8).

Algorithmus 3: *FindBestParent*-Prozedur

```
1:  procedure q_parent ← FindBestParent(V, q_near, q_new)
2:      q_parent ← q_near;
3:      foreach q ∈ V \ q_near do
4:          c_i ← Cost(q) + c(q, q_new);
5:          (q_s, ε) ← Steer(q, q_new);
6:          if q_s = q_new ∧ c_i < Cost(q_new) ∧ IsFeasible(ε) then
7:              q_parent ← q;
8:              Cost(q_new) ← c_i;
9:          end
10:     end
11: end
```

Die *Rewire*-Prozedur ist in Algorithmus 4 dargestellt und untersucht, ob $\boldsymbol{q}_{new}$ selbst ein besserer Elternknoten für andere Knoten aus $V_r = V \setminus \boldsymbol{q}_{parent}$ sein kann. Dabei ist die *Rewire*-Prozedur der *FindBestParent*-Prozedur sehr ähnlich. Während bei der *FindBestParent*-Prozedur Verbindungen von $\boldsymbol{q}$ nach $\boldsymbol{q}_{new}$ untersucht werden, werden bei der *Rewire*-Prozedur Verbindungen von $\boldsymbol{q}_{new}$ nach $\boldsymbol{q}$ betrachtet. Falls $\boldsymbol{q}_{new}$ ein besserer Elternknoten für $\boldsymbol{q}$ ist, wird der Baum $\mathcal{T}$ umstrukturiert, indem der Elternknoten für $\boldsymbol{q}$ aktualisiert wird (Zeile 7). Die Kostendifferenz $\Delta c_i < 0$ in Zeile 6, welche die Kostenverbesserung für $\boldsymbol{q}$ darstellt, wird innerhalb der *Reconnect*-Funktion rekursiv auf den Kostenwert aller Kindknoten von $\boldsymbol{q}$ sowie deren Kindknoten addiert. Somit wird der Kostenwert für den gesamten Teilbaum ausgehend von $\boldsymbol{q}$ aktualisiert.

Algorithmus 4: *Rewire*-Prozedur

```
1:  procedure Rewire(T, V_r, q_new)
2:      foreach q ∈ V_r do
3:          c_i ← Cost(q_new) + c(q_new, q);
4:          (q_s, ε) ← Steer(q_new, q);
5:          if q_s = q ∧ c_i < Cost(q) ∧ IsFeasible(ε) then
6:              Δc_i ← c_i − Cost(q);
7:              T.Reconnect(q, q_new, Δc_i);
8:          end
9:      end
10: end
```

Sowohl für die *FindBestParent*-Prozedur aus Algorithmus 3 als auch für die *Rewire*-Prozedur aus Algorithmus 4 wird eine *Steer*-Funktion benötigt, die zwei Konfigurationen exakt miteinander verbindet. Die exakte Verbindung zweier Konfigurationen wird dann problematisch, wenn diese als Systemzustände dargestellt werden und die Dynamik berücksichtigt werden

muss. Eine exakte Verbindung zwischen zwei Konfigurationen entspräche in diesem Fall der exakten Verbindung zweier Bewegungszustände. Für die Lösung dieses Problems muss unter Umständen ein aufwändiges nichtlineares Optimierungsproblem wie in (3.1) gelöst werden.

Typischerweise wird die Länge der Teilstrecke $\boldsymbol{\varepsilon}$ innerhalb der *Steer*-Funktion beschränkt. Die resultierende Schrittweite $\lambda_s = |\boldsymbol{\varepsilon}|$ wird dann gleichzeitig als Radius für die *Near*-Funktion verwendet. In [43] wird gezeigt, dass λ_s in Abhängigkeit von der Kardinalität des Baums berechnet werden kann. Je mehr Knoten im Graphen enthalten sind, die den freien Konfigurationsraum füllen, desto kleiner kann die Schrittweite für die aktuelle Iteration gewählt werden. Eine entsprechende Verkleinerung der Schrittweite führt dazu, dass die Anzahl der Knoten aus V reduziert wird, was den Rechenaufwand der *FindBestParent*- und *Rewire*-Funktionen reduziert, ohne die Konvergenz zur optimalen Lösung zu gefährden. Für eine gegebene obere Beschränkung der Schrittweite λ_{max} wird in einer Iteration die Schrittweite mit

$$\lambda_s = \min\left\{ \left(\frac{\gamma}{\zeta_d} \frac{\log n}{n} \right)^{\frac{1}{d}},\ \lambda_{max} \right\} \tag{3.2}$$

berechnet. Hierbei bezeichnet d die Dimension des Konfigurationsraums und n die Kardinalität des Baums. ζ_d ist das Volumen der Einheitskugel in $\mathbb{R}^d$ und

$$\gamma > 2^d \left(1 + \frac{1}{d}\right) \mu(\mathcal{C}_{free}) \tag{3.3}$$

ist eine Konstante, wobei $\mu(\mathcal{C}_{free})$ dem Volumen des freien Konfigurationsraums entspricht.

Eine auf RRT* aufbauende Erweiterung ist *Informed-RRT** [44], bei dem das Gebiet, aus dem zufällige Samples gezogen werden, einer Ellipse entspricht, die fortlaufend auf Basis der aktuellen Lösung verkleinert wird. Darüber hinaus gibt es Erweiterungen, die nicht nur einen Baum vom Startpunkt aus wachsen lassen, sondern einen zweiten Baum ausgehend vom Zielpunkt aufbauen [45]. Diese Idee lässt sich auch zu einer Erweiterung verallgemeinern, bei der beliebig viele Bäume parallel wachsen, die anschließend miteinander verbunden werden müssen. Für die Berücksichtigung der Dynamik wurde der *closed-loop rapidly-exploring random tree* (CL-RRT) [5] entwickelt. Hierbei wird innerhalb der *Steer*-Funktion eine zufällige Stellgröße für das Fahrzeug generiert. Der neue Zustand entsteht dann durch die Vorwärtssimulation mit der zufällig gesampelten Stellgröße. Die Erweiterung RRT*FN [46] verwendet eine feste Anzahl von Knoten innerhalb des Baums und beschränkt somit den Speicherbedarf. Die Bezeichnung FN steht hierbei für *fixed nodes*. Ist die maximale Anzahl an Knoten erreicht, werden zufällige Blattknoten des Baums entfernt.

3.2. Überblick zu aktuellen Entwicklungen

Im Raumfahrtsektor werden typischerweise gitter-basierte Methoden zur Pfadplanung von Rovern eingesetzt. In [47] wird der A*-Algorithmus für die Pfadsuche in einem Gitternetz verwendet, wobei die Zellen zusätzliche geländespezifische Kosten wie die Steilheit oder Rauigkeit des Bodens enthalten. Für einige Mars-Rover kommt der D*-Algorithmus zu Einsatz. Die Beschaffenheit des felsigen Terrains wird durch Kameras analysiert und in einem Gitternetzmodell gespeichert. Anschließend wird in diesem Gitternetz nach einem globalen Pfad gesucht, wobei die Struktur des Geländes und der Aufwand an Lenkbewegungen mit in das Gütekriterium einfließen [48, 49]. In [50] werden Bewegungsvorlagen (engl. *motion primitives*) verwendet. Dazu werden offline Trajektorien für zahlreiche Manöver generiert und in einer Datenbank gespeichert. Die Trajektorien basieren auf Polynomen und berücksichtigen das dynamische Bewegungsverhalten des Rovers. Mithilfe des D*-Algorithmus erfolgt online die optimale Aneinanderreihung mehrerer Bewegungsvorlagen.

Auch im Bereich unbemannter Luftfahrzeuge, wie beispielsweise Quadrokopter, finden sich zahlreiche Veröffentlichungen mit dem Schwerpunkt der Pfad- und Bewegungsplanung. In [51] wird die Trajektoriengenerierung auf Basis festgelegter Wegpunkte vorgestellt, wobei die Trajektorie aus mehreren Polynomen besteht. Es werden kinematische Beschränkungen für Geschwindigkeit und Beschleunigung vorgegeben. Durch eine Optimierung der Parameter des Polynoms wird zudem die vierte Ableitung des Polynoms minimiert, welche im algebraischen Zusammenhang mit der Stellgröße des Quadrokopters steht. In [52] wird dieser Ansatz erweitert, indem die Pfade für das Ausweichen von Hindernissen mit dem RRT*-Algorithmus erzeugt werden. Die Verwendung von *motion primitives* für Quadrokopter wird in [53] vorgestellt. Die Aneinanderreihung der generierten Bewegungsvorlagen erfolgt mithilfe des A*-Algorithmus.

Im Automobilbereich werden sampling-basierte Verfahren wie RRT [54] oder RRT* [55] verwendet, um Pfade oder Trajektorien zu generieren. Das automatische Fahren entlang einer Fahrspur sowie das Ausweichen von Hindernissen, die sich auf dieser Fahrspur befinden, wird in [5, 56] vorgestellt. In [6] wird ein A*-basierter Ansatz verfolgt, um kollisionsfreie Pfade zu einer freien Parkfläche auf einem Parkplatz zu finden. Der resultierende Pfad wird anschließend durch ein nichtlineares Optimierungsverfahren geglättet. Die Planung von Manövern wie das Abbiegen an einer Kreuzung wird in [57] thematisiert. Hierbei werden Bézier-Kurven 3. Grades verwendet, deren Parameter durch eine nichtlineare Optimierung bestimmt werden. Anhand einer nachgelagerten linearen modellprädiktiven Regelung wird das Geschwindigkeitsprofil entlang der Bézier-Kurve berechnet. In [58] wird die Trajektorie für einen Spurwechsels generiert, indem das Problem als lineares Optimierungsproblem formuliert und gelöst wird.

Im maritimen Bereich findet sich eine Vielzahl an Veröffentlichungen, welche die Thematik der Kollisionsvermeidung von Schiffen auf hoher See unter Berücksichtigung von internationalen Kollisionsverhütungsregeln (*Convention on the international regulations for preventing collisions at sea* (COLREG)) bearbeiten. Beispiele finden sich in [59, 60, 61, 62, 63, 64, 65]. Klassischerweise wird ein kinematisches Modell oder gar kein Bewegungsmodell verwendet, um eine Ausweichbahn zu berechnen, damit Kollisionen mit anderen Verkehrsteilnehmern vermieden werden. Jedem Fahrzeug wird ein Sicherheitsbereich in Form eines entsprechend großen Kreises zugeordnet. In [66] wird der RRT-Algorithmus verwendet, um auch statische Hindernisse wie Inseln beim Ausweichen einzukalkulieren. Um zusätzlich das Bewegungsverhalten des Fahrzeuges einbeziehen zu können, werden in [67] *motion primitives* verwendet. Für die Pfadplanung von ASVs kommen gitter-basierte Verfahren wie A* [68] oder Dijkstra [69] zum Einsatz. Ebenso werden sampling-basierte Verfahren wie RRT [70] oder RRT* [71] verwendet. In [72, 73] wird die Trajektorienoptimierung als beschränktes Optimierungsproblem formuliert und numerisch mit entsprechenden Solvern gelöst. In den Nebenbedingungen des Optimierungsproblems lassen sich sowohl die Dynamik als auch Stellgrößenbeschränkungen berücksichtigen. Hindernisse werden in derartigen Ansätzen typischerweise durch Kreise oder Ellipsen approximiert und können ebenfalls als Nebenbedingung in das Optimierungsproblem integriert werden [74, 75]. Eine Alternative ist die Beschreibung des freien Raums um das Fahrzeug als ein konvexes Polygon [76]. In [77] werden darüber hinaus mehrere Iterationen durchgeführt, um eine Lösung basierend auf mehreren Teilproblem mit jeweils freien konvexen Bereichen zu finden.

Im Rahmen dieser Arbeit werden zahlreiche Veröffentlichungen zur Thematik der Bewegungsplanung unabhängig von der Fahrzeugklasse untersucht. In der Abbildung 3.4 ist eine Auswahl der Veröffentlichungen anhand mehrerer Eigenschaften in einer grafischen Übersicht zusammengestellt. Zusätzliche Informationen zu den ausgewählten Veröffentlichungen sind in den Tabellen 3.1, 3.2 und 3.3 aufgelistet. Im Diagramm sind die unterschiedlichen Arten der Planung durch gestrichelte Bereiche skizziert. Die Pfadplanung beschränkt sich auf den Bereich, in dem kein dynamisches Bewegungsmodell notwendig ist. Gleichzeitig können bei

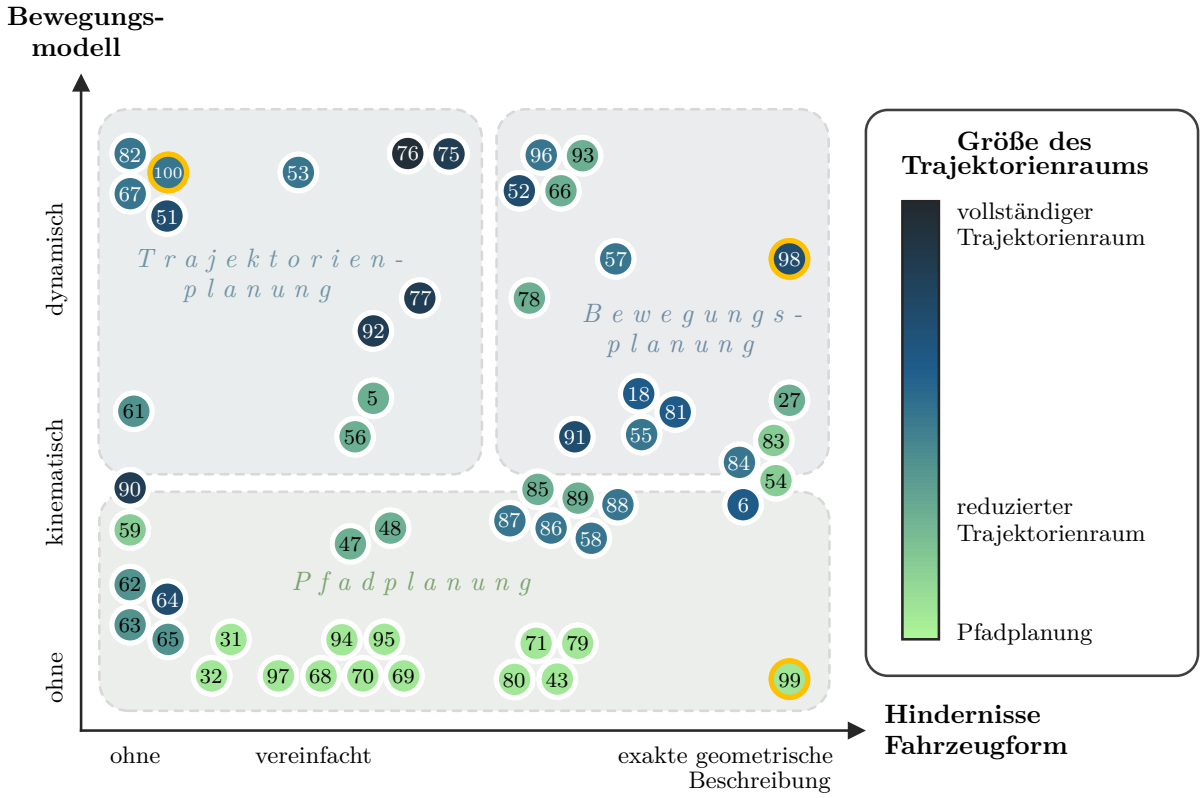


Abbildung 3.4.: Einordnung ausgewählter Verfahren zur Pfad-, Trajektorien- und Bewegungsplanung (orange: Veröffentlichungen des Autors).

der Pfadplanung Hindernisse sowie die Fahrzeugform exakt berücksichtigt werden. Wie in Abschnitt 2.2 bereits beschrieben, werden in der Trajektorienplanung zwar dynamische Modelle verwendet, jedoch spielt die Beachtung von Hindernissen nur eine untergeordnete Rolle. Im Gegensatz zur Trajektorienplanung steht in der Bewegungsplanung das Einbeziehen von Hindernissen an zentraler Stelle.

Entlang der Abszisse ist die geometrische Beschreibungsform für Hindernisse und die Fahrzeugform dargestellt. So wird die geometrische Form entweder vernachlässigt (ohne), durch Punkte, Rechtecke oder Kreise approximiert (vereinfacht) oder durch Polygone beschrieben (exakte geometrische Beschreibung). Die Art des verwendeten Bewegungsmodells ist auf der Ordinate skizziert. Entweder wird kein dynamisches Bewegungsverhalten berücksichtigt (ohne) oder es kommen kinematische oder dynamische Bewegungsmodelle zum Einsatz. Es werden unterschiedliche Klassen dynamischer Modelle verwendet, die von einem linearen Modell wie einem Doppelintegrator [55, 57, 78] bis hin zu Bewegungsmodellen basierend auf nichtlinearen gewöhnlichen Differenzialgleichungen reichen [66, 76].

Als dritte Eigenschaft ist die Größe des Trajektorienraums anhand der farblichen Markierung einer Veröffentlichung dargestellt. Wird das Problem der Bewegungsplanung unter der Annahme betrachtet, dass eine Lösung existiert, dann ist die Menge aller möglichen Trajektorien, welche Start und Ziel miteinander verbinden und in $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$ liegen, beliebig groß. Diese Menge wird in Abbildung 3.4 als Größe des Trajektorienraums bezeichnet. Verfahren, die das Planungsproblem als OCP formulieren, können den vollständigen Trajektorienraum für das Finden einer bestmöglichen Lösung nutzen. Durch das Optimierungsverfahren wird ein optimaler Stellgrößenverlauf erzeugt. Da prinzipiell beliebige Stellgrößenverläufe generiert werden können, lassen sich alle potenziellen Trajektorien erzeugen. Andere Verfahren nutzen nur eine Untermenge des Trajektorienraums, indem bestimmte Annahmen getroffen werden. Beispielsweise wird die Stell-

Ref.	Fahrzeug	Verfahren	Modell	Hindernis	Fahrzeugform
[5]	G	CL-RRT	dynamisch	2D-Gitter	Rechteck
[6]	G	Hybrid-A* / PS	kinematisch	Linie	Rechteck
[18]	P	RRT*	kinodynamisch	Rechteck	Punkt
[27]	G	RRT	kinodynamisch	Polygon	Polygon
[31]	G	APF	-	Kreis	Punkt
[32]	G	APF	-	Punkt	Punkt
[43]	P	RRT*	-	Rechteck	Punkt
[47]	G	A*	-	2D-Gitter	Punkt
[48]	G	D*	-	2D-Gitter	Punkt
[51]	A	NLP	dynamisch	-	-
[52]	A	RRT* / OCP	dynamisch	Rechteck	Punkt
[53]	A	Hybrid-A*	dynamisch	2D-Gitter	Punkt
[54]	G	RRT	dynamisch	Linie	Rechteck
[55]	G	RRT*	dynamisch	Linie	?
[56]	G	CL-RRT	dynamisch	2D-Gitter	Rechteck
[57]	G	NLP	kinematisch	Linie	Rechteck
[58]	G	OCP	kinematisch	Kreis	Punkt
[59]	S	RRT*	kinematisch	-	-
[61]	S	OCP	kinematisch	-	-
[62]	S	VO	kinematisch	-	-
[63]	S	VO / MILP	kinematisch	-	-
[64]	S	MQPSO	kinematisch	-	-
[65]	S	VO / RRT	kinematisch	-	-
[66]	S	RRT	dynamisch	Polygon	Punkt
[67]	S	A*	kinematisch	-	-
[68]	S	A*	-	2D-Gitter	Punkt
[69]	S	Dijkstra	-	2D-Gitter	Punkt
[70]	S	RRT	-	2D-Gitter	Punkt
[71]	S	RRT*	-	Polygon	?
[75]	S	A* / OCP	dynamisch	Ellipse	Punkt
[76]	S	OCP	dynamisch	Polygon	Polygon
[77]	S	OCP	dynamisch	Polygon	Polygon
[78]	P	CC-RRT	dynamisch	Polygon	Punkt
[79]	P	RRT*	-	Rechteck	Punkt
[80]	P	RT-RRT*	-	Rechteck	Punkt
[81]	A	RRT*	kinodynamisch	Linie, Rechteck	Kreis
[82]	A	NLP	dynamisch	-	-
[83]	G/A	RRT	dynamisch	Rechteck	Rechteck
[84]	G	Anytime RRT*	kinematisch	Rechteck	Rechteck
[85]	G	CL-RRT#	kinematisch	?	Punkt
[86]	G	RRT*FND	kinematisch	Rechteck, Kreis	Punkt
[87]	G	Flat-RRT*	kinematisch	Kreis	Punkt
[88]	G	MVRRT*	kinematisch	Rechteck	Punkt
[89]	G	DPRM*	kinematisch	Rechteck	Punkt
[90]	G	OCP	kinematisch	-	-
[91]	G	Poly-RRT*	kinematisch	Kreis	Punkt
[92]	G	OCP	kinematisch	2D-Gitter	Rechteck
[93]	G	CL-RRT	dynamisch	Linie	?
[94]	S	A* / PS	-	2D-Gitter	Punkt
[95]	S	FMM / VO	-	2D-Gitter	Punkt
[96]	S	A*	dynamisch	Rechteck	?
[97]	S	Vornonoi	-	Polygon	Punkt
[98]	S	RRT*	dynamisch	Polygon	Polygon
[99]	S	RRT*	-	Polygon	Polygon
[100]	S	CL-Sim	dynamisch	-	-

Tabelle 3.1.: Ausschnitt der Liste untersuchter Veröffentlichungen. Unbekannte Eigenschaften sind mit ? markiert.

Fahrzeug	Klasse	Beispiel
P	Partikel	Punktmasse
A	Luftfahrzeug (*aerial vehicle*)	Quadrocopter, Helikopter
G	Landfahrzeug (*ground vehicle*)	Rover, Differenzialradroboter, Automobil
S	Wasserfahrzeug (*surface vehicle*)	ASV, Schiff

Tabelle 3.2.: Liste der Fahrzeugklassen aus Tabelle 3.1.

Verfahren	Bezeichnung
APF	Artificial Potential Field
CC-RRT	Chance-Constrained RRT
CL-RRT	Closed-Loop RRT
CL-Sim	Closed Loop Simulation
DPRM*	Differential PRM*
FMM	Fast-Marching Method
MILP	Mixed Integer Linear Programming
MQPSO	Modified Quantum Particle Swarm Optimization
MVRRT*	Minimum-Violation RRT*
NLP	Nonlinear Programming
PS	Path Smoothing
RRT*FND	RRT* Fixed Nodes Dynamic
RT-RRT*	Real-Time RRT*
VO	Velocity Obstacle

Tabelle 3.3.: Liste einiger Verfahren aus Tabelle 3.1.

größe durch eine parametrisierte Funktion beschrieben. Eine Möglichkeit ist die Verwendung von Polynomen [51, 98, 101]. Durch die Beschränkung des Stellgrößenverlaufs auf eine spezielle Klasse von Funktionen reduziert sich die Anzahl an Optimierungsvariablen. Anstatt die Stellgröße selbst zu jedem Zeitpunkt über einen vorgegebenen Zeithorizont zu optimieren, werden die Parameter der Funktion optimiert. Als Folge reduziert sich der Trajektorienraum, da aufgrund der Limitierung auf spezielle Funktionen nicht mehr jeder beliebige Stellgrößenverlauf erzeugt werden kann. Ein weiteres Beispiel ist die Verwendung von *motion primitives*. Ein *motion primitive* ist eine Bewegungsvorlage und entspricht einem bestimmten Manöver wie beispielsweise Geradeausfahrt, Drehen oder dem Übergang zwischen diesen zwei Bewegungen. Durch die Aneinanderreihung passender Bewegungsvorlagen entsteht eine Trajektorie. Anwendungsbeispiele von *motion primitives* finden sich in [67, 82, 87, 96]. Aufgrund der beschränkten Anzahl der Manöver ergibt sich ein deutlich reduzierter Trajektorienraum. Verfahren zur Pfadplanung erzeugen keine Trajektorie. Demnach entspricht der zugehörige Trajektorienraum einer leeren Menge.

4. Problemanalyse und Lösungsansatz

In diesem Kapitel werden die Herausforderungen bezüglich der Bewegungsplanung für ein autonomes Wasserfahrzeug analysiert. Anhand eines konkreten Beispielszenarios werden die Problembeschreibung sowie die daraus resultierenden Anforderungen an ein Verfahren abgeleitet. Anschließend werden bekannte Verfahren aus der Literatur bezüglich der definierten Anforderungen diskutiert und ein entsprechender Lösungsansatz für die Problematik der Bewegungsplanung von ASVs wird vorgestellt.

4.1. Beispielszenario und Problembeschreibung

Die Bewegungsplanung ist in vielen Bereichen zu finden, besonders in der mobilen Robotik und in der Entwicklung selbstfahrender Autos. Im Bewegungsverhalten unterscheiden sich Wasserfahrzeuge jedoch deutlich von Landfahrzeugen. Viele für Landfahrzeuge getroffene Annahmen können für Wasserfahrzeuge nicht getroffen werden. Um die Problemstellungen für die Bewegungsplanung von Wasserfahrzeugen analysieren zu können, wird das folgende Beispiel betrachtet, welches in der Abbildung 4.1 gezeigt ist.

Ein hoch automatisiertes Binnenschiff soll selbstständig zu einer vorgegebenen Anlegeposition fahren und automatisch anlegen. Diese Mission lässt sich grob in drei Teilaufgaben unterteilen. In der Transitphase fährt das Binnenschiff mit einer nominalen Geschwindigkeit in Vorausrichtung. Es fährt entlang einer Bahn, die durch den Kanal- oder Flussverlauf vorgegeben ist. Bei der Einfahrt in ein Hafenbecken reduziert das Fahrzeug die Geschwindigkeit und die Manövrierphase beginnt. In dieser Phase werden alle zur Verfügung stehenden Antriebe verwendet, um sicher in der Nähe von Strukturen in allen Freiheitsgraden bei kleiner werdenden Geschwindigkeiten zu manövrieren. Dabei wird das Fahrzeug näher an die Anlegeposition gebracht. Die letzte Distanz bis zum Liegeplatz wird mithilfe der dynamischen Positionierung (DP) zurückgelegt.

Die vorliegende Arbeit betrachtet die Bewegungsplanung in der Phase des Manövrierens. Da diese Phase durch Bewegungen in der Nähe von Strukturen gekennzeichnet ist, bedarf es

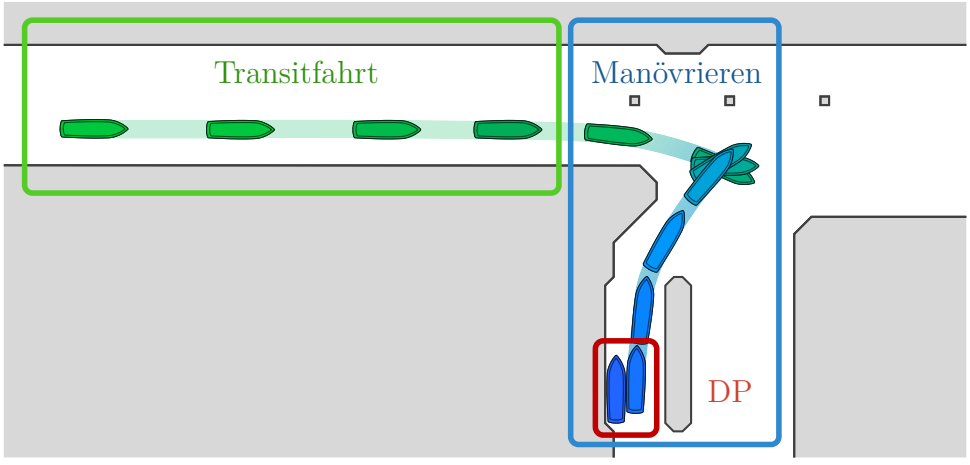


Abbildung 4.1.: Beispielhafte Unterteilung einer Mission in drei Phasen.

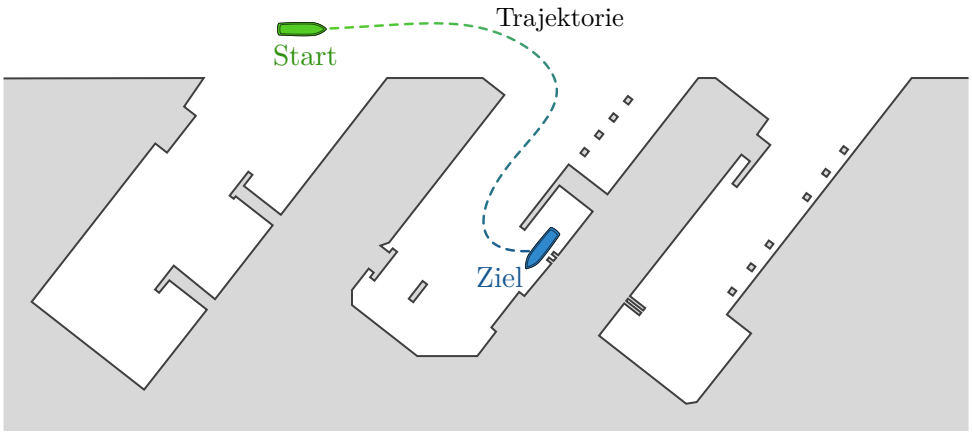


Abbildung 4.2.: Problemstellung anhand eines Beispielszenarios.

besonderer Aufmerksamkeit bei der Planung. Ein Hafenbecken ist durch Hindernisse wie Stege, Verankerungen, Brückenpfeiler, Kranfundamente und andere Konstruktionen charakterisiert, die beispielsweise zum Be- und Entladen von Schiffen notwendig sind. Es ergibt sich eine im Allgemeinen nicht-konvexe Form der Umgebung, in der ein Fahrzeugsystem eigenständig Aktionen planen muss, um ein Missionsziel zu erfüllen. Aus diesem Grund muss die genaue Hindernisstruktur sowie die exakte Form des ASVs einkalkuliert werden, damit der freie Raum bestmöglich und vor allem sicher genutzt werden kann.

4.2. Anforderungen an die Bewegungsplanung

Das Planungsproblem wird folgendermaßen zusammengefasst. Gegeben seien ein Anfangszustand und eine Zielpose sowie unbewegte Hindernisse. Der Anfangszustand enthält neben der Anfangspose auch die Anfangsgeschwindigkeiten sowie die aktuellen Antriebskräfte und -momente. Die Zielpose ist durch eine zweidimensionale Position sowie durch einen Heading-Winkel, der die Lage des Fahrzeuges beschreibt, charakterisiert. Hindernisse werden durch Polygone beschrieben. Abbildung 4.2 stellt die Manövrierphase dar und verdeutlicht die Problemstellung anhand eines Beispielszenarios. Die Aufgabe ist das Finden einer Trajektorie, die den Anfangszustand mit der Zielpose verbindet. Dabei müssen die folgenden Punkte beachtet werden.

1. **Kollisionsfreie Trajektorie**
   Die Trajektorie muss im freien Raum liegen und darf nicht mit Hindernissen kollidieren. Dabei muss jeder Zustand der Trajektorie kollisionsfrei sein.

2. **Fahrzeugform und -größe**
   Die geometrische Form des Fahrzeuges muss bei der Planung miteinbezogen werden. Das Fahrzeug wird dabei ebenfalls durch ein oder mehrere Polygone beschrieben.

3. **Fahrzeugdynamik**
   Für den zeitlichen Verlauf der Zustands- und Eingangsgrößen gelten differenzielle Beschränkungen. Das bedeutet, dass keine beliebige Abfolge von Zuständen möglich ist, sondern dass dieser Abfolge ein dynamisches Bewegungsmodell zugrunde liegen muss.

4. **Beschränkungen der Stellgröße**
   Die Stellgrößen des Fahrzeuges sind physikalisch und technisch beschränkt. Diese Beschränkungen müssen bei der Planung berücksichtigt werden, um Trajektorien zu generieren, die durch das Fahrzeug realisierbar sind.

5. **Bestmögliche Trajektorie**
   Die Menge aller realisierbaren Trajektorien kann beliebig groß sein. Aus diesem Grund muss eine bestmögliche Trajektorie anhand definierter Optimierungskriterien gefunden werden.

Diese Anforderungspunkte sind entsprechend ihrer Priorität geordnet und müssen beim Entwurf von Methoden zur Bewegungsplanung von Wasserfahrzeugen beachtet werden. Viele der untersuchten Methoden basieren auf Vereinfachungen, in denen einige der genannten Punkte unberücksichtigt bleiben.

4.3. Vergleich existierender Verfahren

In den letzten Jahrzehnten wurden viele Verfahren entwickelt, die hauptsächlich in der mobilen Robotik und im Bereich landbasierter Fahrzeuge zum Einsatz kommen, weshalb die Methoden für derartige Anwendungsfälle ausgelegt sind. Entsprechende Lösungspfade lassen sich beispielsweise sehr gut durch Geraden und Kreisbögen beschreiben, was für die meisten praktischen Anwendungen in diesem Bereich ausreichend ist. Oftmals kommen kinematische Modelle, die neben der Pose Geschwindigkeiten und Beschleunigungen berücksichtigen, zum Einsatz. Die Masse des Fahrzeuges sowie die Wirkung von Kräften bleibt unberücksichtigt.

Der große Unterschied zu Wasserfahrzeugen ist, dass diese ständig Umwelteinflüssen wie Wind und Strömung ausgesetzt sind, die mitunter starken Einfluss auf die Bewegung haben. Um ein Wasserfahrzeug unter diesen Bedingungen auf Position zu halten, ist der permanente Einsatz der Antriebsorgane notwendig. Ein stehendes Landfahrzeug verbleibt hingegen an seiner Position, ohne dass zusätzliche Antriebskräfte notwendig sind. Auf ein Wasserfahrzeug wirken zudem hydrodynamische Kräfte [102]. Für eine hinreichend genaue Beschreibung des Bewegungsverhaltens sind ausschließlich kinematische Modellansätze demzufolge nicht ausreichend.

Im Diagramm der Abbildung 3.4, welches eine Einordnung ausgewählter Verfahren zeigt, sind die genannten Anforderungen auf den Achsen aufgetragen und farblich gekennzeichnet. Das Berücksichtigen von Hindernissen sowie der geometrischen Fahrzeugform hat die höchste Priorität und ist entlang der Abszisse gekennzeichnet. Die nächsten wichtigen Anforderung sind das Einbeziehen der Fahrzeugdynamik sowie der Stellgrößenbeschränkung. Diese Eigenschaften werden entlang der Ordinate zusammengefasst. Die Anforderung mit der niedrigsten Priorität ist das Finden einer bestmöglichen Trajektorie anhand definierter Optimierungskriterien. Im Diagramm ist diese Eigenschaft durch eine Farbskala dargestellt. Ein Verfahren, welches alle genannten Anforderungen erfüllt und optimale Trajektorien erzeugt, läge im Diagramm oben rechts und wäre dunkelblau markiert. Theoretisch ist ein derartiges Verfahren durchaus denkbar. Ein Beispiel wäre ein RRT*-basierter Planer, der direkt den Phasenraum $\mathcal{X}$ sampelt und alle Zustände optimal durch das Lösen von OCPs verbindet. Aus praktischer Sicht ist das jedoch nicht sinnvoll, da enorm viele Samples erzeugt werden müssten, um einen hochdimensionalen Phasenraum ausreichend gut abzutasten, was zudem einen erheblichen Speicheraufwand mit sich bringt. Gleichzeitig würde die damit einhergehende Anzahl der zu lösenden OCPs dazu führen, dass das Problem nicht in akzeptabler Zeit gelöst werden kann.

Anhand des Diagramms aus Abbildung 3.4 ist zu erkennen, dass einige Verfahren Hindernisse unbeachtet lassen und andere Verfahren Approximationen verwenden. Mit Blick auf Tabelle 3.1 wird deutlich, dass Hindernisse häufig durch zweidimensionale Gitter und das Eigenfahrzeug als Punkt oder Rechteck dargestellt werden. Das erlaubt eine rechentechnisch schnelle Kollisionsbestimmung, die jedoch für die Bewegungsplanung von Wasserfahrzeugen in der Nähe von Strukturen aufgrund der beschränkten Auflösung des Gitters und der Missachtung der Fahrzeugform nicht ausreichend ist. Eine exakte geometrische Modellierung sowohl von Hindernissen als auch der Fahrzeugform durch beispielsweise Polygone findet nicht immer statt.

Dabei ist jedoch zu erwähnen, dass bei sampling-basierten Verfahren der Kollisionstest durch *black-box*-Funktionen realisiert wird, die sich problemlos austauschen ließen.

Für die Bewegungsplanung autonomer Wasserfahrzeuge sind nichtlineare dynamische Modelle notwendig. Verfahren, deren verwendete Modellstruktur für die Bewegungsplanung von Wasserfahrzeugen geeignet ist, finden sich daher im oberen Bereich des Diagramms. Durch die Formulierung des Planungsproblems als OCP lassen sich dynamische Modelle direkt benutzen. Gleichzeitig können Stellgrößenbeschränkungen beachtet und der gesamte Trajektorienraum genutzt werden, wie beispielsweise in [75, 76]. Jedoch ergeben sich Probleme bei der Umsetzung der ersten beiden Anforderungen. Der Konfigurationsraum ist dadurch charakterisiert, dass zwei Konfigurationen $\boldsymbol{q}_{obs} \in C_{obs}$ und $\boldsymbol{q}_{free} \in C_{free}$ beliebig dicht beieinander liegen können. Das bedeutet, dass eine kollisionsfreie Konfiguration durch eine minimale Veränderung zu einer Konfiguration führen kann, die eine Kollision zur Folge hat. Diese sprunghafte Änderung ist nicht differenzierbar und somit problematisch, da Solver typischerweise auf Ableitungen angewiesen sind, oder versuchen, diese zu schätzen. Als Alternative werden Hindernisse und auch das Eigenschiff häufig durch Kreise oder Ellipsen approximiert, da sich diese Art der Beschreibung gut in Nebenbedingungen des Optimierungsproblems integrieren lässt. Zudem werden Slack-Variablen verwendet, um Kollisionen durch zusätzliche Kosten zu bestrafen, anstatt sie durch Zustandsbeschränkungen auszuschließen. Im Diagramm in Abbildung 3.4 sind derartige Verfahren in der Mitte des oberen Bereichs zu finden. Verfahren, die dennoch eine exakte geometrische Beschreibung unter Berücksichtigung der Dynamik anstreben, weisen eine Reduktion des Trajektorienraums auf, wie beispielsweise in [52, 66, 93, 96].

Die Veröffentlichungen des Autors zur Thematik der Bewegungsplanung autonomer Wasserfahrzeuge sind in der Abbildung 3.4 orange umrandet. In [98] wird ein RRT*-basierter Bewegungsplaner realisiert, der einen 6-dimensionalen Phasenraum sampelt und Zustände mithilfe von Polynomen dritten Grades exakt miteinander verbindet. Die Parameter des Polynoms werden durch die Minimierung einer Kostenfunktion, welche den Zeithorizont und den Stellgrößenaufwand gewichtet, ermittelt. Es wird gezeigt, dass das Verfahren trotz der schnellen Berechnung eines optimalen Polynoms im Vergleich zum Lösen eines OCPs für die exakte Verbindung zweier Zustände aufgrund der dennoch hohen Rechenzeit nicht praktikabel ist. In [99] wird ein RRT*-basierter Pfadplaner vorgestellt, in dem sowohl Hindernisse als auch die geometrische Form des ASVs durch konvexe Polygone dargestellt werden. Zudem wird die Kostenfunktion dahingehend erweitert, dass der Abstand zu Hindernissen gewichtet wird, um das Risiko einer Kollision zu minimieren. Ein Verfahren zur Trajektoriengenerierung auf Basis eines gegebenen Pfades wird in [100] beschrieben. Mithilfe der exakten Linearisierung wird das nichtlineare Bewegungsmodell des ASVs auf Geschwindigkeitsebene linearisiert. Durch einen überlagerten Positionsregler wird in Kombination mit einem Guidance-Algorithmus aus dem Pfad eine Trajektorie generiert.

4.4. Lösungsansatz

Aus dem vorangestellten Vergleich lassen sich die folgenden Problemstellungen für die Bewegungsplanung von Wasserfahrzeugen ableiten. Verfahren aus dem Bereich landbasierter Fahrzeuge, die kinematische Modelle verwenden, sind für die Bewegungsplanung von ASVs ungeeignet. Im maritimen Bereich kommen unter anderem nichtlineare Optimierungsverfahren zum Einsatz, in denen sich das dynamische Bewegungsverhalten berücksichtigen lässt. Jedoch werden Hindernisse sowie die geometrische Form des ASVs nicht exakt berücksichtigt, was für das Manövrieren in der Nähe von Strukturen problematisch ist.

In dieser Arbeit wird ein neuartiges Verfahren zur Bewegungsplanung von ASVs vorgestellt, welches den beschriebenen Anforderungen aus 4.2 gerecht wird. Dazu wird ein sequenzieller

Ansatz verfolgt, bei dem das Planungsproblem in zwei Teilprobleme zerlegt wird. Zuerst wird ein optimaler kollisionsfreier Pfad mit dem RRT*-Algorithmus gesucht. Dieser Pfad dient als Ausgangspunkt für die nachfolgende Bewegungsplanung und legt gleichzeitig das zugehörige Suchgebiet fest. Der entwickelte Bewegungsplaner entspricht einer modifizierten Variante des Pfadplaners, bei dem die Berücksichtigung des dynamischen Bewegungsverhaltens innerhalb der *IsFeasible*-Funktion des RRT*-Algorithmus erfolgt. Mithilfe der numerischen Simulation eines geschlossenen Regelkreises werden aus den Zweigen des Baums Trajektorien generiert, die für den Kollisionstest verwendet werden. Das Regelsystem besteht aus einer Zustandsregelung, die auf dem Entwurf einer exakten Linearisierung [11, 12] aufbaut, um Nichtlinearitäten der Regelstrecke auf Geschwindigkeitsebene kompensieren.

Der Begriff der sequenziellen Planung ist kein festgelegter Begriff aus der Literatur und wird hier verwendet, um zu kennzeichnen, dass sich das entwickelte Verfahren zur Bewegungsplanung aus zwei aufeinanderfolgenden Teilverfahren zusammensetzt.

5. Sequenzielle Bewegungsplanung

Die Aufgabe der Bewegungsplanung besteht darin, die bestmögliche, kollisionsfreie Trajektorie auf der Grundlage der drei Eingangsgrößen Anfangszustand, Zielpose und Hindernisse zu berechnen. Demzufolge entspricht eine Trajektorie einem Bewegungsplan, mit dem das Fahrzeug vom aktuellen Bewegungszustand zur gegebenen Zielpose gebracht wird, ohne dass dabei Kollisionen mit Hindernissen entstehen. In der Praxis ändern sich diese Eingangsgrößen mit der Zeit, beispielsweise aufgrund neu erkannter Hindernisse oder einer neu vorgegebenen Zielpose. Bei jeder Veränderung muss ein der Situation angepasster Bewegungsplan generiert werden. Die Berechnung eines Bewegungsplans, der den Anforderungen aus 4.2 gerecht wird, stellt ein rechentechnisch aufwändiges Problem dar. Aus diesem Grund wird ein sequenzieller Ansatz verfolgt, bei dem das Ausgangsproblem in zwei Teilprobleme zerlegt wird. Das Konzept wird im folgenden Abschnitt beschrieben.

5.1. Konzept der sequenziellen Bewegungsplanung

Im Alltag lösen Menschen das Problem der Bewegungsplanung intuitiv, wie beim Autofahren. Die generelle Vorgehensweise lässt sich in Algorithmen übertragen. Möchte eine Person an einen bestimmten Ort gelangen, dann führt diese Person unbewusst eine Planung in drei aufeinanderfolgenden Schritten aus. Zunächst orientiert sich die Person an einer Karte und sucht nach einem im Allgemeinen kurzen Weg zum Ziel. Entlang des geplanten Weges beobachtet die Person die unmittelbare Umgebung und sucht intuitiv nach dem bestmöglichen Pfad, um dem nächsten Teilziel des geplanten Weges näher zu kommen und dabei gleichzeitig erkannten Hindernissen auszuweichen. Für einen gewissen Zeithorizont in die Zukunft plant die Person anschließend die notwendigen Lenkbewegungen, um beispielsweise ein Hindernis zu umfahren.

Diese Art der Problemaufteilung wird auf die sequenzielle Bewegungsplanung von Wasserfahrzeugen übertragen. Das hat den Vorteil, dass zum einen die Einzelprobleme rechentechnisch effizienter zu lösen sind und zum anderen das Verfahren für den Anwender transparenter wird. Die Abbildung 5.1 zeigt die drei aufeinanderfolgenden Schritte zur Lösung des Planungsproblems.

Zu Beginn erfolgt die **globale Wegplanung** anhand einer Karte, die ein Netzwerk aus Wasserstraßen darstellt. Aus algorithmischer Sicht lässt sich das Wasserstraßennetz als Graph beschreiben und mit entsprechenden Werkzeugen aus der Graphentheorie analysieren, um den kürzesten Weg zu finden. An dieser Stelle werden keine Hindernisse und auch kein dynamisches Bewegungsverhalten des Fahrzeuges berücksichtigt. Die lokalen Hindernisse an verschiedenen Stellen entlang des geplanten Weges sind noch nicht bekannt. Ebenso wenig ist es notwendig, die exakte Trajektorie für das Umfahren eines weit entfernten Hindernisses zu planen. Das Ergebnis der globalen Wegplanung besteht aus Aktionen, wie einer Richtungsänderung an einer Mündung.

Im Anschluss erfolgt die **lokale Pfadplanung**. Der Bereich für die Pfadplanung beschränkt sich beispielsweise auf ein Gebiet, welches mit der Umfelderkennung beobachtet werden kann. In diesem Gebiet wird nach dem bestmöglichen, kollisionsfreien Pfad gesucht, um dem nächsten Teilziel der globalen Wegplanung näher zu kommen. Dabei geht es konkret um die Fragestellung, wie Hindernissen mit größtmöglichem Abstand ausgewichen werden kann. Das Bewegungsverhalten spielt hierbei ebenfalls noch keine Rolle.

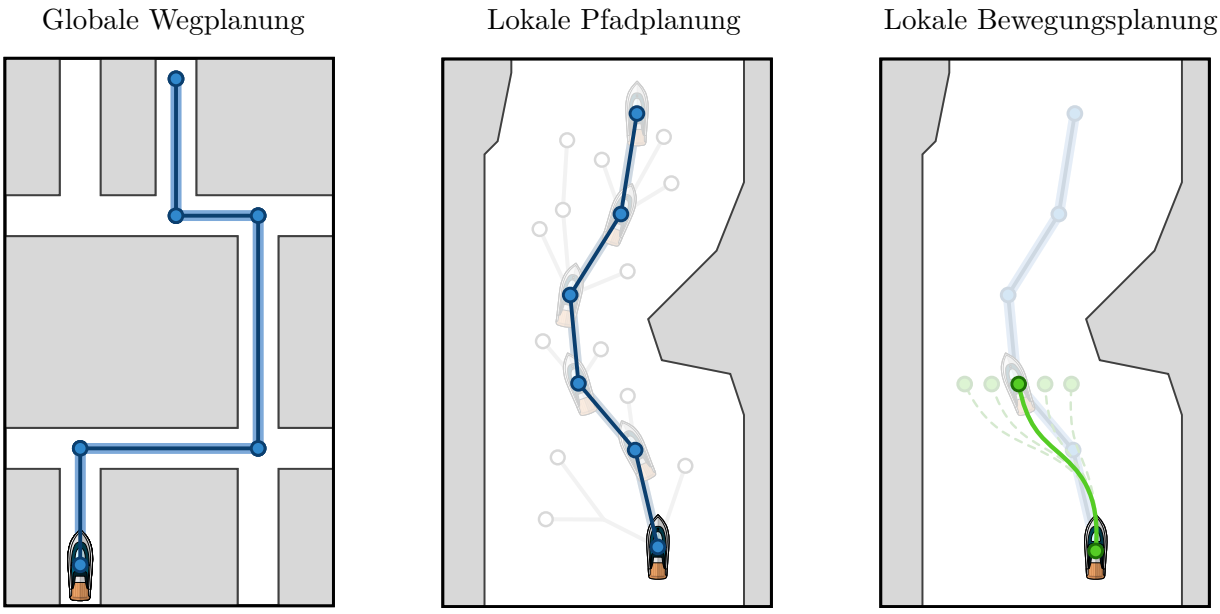


Abbildung 5.1.: Strategie der sequenziellen Bewegungsplanung in drei aufeinanderfolgenden Schritten.

Der resultierende Pfad wird in der **lokalen Bewegungsplanung** als Führungslinie verwendet. Innerhalb eines beschränkten Bereichs wird nach einer bestmöglichen, kollisionsfreien Trajektorie in der Nähe der Führungslinie gesucht. Neben den umliegenden Hindernissen wird das dynamische Bewegungsverhalten des Fahrzeuges einbezogen. Gleichzeitig werden Beschränkungen der Stellgröße berücksichtigt.

Da sich diese Arbeit auf die Phase des Manövrierens konzentriert, wie in Abschnitt 4.1 vorgestellt, ist keine globale Wegplanung notwendig. Der globale Plan entspräche der Anlegeposition. Deshalb werden nur die zwei Teilschritte der lokalen Pfad- und Bewegungsplanung betrachtet. In der Pfadplanung, auf die im Detail in Abschnitt 5.3 eingegangen wird, kommt der RRT*-Algorithmus zur Anwendung. Durch eine neuartige Erweiterung der Kostenfunktion wird der Abstand zu Hindernissen gewichtet. Zudem lassen sich Querbewegungen gegenüber Längsbewegungen bereits während der Pfadplanung bestrafen. Der Algorithmus für die Bewegungsplanung basiert auf dem der Pfadplanung, welcher um zusätzliche Funktionalitäten erweitert wird, die das Bewegungsverhalten einbeziehen. Dazu werden aus den einzelnen Zweigen des Baums Trajektorien auf Basis einer numerischen Simulation generiert. Neben einem dynamischen Modell des Fahrzeuges kommt in der Simulation ein Regelsystem zum Einsatz. Auf das Bewegungsmodell wird in Abschnitt 5.2 eingegangen und die Bewegungsplanung wird in Abschnitt 5.4 detailliert beschrieben. Eine Besonderheit des entwickelten Verfahrens ist die Einbettung des Bewegungsverhaltens in die *IsFeasible*-Funktion des RRT*-Algorithmus, die als *black-box*-Funktion für den Kollisionstest dient. Der Bewegungsplan muss kontinuierlich neu berechnet werden, um veränderte Informationen bezüglich der Umgebung einbeziehen zu können. In Abschnitt 5.5 wird eine Vorgehensweise beschrieben, um online fortlaufend Trajektorien zu generieren und diese in geeigneter Weise zusammenzufügen. Für das *Control-System* entsteht eine glatte Referenztrajektorie ohne Unstetigkeiten in den Zustandsgrößen und ohne Sprünge in der Stellgröße.

5.2. Bewegungsmodell für traversierfähige Wasserfahrzeuge

Autonom agierende Systeme müssen ein hohes Maß an Zuverlässigkeit aufweisen. Besonders in der Nähe von Strukturen, wo präzises Manövrieren erforderlich ist, müssen Fahrzeuge jederzeit in der Lage sein, auf Gefahrensituationen zu reagieren. Als Fahrzeugklasse werden deshalb traversierfähige Fahrzeuge vorausgesetzt, die sich aufgrund ihrer Antriebskonfiguration unabhängig in allen drei Freiheitsgraden (DoF) in der Welt $\mathcal{W} = \mathbb{R}^2$ bewegen können. Das Bewegungsverhalten wird durch das nichtlineare Differenzialgleichungssystem

$$\dot{\boldsymbol{\eta}} = \boldsymbol{T}_b^n(\psi)\boldsymbol{\nu} \tag{5.1}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \tag{5.2}$$

$$\dot{\boldsymbol{\tau}} = \boldsymbol{A}_\tau\boldsymbol{\tau} + \boldsymbol{B}_\tau\boldsymbol{\tau}_c \tag{5.3}$$

beschrieben. Hierbei ist $\boldsymbol{\eta} = (x, y, \psi)^{\mathrm{T}}$ die Pose, wobei $(x, y)^{\mathrm{T}}$ die Position des *b-frame* bezüglich des *n-frame* kennzeichnet und ψ den Heading-Winkel beschreibt. Die Geschwindigkeit des *b-frame* bezüglich des *n-frame*, gegeben in körperfesten Koordinaten, ist mit $\boldsymbol{\nu} = (u, v, r)^{\mathrm{T}}$ bezeichnet, wobei u und v die translatorischen Geschwindigkeiten in Längs- und Querrichtung beschreiben und r die Winkelgeschwindigkeit um die z^b-Achse kennzeichnet, wie in Abbildung 2.1 dargestellt. Der körperfeste Kräfte-Momente-Vektor wird mit $\boldsymbol{\tau} = (X, Y, N)^{\mathrm{T}}$ bezeichnet, wobei X und Y die Kraft in Längs- bzw. Querrichtung und N das Moment um die z^b-Achse kennzeichnet. Die Kombination aus Pose, Geschwindigkeit und Kraft bildet den 9-dimensionalen Bewegungszustand

$$\boldsymbol{x} = \left(\boldsymbol{\eta}^{\mathrm{T}},\ \boldsymbol{\nu}^{\mathrm{T}},\ \boldsymbol{\tau}^{\mathrm{T}}\right)^{\mathrm{T}} . \tag{5.4}$$

Gleichung (5.1) stellt einen kinematischen Zusammenhang zwischen der Pose $\boldsymbol{\eta}$ und der Geschwindigkeit $\boldsymbol{\nu}$ her.

$$\boldsymbol{T}_b^n(\psi) = \begin{pmatrix} \boldsymbol{R}_b^n(\psi) & \boldsymbol{0}_{2\times 1} \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.5}$$

ist die Transformationsmatrix, mit deren Hilfe vektorielle Größen vom *b-frame* ins *n-frame* transformiert werden. Das Geschwindigkeitsmodell (5.2) entspricht den Newtonschen Gesetzen und wird hier in einer verallgemeinerten Form verwendet, die der gängigen Systembeschreibung nichtlinearer eingangsaffiner Systeme entspricht. Hierbei ist $\boldsymbol{f}(\boldsymbol{\nu})$ eine beliebig nichtlineare Funktion. Für den Entwurf einer exakten Linearisierung wird jedoch gefordert, dass die zweite partielle Ableitung von $\boldsymbol{f}(\boldsymbol{\nu})$ bezüglich $\boldsymbol{\nu}$ existiert. Zudem muss die quadratische Eingangsmatrix $\boldsymbol{B}$ invertierbar sein. Diese Annahme deckt sich jedoch direkt mit der Voraussetzung traversierfähiger Fahrzeuge, bei denen die Eingangsmatrix $\boldsymbol{B}$ vollen Rang besitzt.

Durch eine Allokation innerhalb des *Control-System* müssen die geforderten Kräfte und Momente auf die zur Verfügung stehenden Antriebsorgane verteilt werden. Dabei besitzen die Antriebe ebenfalls dynamisches Verhalten, was dazu führt, dass sich eine geforderte Kraft nicht in unendlich kurzer Zeit realisieren lässt. Im Rahmen der Bewegungsplanung wird ein Bewegungsmodell auf Kräfte-Momente-Basis verwendet, da auf diese Weise keine fahrzeug-spezifischen Antriebskonfigurationen beachtet werden müssen und das Verfahren zur Bewegungsplanung ohne strukturelle Änderungen universell für verschiedene Fahrzeuge angewendet werden kann. Um dennoch die Antriebsdynamik berücksichtigen zu können und gleichzeitig auf einer allgemeinen Kräfte-Momente-Ebene zu bleiben, wird das Kraftmodell (5.3) eingeführt, welches als lineares Zustandsraummodell mit der Systemmatrix $\boldsymbol{A}_\tau$ und der Eingangsmatrix $\boldsymbol{B}_\tau$, die ebenfalls vollen Rang besitzt, realisiert ist. Hierbei ist $\boldsymbol{\tau}_c = (X_c, Y_c, N_c)^{\mathrm{T}}$ der Eingangsvektor und entspricht der geforderten oder kommandierten Kraft. Die Antriebsdynamik muss auf eine vergleichbare Kräfte-Momente-Dynamik abgebildet werden. Weiterhin ist es denkbar, dass ein Allokationssystem so ausgelegt wird, dass sich ein gewünschtes Verhalten zwischen der geforderten Kraft

$\boldsymbol{\tau}_c$ und der realisierten Kraft $\boldsymbol{\tau}$ ergibt. Für das Kraftmodell wird ein Verzögerungsglied erster Ordnung mit

$$\boldsymbol{A}_\tau = \begin{pmatrix} -\frac{1}{T_X} & 0 & 0 \\ 0 & -\frac{1}{T_Y} & 0 \\ 0 & 0 & -\frac{1}{T_N} \end{pmatrix}, \quad \boldsymbol{B}_\tau = \begin{pmatrix} \frac{1}{T_X} & 0 & 0 \\ 0 & \frac{1}{T_Y} & 0 \\ 0 & 0 & \frac{1}{T_N} \end{pmatrix} \tag{5.6}$$

und den zugehörigen Zeitkonstanten T_X, T_Y und T_N definiert, bei dem $\boldsymbol{\tau}$ und $\boldsymbol{\tau}_c$ stationär identisch sind.
Mit der Funktion $\boldsymbol{f}(\boldsymbol{\nu})$ im Geschwindigkeitsmodell (5.2) werden verschiedene Effekte modelliert wie beispielsweise lineare und nichtlineare Dämpfung, aber auch die nichtlinearen Querkopplungen, die sich zwangsläufig aus der Transformation der inertialen Bewegungsgleichungen in das körperfeste Koordinatensystem ergeben. Die Funktion wird in der Form

$$\boldsymbol{f}(\boldsymbol{\nu}) = \boldsymbol{F} \cdot \boldsymbol{n}(\boldsymbol{\nu}) \tag{5.7}$$

definiert. Dabei ist $\boldsymbol{F} \in \mathbb{R}^{3\times m}$ eine Koeffizienten-Matrix, welche die konstanten Modellparameter enthält, und $\boldsymbol{n}(\boldsymbol{\nu}) \in \mathbb{R}^m$ ist der Vektor, der verschiedene Kombinationen aus Geschwindigkeitskomponenten beinhaltet. Alle linearen und nichtlinearen Terme der Bewegungsgleichung werden somit in einem m-dimensionalen Vektor zusammengefasst. Für ein ASV kann der nichtlineare Vektor beispielsweise mit

$$\boldsymbol{n}(\boldsymbol{\nu}) = (u,\ v,\ r,\ vr,\ ur,\ uv,\ r^2,\ u^3,\ v^3,\ r^3)^{\mathrm{T}}$$

beschrieben werden. Die einzelnen Elemente basieren auf physikalischen Modellen aus der Literatur [102]. Ein ausführlicher Vergleich der verwendeten Modellstruktur mit Bewegungsmodellen aus der Literatur ist in Appendix A.1 zu finden. Durch diese Art der Modellierung hat der Entwickler prinzipiell die Freiheit, sich eine gewünschte nichtlineare Struktur definieren zu können, mit denen der Prozess ausreichend gut abgebildet werden kann. Im weiteren Verlauf der Arbeit wird daher von der allgemeinen Form (5.7) ausgegangen.

5.3. Pfadplanung

Die Pfadplanung übernimmt die erste Teilaufgabe der sequenziellen Bewegungsplanung. Die Konfiguration eines Fahrzeuges wird mit

$$\boldsymbol{q} = (x,\ y,\ \psi)^{\mathrm{T}} \tag{5.8}$$

bezeichnet und entspricht der Pose $\boldsymbol{\eta}$. Die Start- und Zielkonfiguration werden mit $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ gekennzeichnet. Die geometrische Form des Fahrzeuges an einer bestimmten Konfiguration wird durch das Polygon $\mathcal{V}(\boldsymbol{q})$ dargestellt und alle unbewegten Hindernisse werden in $\mathcal{O}$ zusammengefasst. In Abbildung 5.2 ist die Problemstellung der Pfadplanung verdeutlicht. Für eine gegebene Start- und Zielkonfiguration ist der Pfad $\boldsymbol{\Pi}$ gesucht, der beide Konfigurationen miteinander verbindet und entlang dessen es zu keiner Überlagerung der geometrischen Fahrzeugform mit gegebenen unbewegten Hindernissen kommt. Ein Pfad $\boldsymbol{\Pi} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_N\}$ entspricht einer geordneten Liste aus Konfigurationen, wobei $\boldsymbol{q}_1 \equiv \boldsymbol{q}_I$ ist und der Abstand von $\boldsymbol{q}_N$ zu $\boldsymbol{q}_G$ minimal ist. Falls der Lösungspfad das Ziel erreicht, gilt $\boldsymbol{q}_N \equiv \boldsymbol{q}_G$.

Das entwickelte Verfahren zur Pfadplanung basiert auf dem RRT*-Algorithmus und ist in Algorithmus 5 dargestellt. Neben den gegebenen Werten $\boldsymbol{q}_I$, $\boldsymbol{q}_G$ und $\mathcal{O}$ wird der Prozedur eine begrenzte Rechenzeit t^p_{max} sowie eine Positionsdifferenz $\Delta\boldsymbol{p} = (\Delta x, \Delta y)^{\mathrm{T}}$ übergeben. Die Rechenzeit einer Iteration hängt signifikant von der Anzahl der Hindernisse sowie der Kardinalität des Baums ab. Aus diesem Grund wird nicht mit einer festen Anzahl von Iterationen gerechnet, sondern die Berechnung wird beim Überschreiten des vorgegebenen Zeitintervalls abgebrochen.

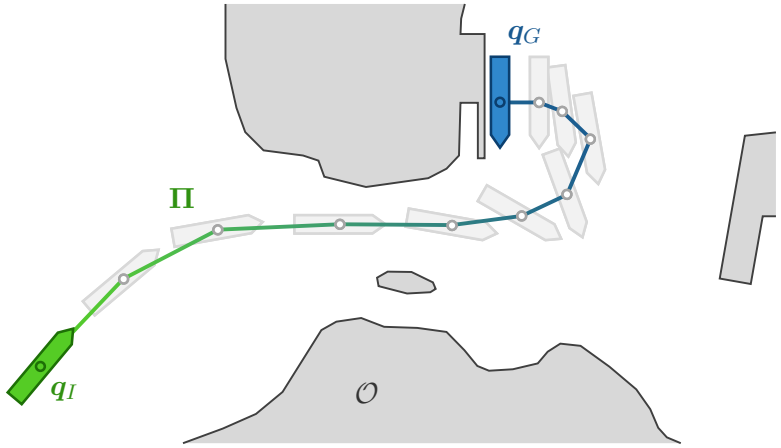


Abbildung 5.2.: Beispielhafte, schematische Darstellung des Pfadplanungsproblems für ein ASV.

Die Pfadplanung erfolgt in einem euklidischen Koordinatensystem. Werden aufeinanderfolgend Planungsprobleme gelöst, bei denen sich der Koordinatenursprung verschiebt, muss diese Verschiebung in Form einer translatorischen Positionsdifferenz $\Delta\boldsymbol{p}$ dem Pfadplaner übergeben werden, damit ein korrekter Warmstart, der die Pfadplanung initialisiert, durchgeführt werden kann. Das Lösen von Planungsproblemen in verschobenen euklidischen Koordinatensystemen wird später von Bedeutung sein, wenn die Bewegungsplanung innerhalb verschiedener Tangentialebenen auf dem Erdellipsoid erfolgt. Wird die Planung ausschließlich in einem euklidischen Koordinatensystem durchgeführt, ohne den Koordinatenursprung zu verschieben, kann $\Delta\boldsymbol{p} = \mathbf{0}$ gesetzt werden.

Algorithmus 5: Pfadplanung

Data: $\mathcal{T} \leftarrow \varnothing$

1: **procedure** $\boldsymbol{\Pi} \leftarrow$ PathPlanner$(\boldsymbol{q}_I, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p}, t^p_{max})$
2: $\quad t_{start} \leftarrow$ TimeNow();
3: $\quad$ **if** $\mathcal{V}(\boldsymbol{q}_I) \cap \mathcal{O} = \emptyset$ **then**
4: $\quad\quad \mathcal{T} \leftarrow$ WarmStart$(\mathcal{T}, \boldsymbol{q}_I, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p})$; // see algorithm 11
5: $\quad\quad error \leftarrow$ FALSE;
6: $\quad\quad$ **while** $\neg error \ \wedge \ ((\text{TimeNow}() - t_{start}) < t^p_{max})$ **do**
7: $\quad\quad\quad (\mathcal{T}, error) \leftarrow$ PathPlannerIteration$(\mathcal{T}, \boldsymbol{q}_G, \mathcal{O})$; // see algorithm 6
8: $\quad\quad$ **end**
9: $\quad$ **end**
10: $\quad \boldsymbol{q}_{solution} \leftarrow$ NearestNeighbor$(\mathcal{T}, \boldsymbol{q}_G)$;
11: $\quad \boldsymbol{\Pi} \leftarrow \mathcal{T}$.GetBranch$(\boldsymbol{q}_{solution})$;
12: **end**

Das Verfahren zur Pfadplanung startet mit einem leeren Baum $\mathcal{T}$, der keine Knoten enthält. Zu Beginn wird der aktuelle Startzeitpunkt ermittelt (Zeile 2). Anschließend wird das geforderte Planungsproblem auf Machbarkeit überprüft, indem die Startkonfiguration auf Kollision getestet wird (Zeile 3). Wenn die Startkonfiguration bereits zu einer Kollision führt, ist das Planungsproblem nicht lösbar, da es keine Möglichkeit gibt, zufällige Samples kollisionsfrei mit $\boldsymbol{q}_I$ zu verbinden. Mithilfe der entwickelten *WarmStart*-Prozedur (Zeile 4) wird der Baum $\mathcal{T}$ initialisiert. Dabei werden möglichst viele Knoten des Baums aus vorangegangenen Iterationen behalten. An dieser Stelle wird $\boldsymbol{q}_I$ als Wurzelknoten des Baums gesetzt. Ist der Warmstart nicht erfolgreich, dann enthält der Baum ausschließlich den Wurzelknoten. In Zeile 6 ist die Iterationsschleife dargestellt, welche unterbrochen wird, sobald ein Iterations-Fehler auftritt oder die vorgegebene Rechenzeit t^p_{max} überschritten wird. Der eigentliche Iterationsschritt

erfolgt in Zeile 7 und ist in Algorithmus 6 detailliert dargestellt. Abschließend wird der beste Lösungsknoten $\boldsymbol{q}_{solution}$ aus dem Baum ermittelt (Zeile 10) und der zugehörige Lösungspfad $\boldsymbol{\Pi}$ bestimmt (Zeile 11). Hierbei ist derjenige Knoten, welcher den kleinsten Abstand zum Zielknoten $\boldsymbol{q}_G$ hat, als bester Lösungsknoten definiert.

Ein Iterationsschritt der Pfadplanung ist in Algorithmus 6 dargestellt. Auf die einzelnen Funktionen wird in den nachfolgenden Unterabschnitten eingegangen. Im Vergleich mit Algorithmus 2 ist zu erkennen, dass der Iterationsschritt dem des RRT*-Algorithmus entspricht. Die maximale Anzahl an Knoten, die in $\mathcal{T}$ gespeichert werden können, ist beschränkt und wird mit n^p_{max} bezeichnet. In den Zeilen 3 und 4 wird ein zufälliger Blattknoten des Baums entfernt, falls der Baum die maximale Anzahl an Knoten erreicht hat, wodurch es theoretisch möglich ist, unendlich viele Iterationen bei physikalisch beschränktem Speicher durchzuführen. Diese Strategie entspricht der bereits erwähnten Erweiterung RRT*FN [46]. Jedoch wird die *RemoveRandomLeaf*-Funktion dahingehend modifiziert, dass nur Blattknoten entfernt werden, die nicht Teil des aktuellen Lösungspfades sind. Hierbei entspricht der aktuelle Lösungspfad ebenfalls dem Zweig, dessen Blattknoten den kleinsten Abstand zu $\boldsymbol{q}_G$ besitzt. Falls kein Blattknoten aus dem Baum entfernt werden konnte, gibt die *RemoveRandomLeaf*-Funktion den Wert $error = \textsc{True}$ zurück. In diesem Fall sind alle Knoten aus $\mathcal{T}$ Teil des aktuellen Lösungspfads und die Iteration kann beendet werden. In Zeile 9 erfolgt das Generieren eines zufälligen Samples $\boldsymbol{q}_{rand}$. Zu diesem Sample wird unter Verwendung einer Distanzmetrik der dichteste Knoten aus $\mathcal{T}$ ausgewählt (Zeile 10). Die *Steer*-Funktion erweitert den Baum von $\boldsymbol{q}_{nearest}$ zu $\boldsymbol{q}_{rand}$ und erzeugt $\boldsymbol{q}_{new}$ (Zeile 11). Das resultierende Pfadstück $\boldsymbol{\varepsilon}$ wird in Zeile 12 auf Kollision mit Hindernissen $\mathcal{O}$ getestet. In Zeile 13 wird die Menge aller Knoten V aus $\mathcal{T}$ ermittelt, welche $\boldsymbol{q}_{new}$ am dichtesten sind. Aus dieser Menge wird anschließend der beste Elternknoten ermittelt (Zeile 14) und die neue Verbindung wird dem Baum hinzugefügt (Zeile 15). In Zeile 16 wird untersucht, ob der neu hinzugefügte Knoten $\boldsymbol{q}_{new}$ selbst ein besserer Elternknoten für einen der Knoten aus der Nachbarschaft sein kann.

Algorithmus 6: Ein Iterationsschritt der Pfadplanung

```
1:  procedure (T, error) ← PathPlannerIteration(T, q_G, O)
2:      error ← False;
3:      if T.IsFull() then
4:          error ← T.RemoveRandomLeaf();
5:          if error then
6:              return;
7:          end
8:      end
9:      q_rand ← Sample(T, q_G);                          // see algorithm 7
10:     q_near ← NearestNeighbor(T, q_rand);
11:     (q_new, ε) ← Steer(T, q_near, q_rand);            // see algorithm 8
12:     if IsFeasible(ε, O) then
13:         V ← Near(T, q_new);
14:         q_parent ← FindBestParent(V, q_near, q_new);  // see algorithm 3
15:         T.Insert(q_parent, q_new);
16:         Rewire(T, V \ q_parent, q_new);               // see algorithm 4
17:     end
18: end
```

5.3.1. Distanzmetrik für SE(2)

Der Konfigurationsraum $\mathcal{C} = SE(2)$ entspricht einer dreidimensionalen Mannigfaltigkeit und ist lokal mit dem euklidischen Raum $\mathbb{R}^3$ vergleichbar, wobei jedoch die dritte Dimension zur Darstellung des Winkels eine Besonderheit besitzt. Die beiden Konfigurationen $(x, y, -\pi)^{\mathrm{T}}$ und $(x, y, +\pi)^{\mathrm{T}}$ sind identisch. Erfolgt die Bewegung entlang der Winkeldimension und wird eine Grenze bei $\pm\pi$ überschritten, dann springt die Winkelgröße zur gegenüberliegenden Grenze. Der resultierende topologische Raum wird auch als Mannigfaltigkeit mit Rand bezeichnet. Diese Eigenschaft muss bei der Pfadplanung berücksichtigt werden. Dazu wird die Funktion

$$\sigma(\psi) = \begin{cases} \mu(\psi) - 2\pi & \text{falls } \mu(\psi) \geq +\pi \\ \mu(\psi) + 2\pi & \text{falls } \mu(\psi) < -\pi \\ \mu(\psi) & \text{sonst} \end{cases} \tag{5.9}$$

definiert, welche den Winkelbereich auf $[-\pi, +\pi)$ abbildet. Hierbei entspricht

$$\mu(\psi) = \psi - 2\pi \ \mathrm{sgn}(\psi) \left\lfloor \frac{|\psi|}{2\pi} \right\rfloor \tag{5.10}$$

der Modulo-Funktion. Zudem wird die Funktion

$$\boldsymbol{\sigma}(\boldsymbol{q}) = \begin{pmatrix} x \\ y \\ \sigma(\psi) \end{pmatrix} \tag{5.11}$$

eingeführt, mit deren Hilfe (5.9) ausschließlich auf Winkelgrößen des Funktionsarguments angewendet wird. Um den Abstand zwischen zwei beliebigen Konfigurationen berechnen zu können, muss eine Distanzmetrik definiert werden, wobei die Eigenschaften entsprechend der Definition 1 gelten müssen [15].

Definition 1. *Eine metrische Funktion $\rho : X \times X \mapsto \mathbb{R}$ entspricht einer Abbildung des topologischen Raums X auf einen skalaren Wert, wobei für beliebige Elemente $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in X$ die folgenden Eigenschaften für eine Distanzfunktion gelten müssen:*

1) Positive Definitheit: $\rho(\boldsymbol{a}, \boldsymbol{b}) \geq 0$

2) Reflexivität: $\rho(\boldsymbol{a}, \boldsymbol{b}) = 0 \Longleftrightarrow \boldsymbol{a} = \boldsymbol{b}$

3) Symmetrie: $\rho(\boldsymbol{a}, \boldsymbol{b}) = \rho(\boldsymbol{b}, \boldsymbol{a})$

4) Dreiecksungleichung: $\rho(\boldsymbol{a}, \boldsymbol{b}) + \rho(\boldsymbol{b}, \boldsymbol{c}) \geq \rho(\boldsymbol{a}, \boldsymbol{c})$

Als metrische Distanzfunktion wird die gewichtete euklidische Metrik

$$\rho(\boldsymbol{q}_0, \boldsymbol{q}_1) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (w_\psi \sigma(\psi_1 - \psi_0))^2} \tag{5.12}$$

verwendet. Hierbei ist $w_\psi > 0$ ein skalarer Parameter mit dem der Winkelabstand gegenüber dem Positionsabstand gewichtet wird. Nach der Definition 1 darf w_ψ nicht den Wert 0 annehmen, da dies die Reflexivitätseigenschaft nicht erfüllen würde. Die Distanzmetrik wird benutzt, um beispielsweise den dichtesten Knoten zum Baum innerhalb der *NearestNeighbor*-Funktion zu finden oder eine Menge der dichtesten Nachbarschaftsknoten mit der *Near*-Funktion zu ermitteln.

5.3.2. Kostenfunktion für die Pfadplanung

Jedem Knoten des Baums wird ein skalarer positiver Kostenwert c zugeordnet, der die kumulierten Kosten ausgehend vom Wurzelknoten entlang des Zweiges bis zum entsprechenden Knoten

angibt. Der Wurzelknoten selbst besitzt den konstanten Kostenwert $c = 0$. Mithilfe des Kostenwertes wird innerhalb der *FindBestParent*- und *Rewire*-Prozeduren geprüft, ob alternative Verbindungen zwischen benachbarten Knoten zu geringeren Kosten für einen Knoten führen.

Als Kostenfunktion kann prinzipiell die Distanzmetrik (5.12) gewählt werden. Ein optimaler Lösungspfad entspräche dann der kürzesten Verbindung zwischen $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$. Ist keine direkte Verbindung von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ aufgrund von Hindernissen möglich, dann wäre die optimale Lösung, sofern sie existiere, dadurch charakterisiert, dass sich $\mathcal{V}(\boldsymbol{q})$ und $\mathcal{O}$ unendlich dicht kämen. Mit anderen Worten: Ein optimaler Pfad läge im Allgemeinen unendlich dicht an Hindernissen. Für Wasserfahrzeuge ist diese Eigenschaft problematisch, da sie permanent externen Störungen wie Wind und Strömung ausgesetzt sind. Das Manövrieren unter Beachtung der Eigendynamik erfordert einen gewissen Bewegungsraum, um beispielsweise Störungen auszuregeln. Aus diesem Grund wird eine Kostenfunktion speziell für den Anwendungsfall traversierfähiger Wasserfahrzeuge vorgestellt. Die Wahl einer Kostenfunktion richtet sich nach den gewünschten Eigenschaften, die ein Lösungspfad aufweisen soll. Die Vorstellungen des Entwicklers darüber, was optimal ist, muss mithilfe einer mathematischen Funktion formuliert werden. Ein optimaler Pfad ist letztlich nur ein Pfad, für den eine vom Entwickler entworfene Kostenfunktion ein globales Minimum annimmt. In dieser Arbeit soll ein Lösungspfad die folgenden Eigenschaften besitzen.

1. **Kürzester Pfad**
   Die Länge des Pfades entsprechend der metrischen Distanzfunktion soll so klein wie möglich sein, um das Ziel auf der kürzesten Strecke zu erreichen.

2. **Maximaler Abstand zu Hindernissen**
   Der Abstand zwischen Fahrzeug und Hindernissen soll so groß wie möglich sein, um die Gefahr einer Kollision zu minimieren.

3. **Bevorzugen der Längsbewegung**
   Entlang eines Pfades soll die Längsachse des Fahrzeuges möglichst parallel zum Pfad verlaufen, um Querbewegungen zu vermeiden, da diese typischerweise mehr Stellaufwand erfordern.

Zunächst scheinen sich die Eigenschaften zu widersprechen. Jedoch erfolgt durch die unterschiedliche Gewichtung der einzelnen Punkte ein *trade-off*. Um ein gewünschtes Verhalten zu erzielen, müssen die Parameter der Kostenfunktion vom Entwickler eingestellt werden. Die Kostenfunktion für zwei gegebene Konfigurationen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ setzt sich aus drei Termen zusammen und wird mit

$$c(\boldsymbol{q}_0, \boldsymbol{q}_1) = c_\rho(\boldsymbol{q}_0, \boldsymbol{q}_1) + c_\mu(\boldsymbol{q}_0, \boldsymbol{q}_1) + c_v(\boldsymbol{q}_0, \boldsymbol{q}_1) \tag{5.13}$$

bezeichnet. Hierbei sorgt der erste Term $c_\rho(\boldsymbol{q}_0, \boldsymbol{q}_1)$ für eine minimale Pfadlänge in $SE(2)$. Der zweite Term $c_\mu(\boldsymbol{q}_0, \boldsymbol{q}_1)$ sorgt für einen möglichst großen Abstand zu Hindernissen und der dritte Term $c_v(\boldsymbol{q}_0, \boldsymbol{q}_1)$ bevorzugt Längsbewegungen entlang der Geraden von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$. Die einzelnen Terme werden im Nachfolgenden detailliert beschrieben. Für den ersten Term wird die Distanzmetrik nach Gleichung (5.12) verwendet, sodass

$$c_\rho(\boldsymbol{q}_0, \boldsymbol{q}_1) = \rho(\boldsymbol{q}_0, \boldsymbol{q}_1) \tag{5.14}$$

gilt. Um den maximalen Abstand zu Hindernissen im zweiten Kostenterm realisieren zu können, wird ein Skalarfeld ähnlich der Potenzialfeld-Methode verwendet, in dem Hindernisse ein hohes Potenzial und somit hohe Kostenwerte aufweisen und die Kosten für freie Bereiche gegen Null gehen. Für die Herleitung des zweiten Kostenterms wird zunächst die Hilfsfunktion

$$c_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1) = \int_{\boldsymbol{\varepsilon}} f_\varepsilon ds \tag{5.15}$$

definiert. Diese beschreibt das Kurvenintegral entlang des Weges $\boldsymbol{\varepsilon}$ von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$ über dem Skalarfeld

$$f_\varepsilon(x, y) = \alpha e^{-\beta \delta^2(x,y)} \ . \tag{5.16}$$

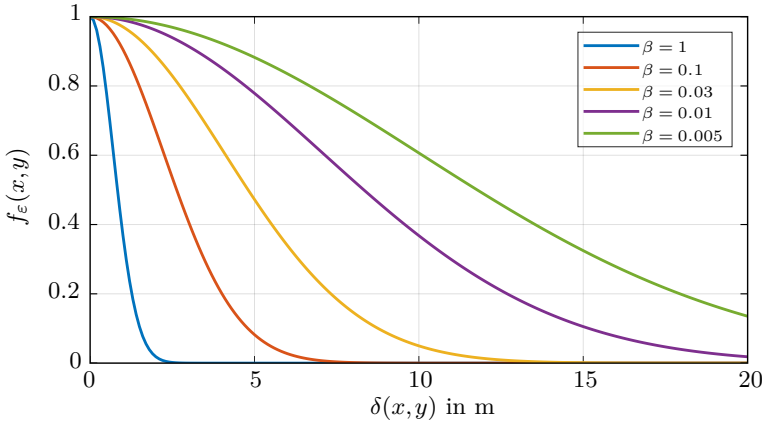


Abbildung 5.3.: Exponentialfunktion des Skalarfeldes $f_\varepsilon(x,y)$ als Funktion des geringsten Abstandes $\delta(x,y)$ für $\alpha = 1$ sowie für verschiedene Werte von β.

Hierbei ist $\delta(x,y)$ der kleinste Abstand der Position $(x,y)^\mathrm{T}$ zu allen Kanten der Polygone aus $\mathcal{O}$. Des Weiteren gilt

$$\forall (x,y) \in \mathcal{O} : \delta(x,y) = 0 \ .$$

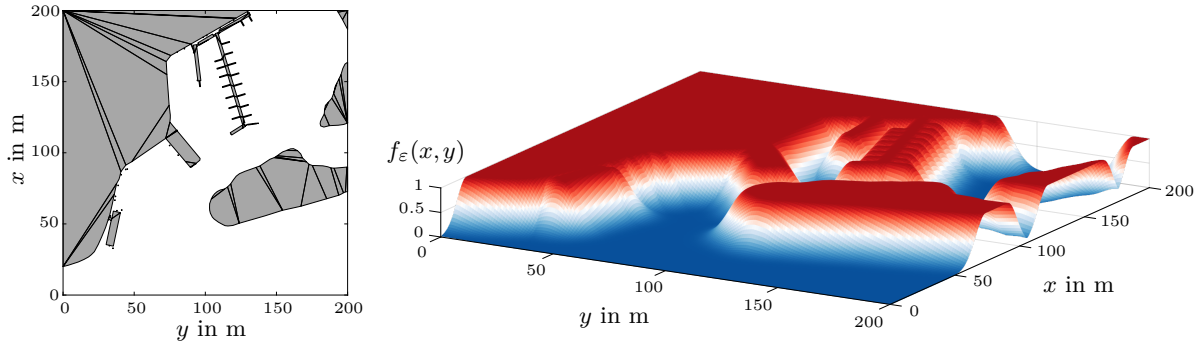
Ob sich ein Punkt innerhalb des Polygons befindet, lässt sich mit wenig Rechenaufwand mithilfe der impliziten Geradengleichung (2.5) bestimmen. Liefert (2.5) negative Funktionswerte für alle Kanten eines Polygons, dann befindet sich $(x,y)^\mathrm{T}$ innerhalb dieses Polygons. Als Basisfunktion für das Skalarfeld wird die Exponentialfunktion mit quadratischem Argument gewählt, da sich auf diese Weise ein Maximalwert für $\delta(x,y) = 0$ ergibt und der Funktionswert für $\delta(x,y) \to \infty$ gegen Null geht. Somit entsteht für das Skalarfeld eine Struktur aus Bergen und Tälern, welche durch zwei Tuningparameter $\alpha \geq 0$ und $\beta > 0$ angepasst werden kann. Die Stärke des Gefälles wird mit β gesteuert, wohingegen α den Maximalwert des Skalarfeldes festlegt und den Kostenterm gleichzeitig gegenüber den anderen beiden Kostentermen c_ρ und c_v wichtet. In der Abbildung 5.3 ist $f_\varepsilon(x,y)$ als Funktion des geringsten Abstandes $\delta(x,y)$ für $\alpha = 1$ sowie für verschiedene Werte von β dargestellt.

Ein Beispiel für ein Skalarfeld ist in Abbildung 5.4 gezeigt. In der Teilabbildung 5.4a sind die Hindernisse durch zahlreiche graue Flächen als eine Menge konvexer Polygone zu erkennen. Die Daten stammen aus einer elektronischen Navigationskarte für Binnenschifffahrtsstraßen [103] (engl. *Inland Electronic Navigational Chart* (IENC)) und beschränken sich auf ein Gebiet von $200\,\mathrm{m} \times 200\,\mathrm{m}$. In der Grafik 5.4b ist das zugehörige Skalarfeld (5.16) abgebildet. In diesem Beispiel ist $\alpha = 1$ und $\beta = 0.01$. Im Vergleich der zwei Abbildungen ist zu erkennen, dass der freie Raum geringe Funktionswerte nahe Null aufweist und dass der Funktionswert in der Umgebung von Hindernissen ansteigt und schließlich den Maximalwert $\alpha = 1$ für den durch die konvexen Polygone belegten Raum erreicht.

Um das Kurvenintegral numerisch zu lösen, wird das Skalarfeld im Voraus an zahlreichen äquidistanten Gitterpunkten berechnet und in einer *lookup table* (LUT) gespeichert. Es entsteht ein Raster mit gleichgroßen, quadratischen Zellen, die den Wert von f_ε an der Position der Gitterzelle enthalten. Die Länge der Seitenkante einer Gitterzelle wird mit g bezeichnet. Alle Zellen, die sich zwischen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ befinden, werden mit einem *ray-tracing*-Algorithmus [104] durchlaufen und aufsummiert. Das Kurvenintegral aus (5.15) wird auf diese Weise durch die Summe

$$\int_{\boldsymbol{\varepsilon}} f_\varepsilon ds \approx \sum_{\{i,j\} \in \mathbb{I}_{line}} cell(i,j) \cdot ds(i,j) \tag{5.17}$$

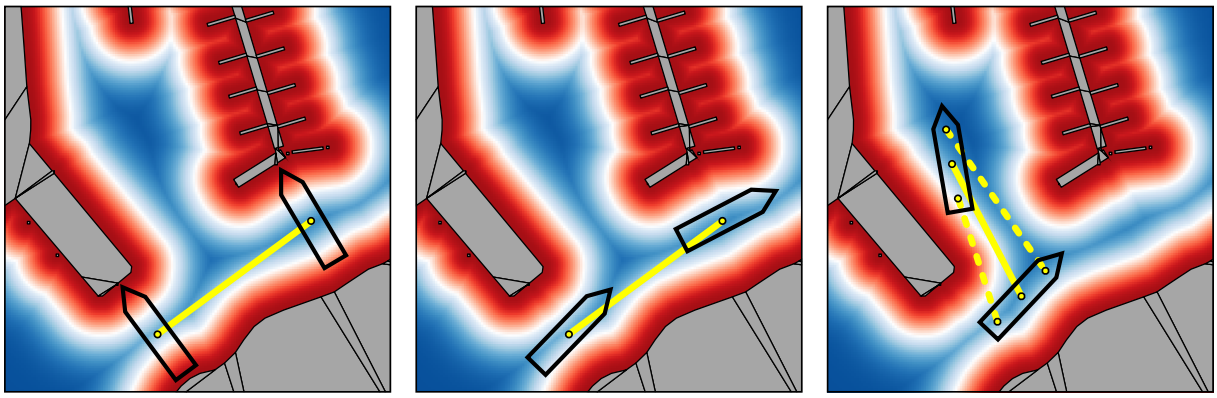
approximiert, wobei $cell(i,j)$ die Werte der LUT für die Indizes (i,j) enthalten und $ds(i,j)$ die Länge der Wegstrecke in der zugehörigen Zelle angibt. Hierbei enthält $\mathbb{I}_{line}$ die Menge aller



(a) Hindernisse aus einer elektronischen Navigationskarte, dargestellt mit 126 konvexen Polygonen.

(b) Resultierendes Skalarfeld f_ε für die Parameter $\alpha = 1$ und $\beta = 0.01$.

Abbildung 5.4.: Beispiel des Skalarfeldes auf Basis einer elektronischen Navigationskarte.



(a) Auwertung des Kurvenintegrals für den Koordinatenursprung des *b-frame*.

(b) Identische Kosten für c_ε wie in Abbildung 5.5a trotz unterschiedlicher Heading-Winkel.

(c) Auswertung an zwei weiteren körperfesten Positionen zur Berücksichtigung des Headings.

Abbildung 5.5.: Auswertung des Kurvenintegrals an mehreren körperfesten Positionen.

Indizes (i, j) zu den Zellen, die durch den *ray-tracing*-Algorithmus durchlaufen werden. Um die Rechenzeit für die Generierung der LUT zu reduzieren, kann die Berechnung von (5.16) nur an jedem m_g-ten Gitterpunkt erfolgen und die dazwischen liegenden Gitterpunkte werden durch eine bilineare Interpolation berechnet, wie in [99] beschrieben.

In (5.15) wird für die Position innerhalb des zweiten Kostenterms lediglich der Koordinatenursprung des körperfesten Koordinatensystems betrachtet. Das bedeutet auch, dass die Ausrichtung des Fahrzeuges bisher keinen Einfluss auf das Ergebnis von $c_\mu = c_\varepsilon$ hat. Da Wasserfahrzeuge jedoch typischerweise länger als breit sind, ist weniger der Abstand des *b-frame* vom Hindernis sondern vielmehr der Abstand der geometrischen Fahrzeugform zum Hindernis von Interesse. Diese Problematik ist in der Abbildung 5.5 anschaulich illustriert. In den beiden Grafiken 5.5a und 5.5b sind Anfangs- und Endposition identisch, sodass das Kurvenintegral entlang der gelben Gerade den gleichen Kostenwert liefert, obwohl die schwarz markierte geometrische Fahrzeugform in Abbildung 5.5a einen offensichtlich geringeren Abstand zu Hindernissen aufweist. Aus diesem Grund wird c_ε an mehreren Punkten im *b-frame* ausgewertet, wie in Abbildung 5.5c angedeutet ist. Für den zweiten Kostenterm aus (5.13)

ergibt sich demzufolge

$$c_\mu(\boldsymbol{q}_0, \boldsymbol{q}_1) = \frac{1}{M} \sum_{i=1}^{M} c_\varepsilon \left(\boldsymbol{q}_0 + \boldsymbol{T}_b^n(\psi_0) \begin{pmatrix} x_i^b \\ y_i^b \\ 0 \end{pmatrix},\ \boldsymbol{q}_1 + \boldsymbol{T}_b^n(\psi_1) \begin{pmatrix} x_i^b \\ y_i^b \\ 0 \end{pmatrix} \right) , \tag{5.18}$$

wobei

$$\boldsymbol{\kappa}_i^b = \begin{pmatrix} x_i^b \\ y_i^b \end{pmatrix} \tag{5.19}$$

den i-ten körperfesten Punkt für die numerische Auswertung des Kurvenintegrals beschreibt. Die maximale Anzahl von Punkten, für die der Kostenwert berechnet wird, ist mit M bezeichnet. In der Teilabbildung 5.5c ist $M = 3$. Die Wahl der Punkte ist als Teil des Tunings anzusehen. Dabei sollten sich die gewählten Punkte $\boldsymbol{\kappa}_i^b$ an der geometrischen Fahrzeugform orientieren. Gleichzeitig muss M so klein wie möglich gewählt werden, um den Rechenaufwand gering zu halten.

Mit dem dritten Term der Kostenfunktion (5.13) werden Längsbewegungen entlang des Pfades begünstigt. Aus zwei gegebenen Konfigurationen $\boldsymbol{q}_0 = (x_0, y_0, \psi_0)^\mathrm{T}$ und $\boldsymbol{q}_1 = (x_1, y_1, \psi_1)^\mathrm{T}$ lässt sich der Winkel der Geraden, welche beide Konfigurationen verbindet, mit

$$\psi_\varepsilon = \operatorname{atan2}(y_1 - y_0,\ x_1 - x_0) \tag{5.20}$$

berechnen. Der Heading-Winkel des Fahrzeuges entlang der Geraden ergibt sich aus der linearen Interpolation beider Konfigurationen und ist definiert als

$$\psi(s) = \psi_0 + \frac{s}{L} \sigma(\psi_1 - \psi_0) \quad \text{mit} \quad s \in [0, L] , \tag{5.21}$$

wobei

$$L = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{5.22}$$

die Länge der Geraden angibt. Die Winkeldifferenz

$$\Delta\psi(s) = \sigma(\psi(s) - \psi_\varepsilon) \tag{5.23}$$

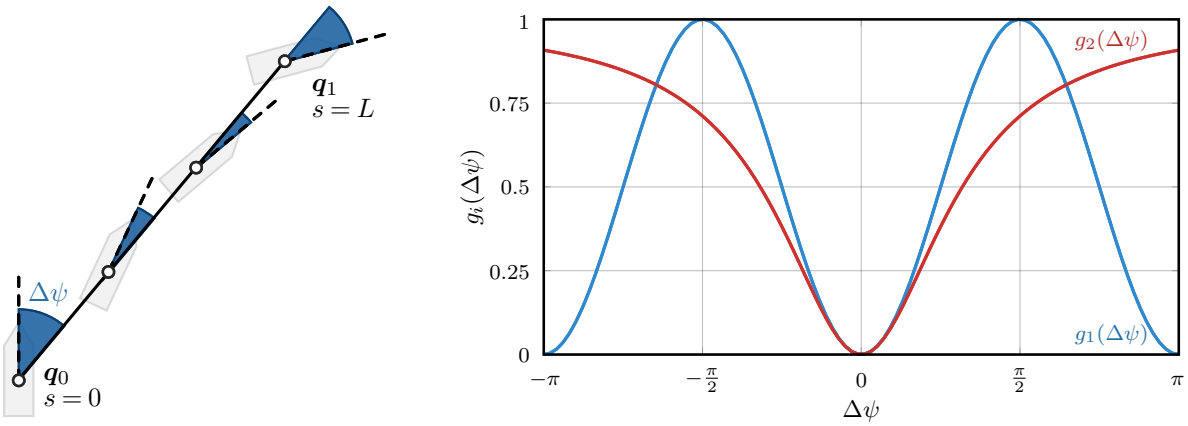
ist ein Maß für das Übereinstimmen des Heading-Winkels mit dem Geraden-Winkel entlang der Strecke von $\boldsymbol{q}_0$ nach $\boldsymbol{q}_1$. Dieser Sachverhalt ist in der Abbildung 5.6a grafisch dargestellt. Mithilfe einer Gewichtungsfunktion muss zunächst definiert werden, welche Winkeldifferenzen bestraft werden sollen. Anschließend wird c_v als das Integral dieser Gewichtungsfunktion entlang der Geraden von $s = 0$ bis $s = L$ beschrieben. Im Rahmen dieser Arbeit werden zwei Gewichtungsfunktionen vorgestellt, die je nach Fahrzeugklasse zweckmäßig sind. Für Fahrzeuge, deren Form achsensymmetrisch bezüglich der x^b- und y^b-Achse ist und bei denen kein Unterschied zwischen der Vorwärts- und Rückwärtsfahrt besteht, ist es sinnvoll, dass vorrangig Querbewegungen mit $\Delta\psi = \pm\pi/2$ zu hohen Kostenwerten führen und Längsbewegungen mit $\Delta\psi = 0$ bzw. $\Delta\psi = \pm\pi$ keine zusätzlichen Kostenwerte hervorbringen. Als Gewichtungsfunktion wird für derartige Fahrzeuge

$$g_1(\Delta\psi) = w_v \sin^2(\Delta\psi) \tag{5.24}$$

gewählt, wobei $w_v \geq 0$ ein Tuningparameter ist, mit dem der Kostenterm c_v gegenüber den Kostentermen c_ρ und c_μ gewichtet werden kann.

Die geometrische Form klassischer Wasserfahrzeuge hat hingegen keine Achsensymmetrie bezüglich der y^b-Achse, da die Rumpfform typischerweise für die Vorwärtsbewegung optimiert ist. Zudem kann die Manövrierfähigkeit beim Rückwärtsfahren je nach Antriebssystem in Kombination mit der Rumpfform stark eingeschränkt sein. Aus diesem Grund wird für derartige Fahrzeugklassen die Gewichtungsfunktion

$$g_2(\Delta\psi) = w_\alpha \frac{(w_\beta \Delta\psi)^2}{1 + (w_\beta \Delta\psi)^2} \tag{5.25}$$



(a) Schematische Darstellung der Winkeldifferenz $\Delta\psi$ entlang einer Geraden von $\boldsymbol{q}_0$ nach $\boldsymbol{q}_1$.

(b) Gewichtungsfunktionen $g_i(\Delta\psi)$ für die Bestrafung von vorrangig Querbewegungen (blau) sowie von Quer- und Rückwärtsbewegungen (rot).

Abbildung 5.6.: Illustration der Winkeldifferenz zwischen Fahrzeug und Pfad (links) sowie unterschiedlicher Gewichtungsfunktionen dieser Differenz für c_v (rechts).

verwendet, die ausschließlich die Längsbewegung begünstigt und Quer- sowie Rückwärtsbewegungen bestraft. Hierbei sind $w_\alpha \geq 0$ und $w_\beta > 0$ zwei Tuningparameter. Mit w_α wird der Kostenterm c_v gegenüber den Kostentermen c_ρ und c_μ gewichtet und mit w_β wird die Gewichtungsfunktion gestaucht bzw. gestreckt.

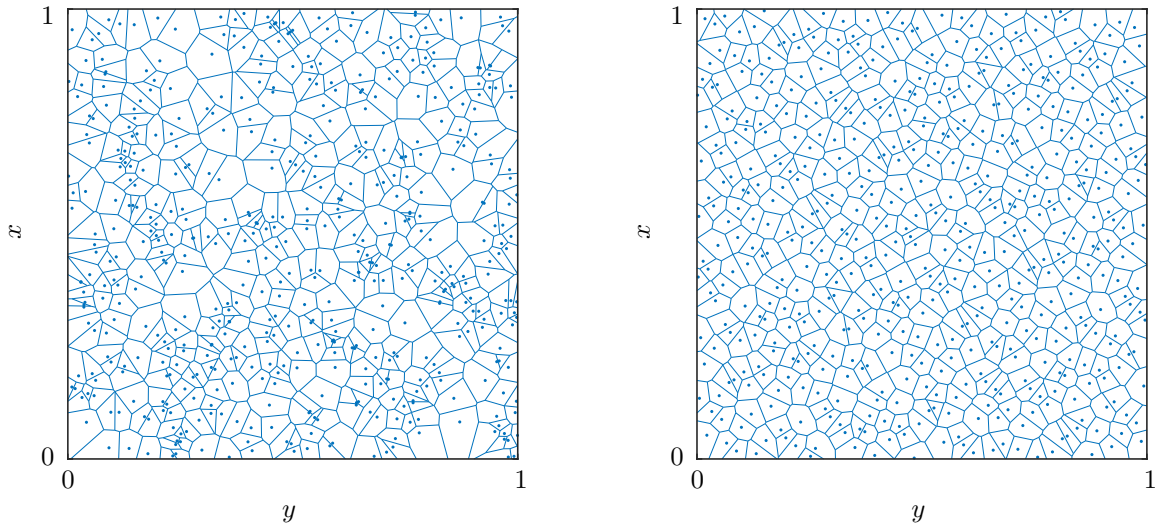
In der Abbildung 5.6b sind die zwei Gewichtungsfunktionen aus (5.24) und (5.25) für $w_v = w_\alpha = w_\beta = 1$ gegenübergestellt. Es ist zu erkennen, dass beide ein Minimum für $\Delta\psi = 0$ besitzen, um die Vorwärtsbewegung zu begünstigen. Während mit $g_1(\Delta\psi)$ lediglich Querbewegungen bestraft werden und die Rückwärtsbewegung in gleicher Weise wie die Vorwärtsbewegung begünstigt wird, werden mit $g_2(\Delta\psi)$ zunehmend große Winkeldifferenzen bestraft. Der Kostenterm c_v wird als das Integral der Gewichtungsfunktionen über die Wegstrecke von $\boldsymbol{q}_0$ nach $\boldsymbol{q}_1$ definiert und ist gegeben mit

$$c_v(\boldsymbol{q}_0, \boldsymbol{q}_1) = \begin{cases} 0 & \text{falls } L = 0 \\ \underbrace{\int_{s=0}^{L} g_1\left(\Delta\psi(s)\right) ds}_{c_{v,1}} + \underbrace{\int_{s=0}^{L} g_2\left(\Delta\psi(s)\right) ds}_{c_{v,2}} & \text{sonst.} \end{cases} \tag{5.26}$$

Durch die Summe der zwei Terme $c_{v,1}$ und $c_{v,2}$ ist ebenfalls eine Kombination der beiden Gewichtungsfunktionen aus (5.24) und (5.25) möglich. Für den Fall $L = 0$ ist der resultierende Kostenwert mit $c_v(\boldsymbol{q}_0, \boldsymbol{q}_1) = 0$ definiert. Dieser Fall entspricht dem Drehen auf der Stelle, bei dem kein Weg zurückgelegt wird. Die bestimmten Integrale $c_{v,1}$ und $c_{v,2}$ aus (5.26) lassen sich analytisch lösen, wobei zur korrekten Behandlung von Winkeldifferenzen mithilfe der $\sigma()$-Funktion nach (5.9) zwischen mehreren Fällen unterschieden werden muss. Eine detaillierte Berechnungsvorschrift für die Ausdrücke $c_{v,1}$ und $c_{v,2}$ ist in Appendix A.2 zu finden.

5.3.3. Sampling

Die Aufgabe des Sampling ist es, den Konfigurationsraum mit einer ausreichend hohen Auflösung abzutasten. Da die Samples zufällig generiert werden und $\mathcal{C}$ unzählbar und unendlich ist, kann mit einem sampling-basierten Verfahren keine Aussage darüber getroffen werden, ob eine Lösung existiert oder nicht. Aus diesem Grund ist sind sampling-basierte Algorithmen nicht vollständig. Jedoch geht die Wahrscheinlichkeit dafür, dass eine Lösung gefunden wird, sofern sie existiert, gegen 1, wenn die Anzahl zufälliger Samples gegen Unendlich geht [15]. Diese Eigenschaft wird als probabilistische Vollständigkeit bezeichnet.



(a) Sampling auf Basis der *rand*-Funktion von MATLAB®.

(b) Sampling auf Basis der Halton-Sequenz.

Abbildung 5.7.: Voronoi-Diagramm für 500 gleichmäßig verteilte 2D-Samples bei unterschiedlichen Pseudozufallszahlengeneratoren.

In der Praxis ist die Anzahl an Samples aus rechentechnischen Gründen beschränkt. Um dennoch eine ausreichende Verteilung von Samples im Konfigurationsraum zu erzeugen, eignen sich spezielle Generatoren für Pseudozufallszahlen, welche eine geringe Diskrepanz aufweisen. Die Diskrepanz ist hierbei ein Maß für die ungleichmäßige Verteilung. Soll beispielsweise der Wertebereich $(0,1)$ gleichmäßig mit N Samples abgetastet werden, dann wäre der optimale Abstand zwischen den Samples $1/(N+1)$.

Für das Sampling innerhalb des vorgestellten Pfadplaners kommt die Halton-Sequenz [105] zum Einsatz, welche multidimensionale Zufallszahlen mit geringer Diskrepanz generiert. Bei diesem deterministischen Pseudozufallszahlengenerator werden neue Samples immer in den größten freien Bereich gelegt. Die Intervalle zwischen den Samples werden fortlaufend unterteilt, wodurch der Raum immer feiner abgetastet wird. Dabei ist der Faktor für die Unterteilung in jeder Dimension verschieden und basiert auf Primzahlen. Für den Wertebereich $(0,1)$ und ein Teilungsverhältnis mit der Primzahl $p=2$ ergeben sich beispielsweise die folgenden Werte für eine Sampling-Sequenz.

$$\left\{\frac{1}{2},\frac{1}{4},\frac{3}{4},\frac{1}{8},\frac{5}{8},\frac{3}{8},\frac{7}{8},\dots\right\}$$

Für die nächste Dimension wird die nächste Primzahl $p=3$ als Teilungsverhältnis gewählt, was zu der Sampling-Sequenz

$$\left\{\frac{1}{3},\frac{2}{3},\frac{1}{9},\frac{4}{9},\frac{7}{9},\frac{2}{9},\frac{5}{9},\frac{8}{9},\frac{1}{27},\dots\right\}$$

führt. Die Sampling-Sequenzen lassen sich mithilfe der in [106] beschriebenen Methode berechnen. Für wenige Dimensionen, unter 10, können mithilfe der Halton-Sequenz Pseudozufallszahlen mit geringer Diskrepanz erzeugt werden [15]. In Abbildung 5.7 ist das Ergebnis zweier Pseudozufallszahlengeneratoren für die Erzeugung zweidimensionaler Samples dargestellt. Die Grafiken zeigen jeweils das Voronoi-Diagramm. Hierbei stellen die Punkte die tatsächlichen Samples dar. Jeder Sample befindet sich in einer Zelle, welche die Menge aller Punkte charakterisiert, die diesem Sample am dichtesten sind. Die Abbildung 5.7a zeigt das Ergebnis von 500 Samples, die mithilfe der *rand*-Funktion von MATLAB® generiert wurden. Im Vergleich

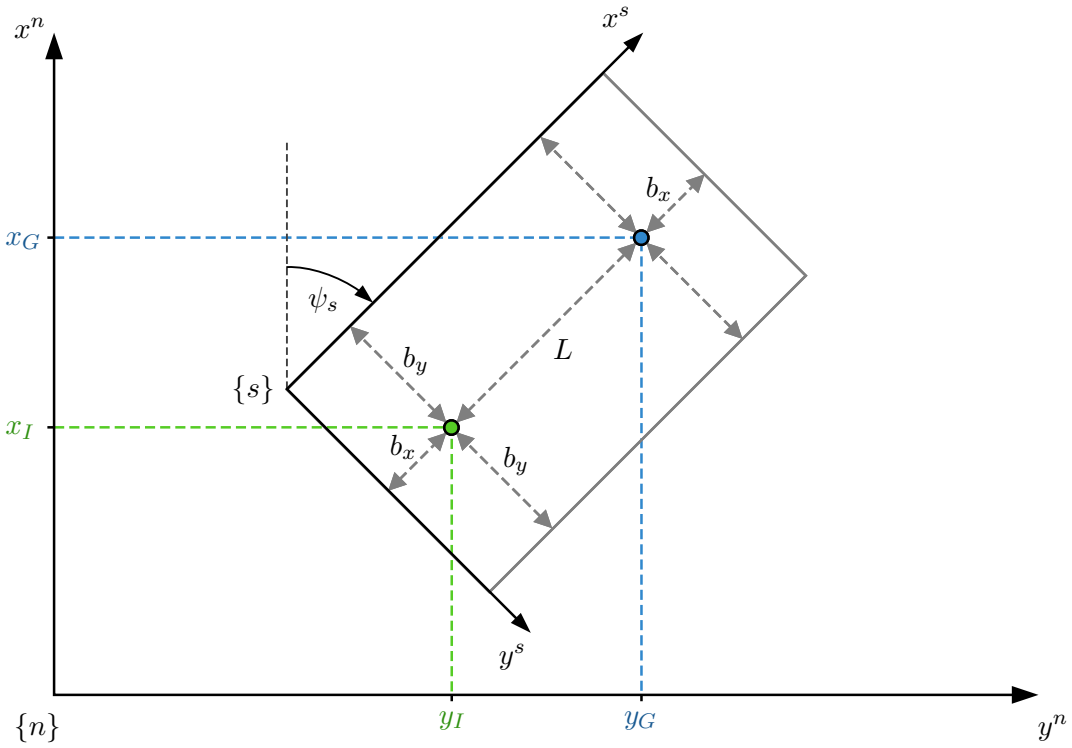


Abbildung 5.8.: Beschreibung des Sampling-Gebietes durch einen orientierten Quader.

dazu ist in Abbildung 5.7b das Ergebnis bei Verwendung der Halton-Sequenz dargestellt. Bei der *rand*-Funktion von MATLAB® handelt es sich um einen Pseudozufallszahlengenerator der gleichmäßige Zufallszahlen im Bereich $(0,1)$ auf Basis des Mersenne-Twister-Generators erzeugt [107]. Es ist zu erkennen, dass die Samples der Halton-Sequenz gleichmäßiger im Raum verteilt sind, während es bei der *rand*-Funktion Bereiche mit vielen oder wenigen Punkten gibt.

Der Konfigurationsraum ist unendlich groß und muss daher für das Sampling beschränkt werden. Das hat zur Folge, dass nur Lösungen in diesem beschränkten Gebiet gefunden werden können. Für die Pfadplanung wird ein orientierter Quader (engl. *Oriented Bounding Box* (OBB)) verwendet, wie in Abbildung 5.8 dargestellt ist. Die gegebenen Konfigurationen $\boldsymbol{q}_I = (x_I, y_I, \psi_I)^{\mathrm{T}}$ und $\boldsymbol{q}_G = (x_G, y_G, \psi_G)^{\mathrm{T}}$ befinden sich innerhalb der OBB und legen die Richtung der x^s-Achse für das Samplingkoordinatensystem $\{s\}$ fest. Ausgehend von diesen beiden Konfigurationen wird die Größe der OBB mithilfe der Parameter $b_x > 0$ und $b_y > 0$ bestimmt. Die Ausrichtung ist mit

$$\psi_s = \operatorname{atan2}\left(y_G - y_I,\ x_G - x_I\right) \tag{5.27}$$

gegeben. Falls die Positionen beider Konfigurationen identisch sind, wird $\psi_s = 0$ gesetzt. Ein zufälliger Sample für die k-te Iteration wird mit

$$\boldsymbol{q}_k = \begin{pmatrix} x_I \\ y_I \\ 0 \end{pmatrix} + \boldsymbol{T}_s^n(\psi_s) \left[\begin{pmatrix} -b_x \\ -b_y \\ 0 \end{pmatrix} + \begin{pmatrix} (L + 2b_x)\, H(k,2) \\ 2b_y H(k,3) \\ \sigma\left(2\pi H(k,5)\right) \end{pmatrix} \right] \tag{5.28}$$

berechnet, wobei

$$L = \sqrt{(x_G - x_I)^2 + (y_G - y_I)^2} \tag{5.29}$$

der Abstand zwischen Start- und Zielposition ist und $\boldsymbol{T}_s^n(\psi_s)$ die Transformationsmatrix beschreibt, mit deren Hilfe Größen aus dem Samplingkoordinatensystem in das Navigationskoordinatensystem transformiert werden. Die Komponenten dieser Transformationsmatrix sind

dabei identisch zu denen aus (5.5). Die Funktion $H(k,p) \in (0,1)$ entspricht der Halton-Sequenz für den k-ten Sample mit dem Teilungsverhältnis der Primzahl p.

Es gibt viele zusätzliche Heuristiken, die in Verfahren aus der Literatur für das Sampling zum Einsatz kommen, um die Konvergenz eines Planungsproblems zu verbessern. Beispiele dafür sind das *informed sampling* [44] oder *goal-oriented sampling* [108, 109]. Für die Pfadplanung wird das *goal sampling* verwendet. Bei dieser Heuristik wird die Zielkonfiguration $\boldsymbol{q}_G$ periodisch anstelle einer Zufallskonfiguration gewählt. Auf diese Weise kann das Ziel exakt erreicht werden, sofern die *Steer*-Funktion zwei Konfigurationen exakt verbinden kann.

Die *Sample*-Prozedur für die Pfadplanung ist in Algorithmus 7 beschrieben. Die Variablen $k \in \mathbb{N}_0$ und $g \in \mathbb{N}_0$ stellen Iterationsvariablen dar und werden zu Beginn auf null gesetzt. In Zeile 2 wird untersucht, ob ein zufälliger Sample aus dem orientierten Quader gezogen werden soll oder ob das *goal sampling* angewendet werden soll. In Zeile 3 wird der k-te Sample nach (5.28) berechnet. Die Pseudozufallszahlen der Halton-Sequenz können im Vorfeld berechnet und in einer LUT gespeichert werden. Die maximale Anzahl dieser gespeicherten Samples ist mit k_{max} gegeben. Die Iterationsvariable k wird in Zeile 4 fortlaufend inkrementiert und bleibt durch die Modulo-Operation im Bereich $[0, k_{max})$. Falls der Zielknoten noch nicht Teil des Baums und $g = 0$ ist, wird die Zielkonfiguration als zufälliger Sample gewählt (Zeile 6). Die Iterationsvariable für das *goal sampling* wird anschließend in Zeile 8 fortlaufend inkrementiert und durch die Modulo-Operation auf den Bereich $[0, g_{max})$ abgebildet. Dabei ist $g_{max} > 0$ ein Tuningparameter, mit dem die Periode für das *goal sampling* festgelegt wird. Außerhalb der Sampling-Prozedur wird am Ende einer Iteration $g = 0$ gesetzt, falls das *goal sampling* durchgeführt wurde und die *Steer*-Funktion den Schritt zum Ziel beschränkt hat und dieser Schritt kollisionsfrei war. Mit anderen Worten: Wurde ein neuer Knoten dem Baum hinzugefügt, der durch *goal sampling* entstanden ist, jedoch nicht das Ziel erreicht hat, dann wird das *goal sampling* fortgesetzt, solange, bis das Ziel erreicht wird oder der Knoten aufgrund einer Kollision nicht dem Baum hinzugefügt werden kann. Dadurch ist es möglich, triviale Lösungen schnell zu finden, ohne zahlreiche zufällige Samples generieren zu müssen.

Algorithmus 7: *Sample*-Prozedur der Pfadplanung

```
   Data: k ← 0; g ← 0
1: procedure q_rand ← Sample(T, q_G)
2:     if (q_G ∈ T) ∨ (g ≠ 0) then
3:         q_rand ← OrientedBoxSampling(k);        // sampling according to (5.28)
4:         k ← (k + 1) mod k_max;
5:     else
6:         q_rand ← q_G;                                          // goal sampling
7:     end
8:     g ← (g + 1) mod g_max;
9: end
```

5.3.4. Steer-Funktion

Mithilfe der *Steer*-Funktion in Algorithmus 6 wird der Baum in Richtung eines zufälligen Samples $\boldsymbol{q}_{rand}$ erweitert. In Algorithmus 8 ist die *Steer*-Prozedur dargestellt, mit der sich zwei Konfigurationen direkt durch eine Gerade in $\mathcal{C}$ unter Beachtung der topologischen Eigenschaften des Raums verbinden lassen. Ist der Abstand zur gesampelten Konfiguration jedoch zu groß, wird dieser auf einen optimalen Abstand beschränkt, um die Konvergenz des Verfahrens zu verbessern. Das Ergebnis der *Steer*-Funktion ist in diesem Fall eine neue Konfiguration $\boldsymbol{q}_{new}$.

Algorithmus 8: *Steer*-Prozedur der Pfadplanung

1: **procedure** $(\boldsymbol{q}_{new}, \boldsymbol{\varepsilon}) \leftarrow \text{Steer}(\mathcal{T}, \boldsymbol{q}_{near}, \boldsymbol{q}_{rand})$
2: $\quad \Delta\boldsymbol{q} \leftarrow \text{PoseDifference}(\boldsymbol{q}_{near}, \boldsymbol{q}_{rand})$; // according to (5.30)
3: $\quad s \leftarrow \|\Gamma(\Delta\boldsymbol{q})\|_2$;
4: $\quad \lambda_s \leftarrow \text{OptimalStepsize}(\text{card}(\mathcal{T}))$; // according to (5.35)
5: $\quad$ **if** $s > \lambda_s$ **then**
6: $\quad\quad \Delta\boldsymbol{q} \leftarrow \frac{\lambda_s}{s}\Delta\boldsymbol{q}$;
7: $\quad$ **end**
8: $\quad \boldsymbol{q}_{new} \leftarrow \boldsymbol{\sigma}(\boldsymbol{q}_{near} + \Delta\boldsymbol{q})$;
9: $\quad \text{Cost}(\boldsymbol{q}_{new}) = \text{Cost}(\boldsymbol{q}_{near}) + c(\boldsymbol{q}_{near}, \boldsymbol{q}_{new})$;
10: $\quad \boldsymbol{\varepsilon} \leftarrow \text{Edge}(\boldsymbol{q}_{near}, \boldsymbol{q}_{new})$;
11: **end**

Zu Beginn wird die Posen-Differenz

$$\Delta\boldsymbol{q} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\psi \end{pmatrix} = \boldsymbol{\sigma}(\boldsymbol{q}_1 - \boldsymbol{q}_0) \tag{5.30}$$

berechnet (Zeile 2). Die Länge dieser Differenz wird als Schrittweite bezeichnet und im RRT*-Algorithmus aus den in Abschnitt 3.1.2 bereits erwähnten Gründen beschränkt. Dafür wird ein zum Konfigurationsraum äquivalenter euklidischer Raum definiert, in welchem die metrische Distanzfunktion ρ zweier Konfigurationen genau dem euklidischen Abstand dieser beiden Konfigurationen entspricht. Die Funktion $\Gamma : \mathcal{C} \mapsto \mathbb{R}^3$ bildet den Konfigurationsraum auf einen äquivalenten euklidischen Raum ab und ist definiert als

$$\Gamma(\boldsymbol{q}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & w_\psi \end{pmatrix} \boldsymbol{q} \tag{5.31}$$

Die Schrittweite wird in Zeile 3 mit

$$s = \|\Gamma(\Delta\boldsymbol{q})\|_2 = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (w_\psi \Delta\psi)^2} \tag{5.32}$$

berechnet. Die optimale Schrittweite λ_s hängt von der Kardinalität des Baums ab und ist mit (3.2) gegeben. Für den Parameter γ wird

$$\gamma = 2^3 \left(1 + \frac{1}{3}\right) \mu(\mathcal{C}) \tag{5.33}$$

gewählt, wobei

$$\mu(\mathcal{C}) = (2b_x + L) \cdot (2b_y) \cdot (2w_\psi \pi) \tag{5.34}$$

dem Volumen des Sampling-Gebietes aus Abschnitt 5.3.3 im euklidischen Raum entspricht. Die Wahl von $\mu(\mathcal{C})$ anstelle von $\mu(\mathcal{C}_{free})$ in (5.33) im Vergleich zu (3.3) sorgt dafür, dass die Ungleichung (3.3) erfüllt ist, da in der Praxis $\mu(\mathcal{C}_{free})$ aufgrund von Hindernissen typischerweise kleiner ist als das Volumen des Sampling-Gebietes. Gibt es keine Hindernisse, würde eine Schrittweite gewählt werden, die nicht der nach (3.2) entspricht. Jedoch kann im RRT*-Algorithmus mit jeder denkbaren Schrittweite gerechnet werden. Unter Umständen führt dies nicht zur bestmöglichen Konvergenz im zeitlichen Sinne, was in diesem Fall eine untergeordnete Rolle spielt, zumal durch die Abwesenheit von Hindernissen ohnehin wesentlich mehr Iterationen bei gleicher Rechenzeit durchgeführt werden können. Aus (3.2), (5.33) und (5.34) resultiert für die optimale Schrittweite in Zeile 4

$$\lambda_s = \min\left\{ \left(32(L + 2b_x) b_y w_\psi \frac{\log n}{n}\right)^{\frac{1}{3}},\ \lambda_{max} \right\}, \tag{5.35}$$

wobei $n = \max(3, \mathrm{card}(\mathcal{T}))$ die Kardinalität des Baums $\mathcal{T}$ kennzeichnet, die nach unten auf 3 beschränkt wird, und λ_{max} ein optionaler Parameter ist, mit dem ein oberes Limit für die Schrittweite definiert werden kann. Die Beschränkung von n ist notwendig, da $\log n / n$ für $n = 1$ null wäre. Das Maximum dieses Bruches liegt bei $n = e$, wobei $e \approx 2.718$ die eulersche Zahl ist. Ist die Schrittweite größer als die optimale Schrittweite (Zeile 5), dann wird die Posendifferenz $\Delta \boldsymbol{q}$ beschränkt (Zeile 6). Anschließend wird in Zeile 8 die neue Konfiguration $\boldsymbol{q}_{new}$ berechnet, indem die Posendifferenz zur gegebenen Konfiguration $\boldsymbol{q}_{near}$ addiert wird. In Zeile 9 wird der Kostenwert für $\boldsymbol{q}_{new}$ berechnet. Dieser setzt sich aus dem Kostenwert für den Knoten $\boldsymbol{q}_{near}$ und den Verbindungskosten (5.13) zwischen beiden Knoten $\boldsymbol{q}_{near}$ und $\boldsymbol{q}_{new}$ zusammen. Die Verbindung beider Knoten bildet die Kante ε (Zeile 10), welche im weiteren Verlauf der Pfadplanung für den Kollisionstest benötigt wird.

5.3.5. IsFeasible-Funktion

Mithilfe der *IsFeasible*-Prozedur wird innerhalb der Pfadplanung getestet, ob sich alle Konfigurationen entlang einer Kante ε im freien Konfigurationsraum $\mathcal{C}_{free}$ befinden. Eine Kante entspricht der Geraden zwischen zwei Konfigurationen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ und ist definiert mit

$$\varepsilon: \ \boldsymbol{q}(s) = \boldsymbol{q}_0 + s \cdot \boldsymbol{\sigma}(\boldsymbol{q}_1 - \boldsymbol{q}_0) \ , \ \ \forall s \in [0,1] \ . \tag{5.36}$$

Die *IsFeasible*-Prozedur der Pfadplanung ist in Algorithmus 9 gezeigt und besteht im Wesentlichen aus einem Kollisionstest. In den Zeilen 2 und 3 werden die Anfangs- und Endkonfiguratio-

Algorithmus 9: *IsFeasible*-Prozedur der Pfadplanung

```
1: procedure feasible ← IsFeasible(ε, O)
2:     q0 ← ε(0);
3:     q1 ← ε(1);
4:     if q0 = q1 then
5:         return FALSE;
6:     end
7:     return ¬CheckCollision(q1, q0, O);          // see algorithm 10
8: end
```

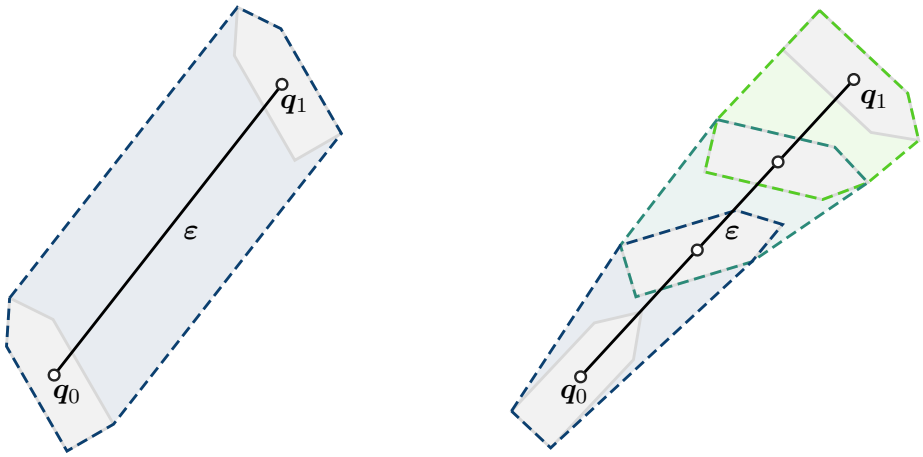
nen der Kante ermittelt. Falls $\boldsymbol{q}_0 = \boldsymbol{q}_1$ ist, gilt die Kante als ungültig (Zeile 4-5). Dadurch wird vermieden, dass $\mathcal{T}$ duplizierte Knoten mit identischen Posen enthalten kann. Ansonsten ist eine Kante gültig, falls es nicht zur Kollision des Fahrzeugpolygons mit Hindernispolygonen entlang dieser Kante kommt (Zeile 7). Der Kollisionstest erfolgt dabei ausgehend von der Endkonfiguration $\boldsymbol{q}_1$ zur Anfangskonfiguration $\boldsymbol{q}_0$, da die Wahrscheinlichkeit von $\mathcal{V}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset$ für $\boldsymbol{q} = \boldsymbol{q}_1$ höher ist, als für $\boldsymbol{q} = \boldsymbol{q}_0$, wodurch die Rechenzeit des Kollisionstests reduziert werden kann. Sowohl das Fahrzeugpolygon $\mathcal{V}(\boldsymbol{q})$ als auch die Hindernispolygone $\mathcal{O}$ werden durch eine Menge konvexer Polygone dargestellt. Im Folgenden wird jedoch davon ausgegangen, dass das Fahrzeugpolygon aus einem einzigen konvexen Polygon besteht. Die Vereinigung aller Fahrzeugpolygone für alle Konfigurationen $\boldsymbol{q}$ entlang der Kante ε von $\boldsymbol{q}_0$ nach $\boldsymbol{q}_1$ wird mit

$$\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1) = \bigcup_{\boldsymbol{q} \in \varepsilon} \mathcal{V}(\boldsymbol{q}) \tag{5.37}$$

bezeichnet und entspricht somit der Vereinigung unendlich vieler konvexer Polygone. Eine Kante ist kollisionsfrei, falls

$$\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1) \cap \mathcal{O} = \emptyset \tag{5.38}$$

gilt. Die Schwierigkeit liegt hierbei in der geometrischen Beschreibung von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$. Für die Vereinigung unendlich vieler Polygone entlang einer Geraden nach (5.37) müssen in der Praxis



(a) Exakte Beschreibung durch eine konvexe Hülle für den Fall $\psi_0 = \psi_1$.

(b) Unterteilung von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ in mehrere konvexe Hüllen für den Fall $\psi_0 \neq \psi_1$.

Abbildung 5.9.: Approximation von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch ein oder mehrere konvexe Polygone.

Näherungen verwendet werden. In [15] wird vorgeschlagen, mehreren Konfigurationen $\boldsymbol{q} \in \boldsymbol{\varepsilon}$ zu sampeln, solange bis eine gewünschte Genauigkeit für die Approximation von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ erreicht ist. Im Rahmen dieser Arbeit wird ein Verfahren für den Kollisionstest eines Polygons entlang einer Geraden vorgestellt, bei dem auf ein derartiges Sampling verzichtet wird. In dem entwickelten Verfahren wird $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch ein oder mehrere konvexe Polygone approximiert. Dabei wird zwischen zwei Fällen unterschieden. Ist $\psi_0 = \psi_1$, dann bleibt der Heading-Winkel entlang der Kante konstant und die resultierende geometrische Form lässt sich exakt durch eine konvexe Hülle aller Vertices der zwei konvexen Polygone $\mathcal{V}(\boldsymbol{q}_0)$ und $\mathcal{V}(\boldsymbol{q}_1)$ beschreiben, wie Abbildung 5.9a illustriert. Ist hingegen $\psi_0 \neq \psi_1$, dann wird $\boldsymbol{\varepsilon}$ in mehrere Teilgeraden zerlegt, sodass die Betragsdifferenz des Heading-Winkels für eine Teilgerade unter einem vorgegebenen Schwellwert $\Delta\psi_{max}$ liegt, wie in Abbildung 5.9b dargestellt ist. Die Approximation von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch eine endliche Anzahl konvexer Polygone wird mit $\mathcal{V}_\varepsilon^\Diamond(\boldsymbol{q}_0, \boldsymbol{q}_1)$ bezeichnet.

Um die Rechenzeit für den Kollisionstest weiter zu reduzieren, werden an den Achsen ausgerichtete Quader (engl. *Axis-Aligned Bounding Boxes* (AABB)) als Hüllkörper für konvexe Polygone verwendet, wie in Abbildung 5.10 illustriert. Bevor untersucht wird, ob sich zwei konvexe Polygone überschneiden, wird getestet, ob sich deren AABBs überschneiden. Ist dies nicht der Fall, kann eine Überlagerung der zugehörigen konvexen Polygone ausgeschlossen werden, da Polygone eine Untermenge von AABBs bilden. Der Überlagerungstest zweier AABBs lässt sich rechentechnisch sehr schnell auswerten.

Die entwickelte *CheckCollision*-Prozedur ist in Algorithmus 10 zusammengefasst. Zu Beginn wird geprüft, ob es zur Überschneidung der AABB von $\mathcal{V}_\varepsilon^\Diamond(\boldsymbol{q}_0, \boldsymbol{q}_1)$ mit einer AABB aus $\mathcal{O}$ kommt (Zeile 2). Für die Bestimmung der AABB von $\mathcal{V}_\varepsilon^\Diamond(\boldsymbol{q}_0, \boldsymbol{q}_1)$ wird der größte Abstand R eines Vertex des Fahrzeugpolygons $\mathcal{V}(\boldsymbol{q})$ zum Koordinatenursprung des *b-frame* verwendet. Dieser Abstand entspricht dem Radius des Kreises, in dem sich alle Vertices des Fahrzeugpolygons befinden. Die oberen und unteren Grenzen der AABB ergeben sich dann mit

$$
\begin{aligned}
x_{min} &= min(\{x_0 - R,\ x_0 + R,\ x_1 - R,\ x_1 + R\}) \\
x_{max} &= max(\{x_0 - R,\ x_0 + R,\ x_1 - R,\ x_1 + R\}) \\
y_{min} &= min(\{y_0 - R,\ y_0 + R,\ y_1 - R,\ y_1 + R\}) \\
y_{max} &= max(\{y_0 - R,\ y_0 + R,\ y_1 - R,\ y_1 + R\})\ .
\end{aligned}
$$

Kommt es zur Überschneidung der AABBs besteht die Möglichkeit einer Kollision.

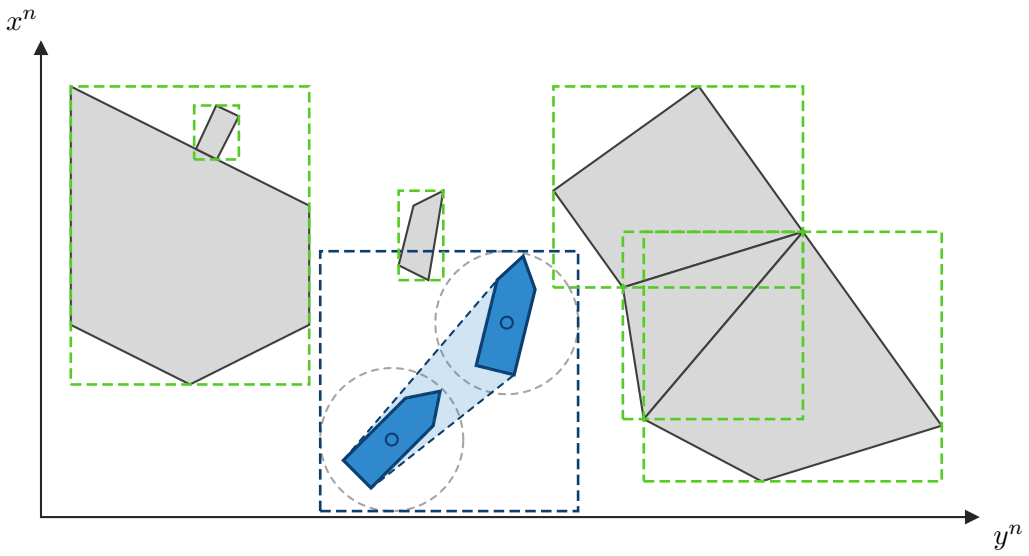
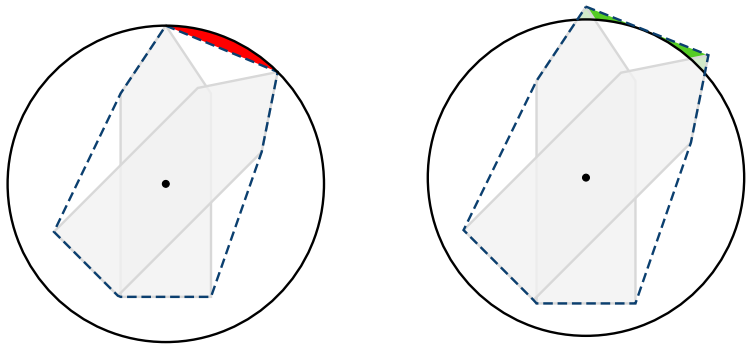


Abbildung 5.10.: Schematische Darstellung an den Achsen ausgerichteter Quader als Hüllkörper für konvexe Polygone.

Algorithmus 10: *CheckCollision*-Prozedur

```
1:  procedure collision ← CheckCollision(q_0, q_1, O)
2:      if AABB(O) ∩ AABB(V_ε^◇(q_0, q_1)) ≠ ∅ then
3:          Δq ← σ(q_1 − q_0);
4:          N ← max(1, ⌈|Δψ/Δψ_max|⌉);
5:          q_a ← q_0;
6:          for i = 1 to N do
7:              q_b ← σ(q_a + (1/N)Δq);
8:              if AABB(O) ∩ AABB(V_ε^◇(q_a, q_b)) ≠ ∅ then
9:                  P_i ← Quickhull(V(q_a), V(q_b));
10:                 if O ∩ P_i then
11:                     return TRUE;
12:                 end
13:             end
14:             q_a ← q_b;
15:         end
16:     end
17:     return FALSE;
18: end
```

In Zeile 3 wird die Differenz beider Konfigurationen berechnet und in Zeile 4 die Anzahl der Unterteilungen N bestimmt, wobei $\Delta\psi_{max} \in (0, \pi)$ einen Schwellwert für die betragsmäßig maximale Änderung des Heading-Winkels entlang einer Teilgerade kennzeichnet. Gleichzeitig legt dieser Parameter fest, wie gut die Approximation von $\mathcal{V}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch mehrere konvexe Polygone für den Fall $\psi_0 \neq \psi_1$ sein soll. Dabei ist zu beachten, dass durch die Approximation Bereiche ignoriert werden, die jedoch zur Kollision führen könnten. Die Abbildung 5.11a zeigt die konvexe Hülle für das Drehen auf der Stelle. Es ist zu erkennen, dass der rot markierte Bereich als kollisionsfrei angesehen werden würde, obwohl sich die Bugspitze des Schiffes beim Drehen auf dem Kreis bewegt. Um diesem Problem zu begegnen, muss innerhalb der Pfadplanung ein größeres Fahrzeugpolygon verwendet werden, damit ein sicherer Kollisionstest bei einer gewählten Approximation durchgeführt wird, wie in Abbildung 5.11b illustriert ist. Eine detaillierte Beschreibung



(a) Missachteter Bereich (rot) beim Kollisionstest aufgrund der Approximation.

(b) Vergrößerung der geometrischen Fahrzeugform für einen sicheren Kollisionstest.

Abbildung 5.11.: Schematische Darstellung der Approximation von $\mathcal{V}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch ein konvexes Polygon (gestrichelte Linie) beim Drehen auf der Stelle.

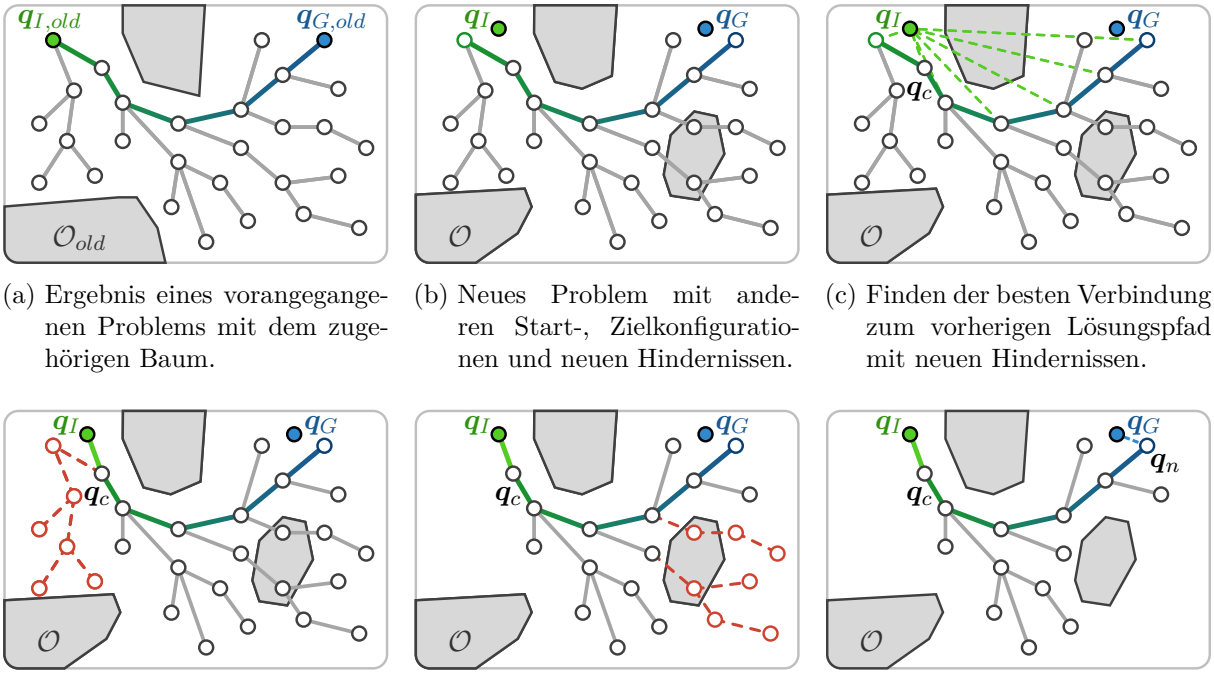
für die Wahl eines größeren Fahrzeugpolygons ist in Appendix A.3 zu finden. In Zeile 6 erfolgt die Iteration über alle N Teilgeraden. Die Anfangs- und Endkonfigurationen jeder Teilstrecke sind mit $\boldsymbol{q}_a$ und $\boldsymbol{q}_b$ bezeichnet und werden in jedem Iterationsschritt angepasst (Zeile 7 und 14). In Zeile 8 wird erneut geprüft, ob es zur Überschneidung des Hüllkörpers der Teilgeraden mit denen der Hindernisse kommt. Besteht die Möglichkeit einer Kollision, wird in Zeile 9 die konvexe Hülle der beiden Fahrzeugpolygone $\mathcal{V}(\boldsymbol{q}_a)$ und $\mathcal{V}(\boldsymbol{q}_b)$ berechnet, wie in Abbildung 5.9a veranschaulicht. Die Berechnung der konvexen Hülle erfolgt mithilfe des Quickhull-Algorithmus [110]. Falls sich die konvexe Hülle $\mathcal{P}_i$ mit anderen Polygonen aus $\mathcal{O}$ überschneidet (Zeile 10), kollidiert das Fahrzeug mit Hindernissen und die Prozedur kann beendet werden (Zeile 11).

5.3.6. Methode zum Warmstart der Pfadplanung

Ein Pfadplanungsproblem besteht aus einer gegebenen Startkonfiguration $\boldsymbol{q}_I$ und einer gegebenen Zielkonfiguration $\boldsymbol{q}_G$ sowie bekannten Hindernissen $\mathcal{O}$. Durch den vorgestellten Pfadplanungs-Algorithmus 5 wird für dieses Problem ein zugehöriger Lösungspfad $\boldsymbol{\Pi}$ berechnet. Verändert sich $\boldsymbol{q}_I$ oder $\boldsymbol{q}_G$ oder liegen aktualisierte Daten der Umgebung in Form neuer Hindernisse vor, ergibt sich ein neues Planungsproblem, sodass der gesamte Baum neu aufgebaut werden müsste, um eine Lösung zu finden. Mithilfe der entwickelten *WarmStart*-Prozedur werden möglichst viele Informationen aus einer vorangegangenen Lösung verwendent, um Konvergenz zu verbessern. Dabei werden Zweige des bestehenden Baums für das neue Planungsproblem zu übernommen. Dadurch startet die Pfadplanung nicht mit einem leeren Baum. Es wird angenommen, dass sich aufeinanderfolgende Planungsprobleme sehr ähnlich sind. Diese Annahme ist in der Praxis erfüllt, da eine neue Startkonfiguration üblicherweise in der Nähe einer vorherigen Startkonfiguration und oft auch in der Nähe des zuvor berechneten Lösungspfads liegt. Typischerweise ist die Zielkonfiguration konstant oder variiert minimal. Gleichzeitig wird sich der Großteil der Hindernisse, welche beispielsweise durch eine Umfelderkennung erkannt wurden, nur wenig verändern. Dennoch kann die *WarmStart*-Prozedur auch auf ein völlig anderes Planungsproblem angewendet werden, wodurch entsprechend weniger Zweige behalten werden. Unter Umständen muss der Baum auch komplett neu aufgebaut werden, was einem klassischen Kaltstart gleichkommt.

Algorithmus 11: *WarmStart*-Prozedur

```
1:  procedure T ← WarmStart(T, q_I, q_G, O, Δp)
2:      T.Transform(Δp);
3:      Π ← T.GetLatestSolutionBranch();
4:      q_c ← FindBestConnection(Π, q_I, O);          // see algorithm 12
5:      if q_c = ∅ then
6:          T.Init(q_I);
7:          return T;
8:      end
9:      T.MakeSubTree(q_c);
10:     T.InsertRoot(q_I, q_c);
11:     T.RemoveInfeasibleBranches(O);
12:     T.UpdateCostValues();
13: end
```



(a) Ergebnis eines vorangegangenen Problems mit dem zugehörigen Baum.

(b) Neues Problem mit anderen Start-, Zielkonfigurationen und neuen Hindernissen.

(c) Finden der besten Verbindung zum vorherigen Lösungspfad mit neuen Hindernissen.

(d) Setzen des neuen Wurzelknotens.

(e) Entfernen sämtlicher ungültiger Zweige.

(f) Resultierender Baum als Ausgangspunkt für das neue Planungsproblem.

Abbildung 5.12.: Veranschaulichung einzelner Teilschritte der *WarmStart*-Prozedur zur Initialisierung des Baums für ein neues Planungsproblem.

In Algorithmus 5 wird die *WarmStart*-Prozedur einmal vor der Iterationsschleife ausgeführt, um den Baum für das neue Planungsproblem zu initialisieren. Die entwickelte *WarmStart*-Prozedur ist in Algorithmus 11 gezeigt. Ebenso sind in Abbildung 5.12 einzelne Teilschritte grafisch dargestellt. Die Teilabbildung 5.12a zeigt den berechneten Lösungspfad für das vorangegangene Planungsproblem, bestehend aus $\boldsymbol{q}_{I,old}$, $\boldsymbol{q}_{G,old}$ und $\mathcal{O}_{old}$, mit dem zugehörigen Baum. Abbildung 5.12b illustriert das neue Planungsproblem zusammen mit dem Baum der vorangegangenen Lösung. Dabei sind die neuen Start- und Zielkonfigurationen minimal versetzt und die Hindernisse haben sich in Form und Anzahl verändert. Zu Beginn wird die Positionsdifferenz $\Delta\boldsymbol{p}$ zu allen Konfigurationen des Baums addiert (Zeile 2), um Verschiebungen des Koordinatenursprungs zu berücksichtigen, wie in Abschnitt 5.3 beschrieben. Im Anschluss wird der zuvor berechnete Lösungspfad ermittelt (Zeile 3). Dieser ist in Abbildung 5.12 durch eine grün-blaue Linie

gekennzeichnet. In Zeile 4 wird nach der besten, kollisionsfreien Verbindung entsprechend der Kostenfunktion (5.13) von der Startkonfiguration $\boldsymbol{q}_I$ zum alten Lösungspfad $\boldsymbol{\Pi}$ gesucht, wie in Abbildung 5.12c angedeutet. Dabei werden die aktuellen Hindernisse $\mathcal{O}$ berücksichtigt. Das Ergebnis ist eine Konfiguration $\boldsymbol{q}_c \in \boldsymbol{\Pi}$. Falls keine Verbindung existiert (Zeile 5), wird ein Kaltstart durchgeführt, indem der Baum mit der Startkonfiguration als Wurzelknoten initialisiert wird (Zeile 6). In Zeile 9 wird der Teilbaum ausgehend von $\boldsymbol{q}_c$ ermittelt und alle vorangegangenen Zweige werden entfernt. In Abbildung 5.12d sind die entfernten Zweige rot markiert. Anschließend wird $\boldsymbol{q}_I$ als Wurzelknoten des Baums gesetzt (Zeile 10). Gleichzeitig ist $\boldsymbol{q}_I$ der Elternknoten von $\boldsymbol{q}_c$. In Zeile 11 werden alle Zweige des Baums entfernt, bei denen es zur Kollision mit den neuen Hindernissen $\mathcal{O}$ kommt, wie in Abbildung 5.12e dargestellt ist. Zudem werden alle Teile des Baums entfernt, die außerhalb des neuen Sampling-Gebietes liegen, welches durch $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ definiert ist. Abschließend wird in Zeile 12 der Kostenwert für alle Knoten anhand der Kostenfunktion (5.13) neu berechnet. Das ist notwendig, da sich der Kostenterm c_μ aufgrund neuer Hindernisse verändert und die neue Kante von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_c$ zu einem Kostenoffset für alle nachfolgenden Knoten von $\boldsymbol{q}_c$ führt. Die Abbildung 5.12f zeigt den resultierenden Baum der *WarmStart*-Prozedur, der bei $\boldsymbol{q}_I$ beginnt und in $\mathcal{C}_{free}$ liegt. Am Ende der *WarmStart*-Prozedur wird nicht nach einer Verbindung zur Zielkonfiguration $\boldsymbol{q}_G$ gesucht, da es während des ersten Iterationsschrittes der Pfadplanung ohnehin zum *goal sampling* kommt, wie in 5.3.3 beschrieben. Im Hinblick auf das neue Planungsproblem sind die Knoten innerhalb des initialen Baums im Allgemeinen nicht optimal verbunden. Jedoch füllt der Baum bereits Teile von $\mathcal{C}_{free}$, sodass die Bereiche in Richtung des Ziels grob erkundet sind, sofern die getroffene Annahme, dass sich zwei aufeinanderfolgende Planungsprobleme sehr ähnlich sind, gilt. Durch gleichverteiltes Sampling werden die Kanten schließlich in den nachfolgenden Iterationsschritten durch die *Rewire*- und *FindBestParent*-Prozeduren des RRT*-Algorithmus fortlaufend optimiert. Daher tendiert auch ein ungünstiger initialer Baum bei zunehmender Anzahl an Samples immer gegen eine optimale Lösung.

In Algorithmus 12 ist die *FindBestConnection*-Prozedur im Detail dargestellt. Zu Beginn ist keine Konfiguration für die beste Verbindung zu $\boldsymbol{\Pi}$ bekannt (Zeile 2). Anschließend werden alle Konfigurationen aus $\boldsymbol{\Pi}$ untersucht (Zeile 4). Dabei wird die Kante von $\boldsymbol{q}_I$ zu $\boldsymbol{q}$ mithilfe der *IsFeasible*-Prozedur auf Gültigkeit getestet. Falls die Verbindung gültig ist, werden in Zeile 7 die Gesamtkosten c_i berechnet, die sich aus den Kosten der Verbindung von $\boldsymbol{q}_I$ zu $\boldsymbol{q}$ sowie den Kosten von $\boldsymbol{q}$ bis zum Ende des Lösungspfades zusammensetzen. Ist für c_i ein neues Minimum gefunden (Zeile 8), wird $\boldsymbol{q}_c = \boldsymbol{q}$ als aktuell beste Konfiguration gesetzt (Zeile 10).

Algorithmus 12: *FindBestConnection*-Prozedur

```
1:  procedure q_c ← FindBestConnection(Π, q_I, O)
2:      q_c ← ∅;
3:      c_min ← ∞;
4:      foreach q ∈ Π do
5:          ε ← Edge(q_I, q);
6:          if IsFeasible(ε, O) then
7:              c_i ← c(q_I, q)+ CostToEndOfPath(q);
8:              if c_i < c_min then
9:                  c_min ← c_i;
10:                 q_c ← q;
11:             end
12:         end
13:     end
14: end
```

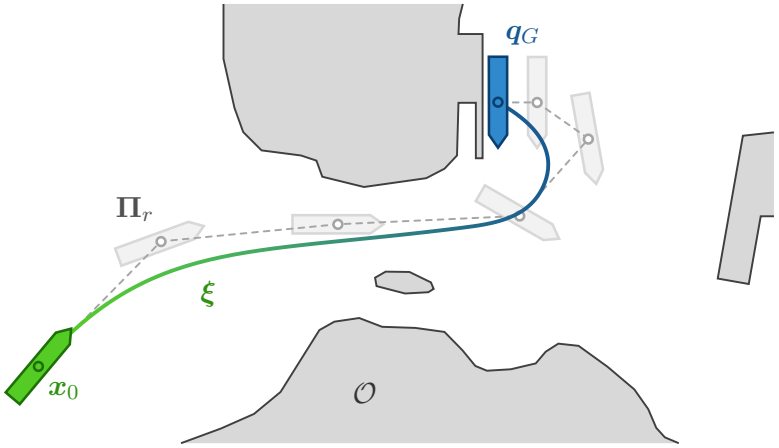


Abbildung 5.13.: Schematische Darstellung des Bewegungsplanungsproblems für ein ASV.

5.4. Bewegungsplanung

In der Bewegungsplanung wird der Lösungspfad aus der Pfadplanung benutzt, um in dessen Umgebung nach einer kollisionsfreien Trajektorie zu suchen. Es wird ein Verfahren zur Trajektoriengenerierung auf Basis eines gegebenen Pfades vorgestellt, welches auf der numerischen Simulation eines geschlossenen Regelkreises beruht. Ein Pfad dient dabei als Sollwertvorgabe für das Regelsystem und wird im Folgenden als Referenzpfad bezeichnet. Prinzipiell ließe sich aus dem Lösungspfad der Pfadplanung eine zugehörige Trajektorie erzeugen, jedoch wäre nicht sichergestellt, dass es entlang dieser Trajektorie keine Kollisionen der geometrischen Fahrzeugform mit Hindernissen gibt. Die Grundidee der Bewegungsplanung besteht deshalb darin, mit dem RRT*-Algorithmus einen Referenzpfad derart zu erzeugen, dass eine kollisionsfreie Trajektorie entsteht. Der Referenzpfad selbst muss dabei nicht kollisionsfrei sein.

Die Struktur des entwickelten Algorithmus zur Bewegungsplanung ist vergleichbar mit der der Pfadplanung. Allerdings werden einige *blackbox*-Funktionen modifiziert. Das Sampling-Gebiet wird auf die Umgebung des gegebenen Lösungspfades der Pfadplanung beschränkt. Der Suchraum entspricht weiterhin dem dreidimensionalen Konfigurationsraum $\mathcal{C}$ und nicht dem neundimensionalen Phasenraum $\mathcal{X}$, welcher Pose, Geschwindigkeit und Kraft in allen drei Freiheitsgraden enthält. Da das Volumen des Suchraums rapide mit jeder weiteren Dimension wächst, würde die Planung in $\mathcal{X}$ zu einem erheblichen Speicher- und Rechenaufwand führen. Durch die Reduktion des Suchraums auf drei Dimensionen kann eine gültige Trajektorie in akzeptabler Rechenzeit gefunden werden, zumal vorrangig das Erkunden von $\mathcal{C} \subset \mathcal{X}$ von Interesse ist. Dies hat zur Folge, dass nicht alle möglichen Trajektorien betrachtet werden. Beispielsweise wird nicht explizit nach der bestmöglichen Geschwindigkeit entlang der Trajektorie im Sinne eines gegebenen Gütefunktionals gesucht. Stattdessen ergeben sich die Zustandsverläufe durch das verwendete Regelgesetz in Kombination mit dem dynamischen Bewegungsmodell. Für eine Applikation des vorgestellten Verfahrens entsteht durch diese Art der Einschränkung kein signifikanter Nachteil, da sich durch das geregelte Fahren entlang eines Pfades plausible Zustands- und Stellgrößenverläufe ergeben.

Abbildung 5.13 illustriert die Problemstellung der Bewegungsplanung für ein ASV. Das Planungsproblem kann als das Finden eines optimalen Referenzpfades $\mathbf{\Pi}_r$ angesehen werden, welcher im Anfangszustand $\boldsymbol{x}_0$ beginnt und in der Zielkonfiguration $\boldsymbol{q}_G$ endet. Die Kostenfunktion zur Bewertung eines optimalen Referenzpfades entspricht der der Pfadplanung aus Abschnitt 5.3.2. Die resultierende Trajektorie, welche sich aus dem optimalen Referenzpfad ergibt, wird mit $\boldsymbol{\xi}$ bezeichnet. Sie beginnt im Anfangszustand $\boldsymbol{x}_0$, wobei dieser nicht Bestandteil der Trajektorie ist, und endet im Zielzustand $\boldsymbol{x}_G = (\boldsymbol{q}_G^{\mathrm{T}}, \mathbf{0}, \mathbf{0})^{\mathrm{T}}$. Sowohl die Geschwindigkeit $\boldsymbol{\nu}$ als auch die Kraft $\boldsymbol{\tau}$ muss im Zielzustand Null sein, damit $\mathcal{X}_{obs} = \mathcal{C}_{obs}$ gilt.

Algorithmus 13: Bewegungsplanung

Data: $\mathcal{T} \leftarrow \varnothing$; $\boldsymbol{x}_0 \leftarrow \varnothing$; $\boldsymbol{\tau}_{c,0} \leftarrow \varnothing$; $\boldsymbol{q}_{solution,old} \leftarrow \varnothing$

1: **procedure** $\boldsymbol{\xi} \leftarrow$ MotionPlanner$(\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}, \boldsymbol{\Pi}, \mathcal{O}, \Delta\boldsymbol{p}, t^m_{max})$
2: $t_{start} \leftarrow$ TimeNow();
3: **if** $\mathcal{V}(\boldsymbol{q}_I) \cap \mathcal{O} = \emptyset$ **then**
4: $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_I$;
5: $\boldsymbol{\tau}_{c,0} \leftarrow \boldsymbol{\tau}_{c,I}$;
6: $\boldsymbol{q}_I \leftarrow$ Pose$(\boldsymbol{x}_I)$;
7: $(\boldsymbol{\Pi}_t, \boldsymbol{q}_G) \leftarrow$ TrimPath$(\boldsymbol{\Pi}, L_{max})$; // see algorithm 15
8: $\mathcal{T}$.Transform$(\Delta\boldsymbol{p})$;
9: $\boldsymbol{\Pi}_{r,old} \leftarrow \mathcal{T}$.GetBranch$(\boldsymbol{q}_{solution,old})$;
10: $\mathcal{S} \leftarrow \{\boldsymbol{\Pi}_{r,old} \cap$ NewSamplingArea, $\boldsymbol{\Pi}_t \setminus \boldsymbol{q}_I\}$;
11: $\mathcal{T}$.Init$(\boldsymbol{q}_I)$;
12: $error \leftarrow$ False;
13: **while** $\neg error \ \wedge \ ($(TimeNow$() - t_{start}) < t^m_{max})$ **do**
14: $(\mathcal{T}, error) \leftarrow$ MotionPlannerIteration$(\mathcal{T}, \boldsymbol{q}_G, \mathcal{O})$; // see algorithm 14
15: **end**
16: **end**
17: $\boldsymbol{q}_{solution} \leftarrow$ NearestNeighbor$(\mathcal{T}, \boldsymbol{q}_G)$;
18: $\boldsymbol{\Pi}_r \leftarrow \mathcal{T}$.GetBranch$(\boldsymbol{q}_{solution})$;
19: $\boldsymbol{\xi} \leftarrow$ExplorePath$(\boldsymbol{x}_0, \boldsymbol{\tau}_{c,0}, \boldsymbol{\Pi}_r, \mathcal{O})$; // see algorithm 17
20: $\boldsymbol{q}_{solution,old} \leftarrow \boldsymbol{q}_{solution}$;
21: **end**

Die Bewegungsplanung ist in Algorithmus 13 dargestellt. Der initiale Zustandsvektor $\boldsymbol{x}_I = (\boldsymbol{\eta}_I^{\mathrm{T}}, \boldsymbol{\nu}_I^{\mathrm{T}}, \boldsymbol{\tau}_I^{\mathrm{T}})^{\mathrm{T}}$ sowie der initiale Stellgrößenvektor $\boldsymbol{\tau}_{c,I}$ und der resultierende Pfad $\boldsymbol{\Pi}$ aus der Pfadplanung bilden zusammen mit den Hindernissen $\mathcal{O}$ den Eingang der Bewegungsplanung. Wie bei der Pfadplanung wird für die Bewegungsplanung die translatorische Positionsdifferenz $\Delta\boldsymbol{p}$ sowie eine maximale Rechenzeit t^m_{max} vorgegeben, nach deren Überschreiten die Iterationsschleife beendet wird. Der initiale Zustandsvektor wird als $\boldsymbol{x}_0$ gespeichert (Zeile 4), da dieser Zustand, ebenso wie die Anfangsstellgröße $\boldsymbol{\tau}_{c,0}$, an verschiedenen Stellen innerhalb des Verfahrens benötigt wird. Nach dem bereits vorgestellten Konzept der sequenziellen Bewegungsplanung aus 5.1 erfolgt die eigentliche Bewegungsplanung in einem räumlich beschränkten Gebiet, um in der unmittelbaren Umgebung des ASVs nach einer bestmöglichen Lösung zu suchen. Dazu wird in Zeile 7 die Länge des gegebenen Pfades $\boldsymbol{\Pi}$ auf ein definiertes L_{max} beschränkt und die Zielkonfiguration $\boldsymbol{q}_G$ berechnet. Der resultierende Teilpfad der *TrimPath*-Prozedur wird mit $\boldsymbol{\Pi}_t$ bezeichnet. In den Zeilen 8 und 9 wird der Referenzpfad $\boldsymbol{\Pi}_{r,old}$ des vorangegangenen Planungsproblems ermittelt. Basierend auf diesem alten Referenzpfad und dem aktuellen Teilpfad wird ein Sample-Set $\mathcal{S}$ generiert (Zeile 10), welches eine Menge an initialen Konfigurationen enthält, die während der *Sample*-Prozedur der Bewegungsplanung zuerst gesampelt werden. Aus $\boldsymbol{\Pi}_{r,old}$ werden dabei nur die Konfigurationen übernommen, die im aktuellen Sampling-Gebiet liegen und aus $\boldsymbol{\Pi}_t$ werden alle Konfigurationen mit Ausnahme von $\boldsymbol{q}_I$ gewählt. Auf die Definition des Sampling-Gebietes wird detailliert in Abschnitt 5.4.2 eingegangen. Mithilfe des Sampling-Sets wird der Lösungspfad der vorangegangenen Bewegungsplanung als Ausgangsbasis verwendet, was einem Warmstart gleichkommt. Diese Vorgehensweise ist vorteilhaft, da ein zuvor berechneter Lösungspfad mit hoher Wahrscheinlichkeit auch eine kollisionsfreie Lösung mit geringen Kosten für das aktuelle Bewegungsplanungsproblem darstellt. In Zeile 11 wird der Baum mit der Startkonfiguration $\boldsymbol{q}_I = \boldsymbol{\eta}_I$ initialisiert. Dabei ist zu erwähnen, dass es sich bei diesem Baum um einen eigenständigen Graphen für die Bewegungsplanung mit separatem Datenspeicher handelt, der trotz der gleichen Namensgebung nicht mit dem Baum der Pfadplanung zu verwechseln ist.

In Zeile 14 erfolgen die Iterationsschritte der Bewegungsplanung, solange kein Fehler auftritt und die Rechenzeit das vorgegebene Zeitintervall t^m_{max} nicht überschreitet (Zeile 13). Anschließend wird in Zeile 17 der dichteste Knoten des Baums zur Zielkonfiguration $\boldsymbol{q}_G$ ausgewählt. Der Referenzpfad $\boldsymbol{\Pi}_r$ entspricht dem Zweig vom Wurzelknoten zum gewählten Lösungsknoten (Zeile 18). Mithilfe der *ExplorePath*-Funktion erfolgt in Zeile 19 die Konvertierung des Referenzpfades zur resultierenden Trajektorie $\boldsymbol{\xi}$.

In Algorithmus 14 ist der Iterationsschritt der Bewegungsplanung dargestellt. Dieser entspricht ebenfalls dem Iterationsschritt des RRT*-Algorithmus und ist vergleichbar mit dem Iterationsschritt der Pfadplanung aus Algorithmus 6. Der Unterschied zur Pfadplanung ergibt sich durch andere Funktionalitäten einiger *black-box*-Funktionen. Zum einen wird das Sampling dahingehend modifiziert, dass Posen aus der Umgebung des gegebenen Pfades genutzt werden. Zum anderen wird die *IsFeasible*-Funktion erweitert, damit Trajektorien für den Kollisionstest verwendet werden können, die sich aus der numerischen Simulation eines geregelten Bewegungsmodells ergeben. Die maximale Anzahl an Knoten, die im Rahmen der Bewegungsplanung in $\mathcal{T}$ gespeichert werden können, wird mit n^m_{max} bezeichnet und muss nicht identisch zu n^p_{max} sein. In Zeile 12 wird der gesamte Zweig vom Wurzelknoten zum neuen Knoten $\boldsymbol{q}_{new}$ ermittelt, welcher in Zeile 13 für den Kollisionstest verwendet wird.

Algorithmus 14: Ein Iterationsschritt der Bewegungsplanung

```
1:  procedure (T, error) ← MotionPlannerIteration(T, q_G, O)
2:      error ← FALSE;
3:      if T.IsFull() then
4:          error ← T.RemoveRandomLeaf();
5:          if error then
6:              return;
7:          end
8:      end
9:      q_rand ← Sample(T, q_G);                          // see algorithm 16
10:     q_near ← NearestNeighbor(T, q_rand);
11:     q_new ← Steer(T, q_near, q_rand);
12:     Π_branch ← {T.GetBranchFromRoot(q_near), q_new};
13:     if IsFeasible(Π_branch, O) then
14:         V ← Near(T, q_new);
15:         q_parent ← FindBestParent(V, q_near, q_new);   // see algorithm 3
16:         T.Insert(q_parent, q_new);
17:         Rewire(T, V \ q_parent, q_new);               // see algorithm 21
18:     end
19: end
```

5.4.1. TrimPath-Funktion

Mittels der *TrimPath*-Prozedur wird die Pfadlänge eines gegebenen Pfades $\boldsymbol{\Pi}$ in $\mathbb{R}^2$ reduziert. Es entsteht ein Teilpfad $\boldsymbol{\Pi}_t$, dessen Endkonfiguration die Zielkonfiguration $\boldsymbol{q}_G$ der Bewegungsplanung darstellt. Die maximale Pfadlänge wird mit dem Parameter L_{max} festgelegt. Durch die Reduktion des Pfades wird vermieden, dass der gesamte Konfigurationsraum der Pfadplanung auch für die Bewegungsplanung gesampelt werden muss. Stattdessen wird nur eine Untermenge des Pfades verwendet, um den bestmöglichen Bewegungsplan zu finden.

Algorithmus 15 zeigt die *TrimPath*-Prozedur, welche aus einem gegebenen Pfad $\boldsymbol{\Pi}$ sowie einer maximale Weglänge L_{max} den Teilpfad $\boldsymbol{\Pi}_t$ sowie die Zielkonfiguration $\boldsymbol{q}_G$ für die Bewegungsplanung berechnet. Zunächst wird $\boldsymbol{q}_G$ mit der Endkonfiguration des Pfades initialisiert (Zeile 2).

Algorithmus 15: *TrimPath*-Prozedur

1: **procedure** $(\boldsymbol{\Pi}_t, \boldsymbol{q}_G) \leftarrow \text{TrimPath}(\boldsymbol{\Pi}, L_{max})$
2: $\quad \boldsymbol{q}_G \leftarrow \boldsymbol{\Pi}.\text{GetFinalPose}()$;
3: $\quad \boldsymbol{\Pi}_t \leftarrow \boldsymbol{q}_G$;
4: $\quad N \leftarrow \boldsymbol{\Pi}.\text{NumberOfPoses}()$;
5: $\quad L_\Sigma \leftarrow 0$;
6: $\quad$ **for** $i = 1$ **to** N **do**
7: $\quad\quad \Delta L \leftarrow \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2}$;
8: $\quad\quad$ **if** $(L_\Sigma + \Delta L) > L_{max}$ **then**
9: $\quad\quad\quad \boldsymbol{q}_G \leftarrow \boldsymbol{\sigma}\left(\boldsymbol{q}_{i-1} + \frac{L_{max} - L_\Sigma}{\Delta L} \boldsymbol{\sigma}\left(\boldsymbol{q}_i - \boldsymbol{q}_{i-1}\right)\right)$;
10: $\quad\quad\quad \boldsymbol{\Pi_t} \leftarrow \{\boldsymbol{\Pi}(0),\ \ldots,\ \boldsymbol{\Pi}(i-1),\ \boldsymbol{q}_G\}$;
11: $\quad\quad\quad$ **break**
12: $\quad\quad$ **end**
13: $\quad\quad L_\Sigma \leftarrow L_\Sigma + \Delta L$;
14: $\quad$ **end**
15: **end**

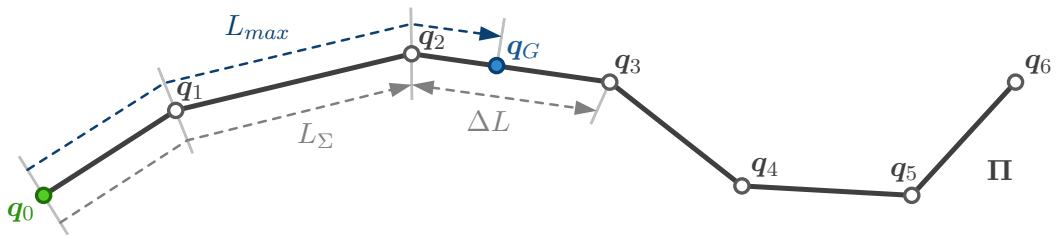


Abbildung 5.14.: Schematische Darstellung der *TrimPath*-Prozedur für die Iteration $i = 3$.

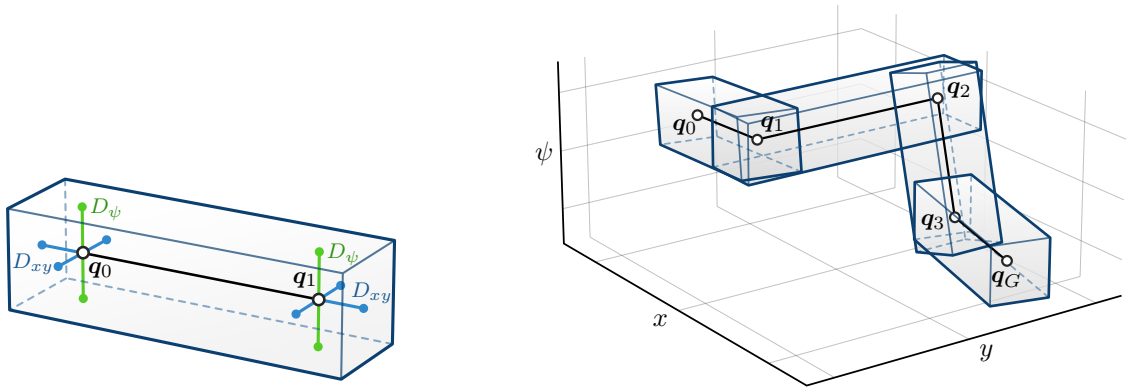
Anschließend wird in Zeile 6 über alle Teilstrecken des Pfades iteriert. Die bisherige Pfadlänge seit dem Beginn der Iteration wird in L_Σ gespeichert. Die Länge einer Teilstrecke ΔL wird in Zeile 7 berechnet. Falls die Summe aus der bisherigen Pfadlänge L_Σ und der Länge der Teilstrecke die maximale Länge überschreitet (Zeile 8), wird die Zielkonfiguration $\boldsymbol{q}_G$ berechnet (Zeile 9) und die Iteration kann beendet werden (Zeile 11). Andernfalls wird L_Σ aktualisiert (Zeile 13) und die nächste Teilstrecke wird untersucht. Für die Berechnung von $\boldsymbol{q}_G$ in Zeile 9 wird mithilfe der $\boldsymbol{\sigma}$-Funktion nach Gleichung 5.11 die topologische Eigenschaft des Konfigurationsraums berücksichtigt.

Die Abbildung 5.14 verdeutlicht die *TrimPath*-Prozedur anhand eines Pfades mit sieben Konfigurationen. Das Teilziel $\boldsymbol{q}_G$ wird während der Iteration $i = 3$ berechnet und die zugehörigen Variablen L_{max}, L_Σ und ΔL sind gekennzeichnet. Der resultierende Teilpfad wäre in diesem Beispiel gegeben mit

$$\boldsymbol{\Pi}_t = \{\boldsymbol{q}_0, \boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_G\} \ .$$

5.4.2. Sampling

Für das Sampling muss der unendlich große Konfigurationsraum auf ein endliches Gebiet beschränkt werden. Der Teilpfad $\boldsymbol{\Pi}_t$, welcher mithilfe der *TrimPath*-Prozedur berechnet wurde, stellt die Grundlage für die entwickelte Sampling-Heuristik dar. Es wird angenommen, dass der optimale Referenzpfad, nach dem in der Bewegungsplanung gesucht wird, in der Nähe von $\boldsymbol{\Pi}_t$ liegt. Diese Annahme ist damit begründet, dass die Kostenfunktion der Bewegungsplanung identisch zu der der Pfadplanung ist.



(a) Illustration der Größe einer Sampling-Box um ein Pfadsegment basierend auf den zwei Parametern D_{xy} und D_ψ.

(b) Grafische Darstellung des gesamten Sampling-Gebietes durch mehrere Sampling-Boxen entlang eines Teilpfades.

Abbildung 5.15.: Schematische Darstellung des Sampling-Gebietes für die Bewegungsplanung auf Basis eines gegebenen Teilpfades $\mathbf{\Pi}_t$.

Für das Sampling werden um jedes Pfadsegment des Teilpfades dreidimensionale Sampling-Boxen erzeugt. Eine schematische Darstellung ist in Abbildung 5.15 gegeben. Jede Sampling-Box ist an dem zugehörigen Pfadsegment ausgerichtet. Die Größe einer Sampling-Box wird so gewählt, dass der senkrechte Abstand eines Samples zum Pfad in Positionsrichtung mit dem Parameter $D_{xy} > 0$ und in Winkelrichtung mit $D_\psi > 0$ festgelegt werden kann, wie Teilabbildung 5.15a illustriert. Anhand von Abbildung 5.15b ist zu erkennen, dass sich die Sampling-Boxen an den Konfigurationen des Pfades überlappen. Folglich wird in den überlappenden Bereichen mit einer größeren Wahrscheinlichkeit gesampelt. Die Verwendung von Sampling-Boxen hat jedoch den Vorteil, dass direkt die Halton-Sequenz für das Sampling verwendet werden kann. Für den k-ten Iterationsschritt der Bewegungsplanung wird ein zufälliger Sample mit

$$\boldsymbol{q}_k = \boldsymbol{\sigma}\left(\boldsymbol{A}_{box,i}\boldsymbol{s}_k + \boldsymbol{b}_{box,i}\right) \tag{5.39}$$

berechnet. Hierbei ist

$$\boldsymbol{s}_k = \begin{pmatrix} H(k,2) \\ H(k,3) \\ H(k,5) \end{pmatrix} \in (0,1)^3 \tag{5.40}$$

der k-te Sample der Halton-Sequenz. Die Matrix $\boldsymbol{A}_{box,i}$ enthält die Skalierung sowie die Rotation der i-ten Sampling-Box und der Vektor $\boldsymbol{b}_{box,i}$ repräsentiert die zugehörige Translation dieser Sampling-Box. Das Volumen einer Sampling-Box wird mit $V_{box,i}$ bezeichnet. Eine Herleitung der Größen $\boldsymbol{A}_{box,i}$, $\boldsymbol{b}_{box,i}$ und $V_{box,i}$ ist in Appendix A.5 zu finden. Um aus einer Anzahl von M Sampling-Boxen diejenige auszuwählen, aus der nach (5.39) Samples generiert werden sollen, wird der Anteil des Volumens einer Sampling-Box in Relation zum Gesamtvolumen aller Sampling-Boxen betrachtet. Dieses Verhältnis wird mit einer Zufallszahl verglichen, die ebenfalls mithilfe der Halton-Sequenz bestimmt wird. Innerhalb der k-ten Iteration wird der Index i einer Sampling-Box mit

$$i = \underset{m\in[1,M]}{\arg\min}\left(\frac{\sum_{i=1}^{m} V_{box,i}}{\sum_{i=1}^{M} V_{box,i}} > H(k,7)\right) \tag{5.41}$$

berechnet.

Die Sample-Prozedur der Bewegungsplanung ist in Algorithmus 16 zusammengefasst und entspricht grundsätzlich der Sample-Prozedur der Pfadplanung aus Algorithmus 7, wobei während der ersten Iterationen der Bewegungsplanung die Sampling-Strategie modifiziert wird.

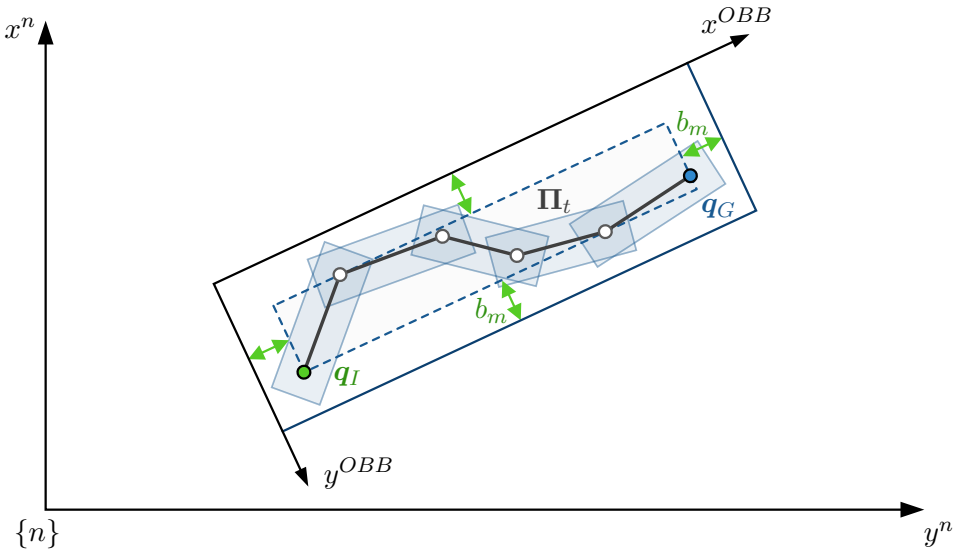


Abbildung 5.16.: Schematische Darstellung der OBB auf Basis des Teilpfades $\mathbf{\Pi}_t$.

Für die ersten Samples werden die Konfigurationen aus dem Sample-Set $\mathcal{S}$ gewählt, da diese eine vielversprechende Ausgangslösung für das Planungsproblem darstellen. Das Sample-Set enthält alle Konfigurationen des Referenzpfades $\mathbf{\Pi}_{r,old}$ einer vorangegangenen Lösung, die sich innerhalb der aktuellen Sampling-Boxen befinden, sowie alle $\boldsymbol{q} \in \mathbf{\Pi}_t \setminus \boldsymbol{q}_I$. Falls dieses Sample-Set noch nicht leer ist (Zeile 2), wird die erste Konfiguration aus diesem Set gezogen (Zeile 3) und in Zeile 4 aus $\mathcal{S}$ entfernt. Wurden alle Samples aus $\mathcal{S}$ gezogen, erfolgt in den Zeilen 6-13 das Sampling nach dem gleichen Grundprinzip wie dem der Pfadplanung, mit dem Unterschied, dass in den Zeilen 7 und 8 Samples nach der beschriebenen Heuristik aus der Umgebung des Teilpfades generiert werden.

Algorithmus 16: *Sample*-Prozedur der Bewegungsplanung

Data: $k \leftarrow 0$; $g \leftarrow 0$; $\mathcal{S} \leftarrow \{\mathbf{\Pi}_{r,old} \cap \text{NewSamplingArea},\ \mathbf{\Pi}_t \setminus \boldsymbol{q}_I\}$

```
1:  procedure q_rand ← Sample(T, q_G)
2:      if S ≠ ∅ then
3:          q_rand ← S.GetFirstValue();
4:          S.Remove(q_rand);
5:      else
6:          if (q_G ∈ T) ∨ (g_s ≠ 0) then
7:              i ← SelectRandomBoxIndex(k);          // according to (5.41)
8:              q_rand ← BoxSampling(i, k);           // according to (5.39)
9:              k ← (k+1) mod k_max;
10:         else
11:             q_rand ← q_G;                         // goal sampling
12:         end
13:         g ← (g+1) mod g_max;
14:     end
15: end
```

Für die numerische Berechnung des Kostenterms c_μ der Kostenfunktion muss eine LUT erzeugt werden, wie in Abschnitt 5.3.2 beschrieben. Die Position und Größe der LUT wird durch eine OBB vorgegeben, wie Abbildung 5.16 zeigt. Zunächst wird diejenige OBB gesucht, welche alle $\boldsymbol{q} \in \mathbf{\Pi}_t$ enthält und den minimalen Flächeninhalt in $\mathbb{R}^2$ aufweist. In der Abbildung

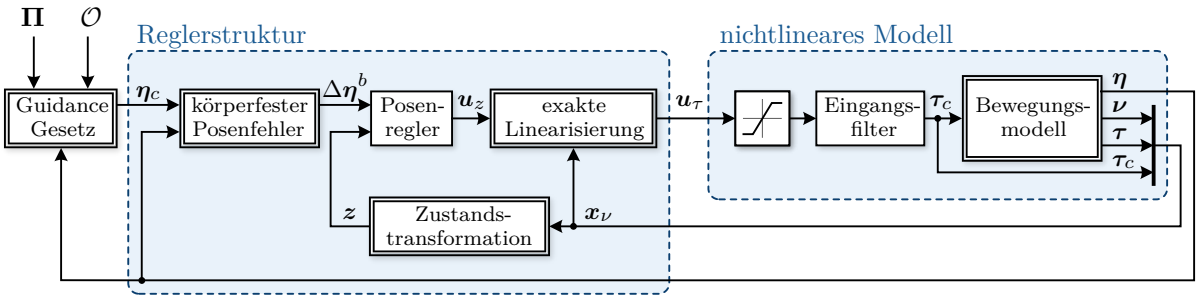


Abbildung 5.17.: Schematische Darstellung der Reglerstruktur für die *ExplorePath*-Prozedur auf der Basis einer exakten Linearisierung.

ist die OBB durch eine gestrichelte Box dargestellt. Ein Verfahren zur Berechnung einer OBB mit minimalem Flächeninhalt findet sich in [111]. Anschließend wird diese Box in jeder Richtung um die Länge

$$b_m = \sqrt{2} D_{xy} + \max\left(\|\boldsymbol{\kappa}_i^b\|_2\right) \tag{5.42}$$

erweitert. Der erste Summand entspricht hierbei dem maximalen Abstand der Ecke einer Sampling-Box zum Pfad und der zweite Summand beschreibt die maximale Länge des Vektors $\boldsymbol{\kappa}_i^b$ aus (5.19), welcher für die Auswertung des Kostenterms (5.18) verwendet wird.

5.4.3. ExplorePath-Funktion

Die *ExplorePath*-Prozedur stellt eine Erweiterung im Vergleich zur Pfadplanung dar. Mithilfe dieser Funktion wird für einen gegebenen Anfangszustand $\boldsymbol{x}_0$ auf Basis eines gegebenen Pfades $\boldsymbol{\Pi}$ eine Trajektorie $\boldsymbol{\xi}$ generiert. Entlang dieser Trajektorie müssen die differenziellen Beschränkungen gemäß den Gleichungen (5.1)-(5.3) gelten. Mit anderen Worten: Der zeitliche Verlauf der Eingangs- und Zustandsgrößen muss dem nichtlinearen Bewegungsmodell des ASVs entsprechen. Innerhalb der Bewegungsplanung müssen zahlreiche Trajektorien erzeugt werden, um den Konfigurationsraum mit möglichst vielen Samples zu erkunden. Die Trajektoriengenerierung muss demzufolge eine geringe Rechenzeit aufweisen. Es wird eine Methode vorgeschlagen, bei der die Trajektorie durch eine numerische Simulation eines geschlossenen Regelkreises berechnet wird. Diese Vorgehensweise hat zudem den Vorteil, dass sich Störgrößen wie beispielsweise der Einfluss von Wind und Strömung durch eine Störgrößenaufschaltung berücksichtigen lassen. Ebenso ließen sich weitere Effekte wie Totzeiten, Hysteresen oder Totzonen in die numerische Simulation integrieren. Die Abbildung 5.17 zeigt die vorgeschlagene Reglerstruktur auf Basis einer exakten Linearisierung für die Trajektoriengenerierung traversierfähiger Wasserfahrzeuge. Das zu regelnde nichtlineare Modell setzt sich aus einem linearen Eingangsfilter und dem nichtlinearen Bewegungsmodell des ASVs zusammen. Das Eingangsfilter besteht aus einem Tiefpass erster Ordnung und dient der Vermeidung sprunghafter Änderungen der Stellgröße $\boldsymbol{\tau}_c$. Es ermöglicht zudem eine Stellratenbeschränkung und ist mit

$$\dot{\boldsymbol{\tau}}_c = \underbrace{\begin{pmatrix} -\frac{1}{T_{f1}} & 0 & 0 \\ 0 & -\frac{1}{T_{f2}} & 0 \\ 0 & 0 & -\frac{1}{T_{f3}} \end{pmatrix}}_{\boldsymbol{A}_c} \boldsymbol{\tau}_c + \underbrace{\begin{pmatrix} \frac{1}{T_{f1}} & 0 & 0 \\ 0 & \frac{1}{T_{f2}} & 0 \\ 0 & 0 & \frac{1}{T_{f3}} \end{pmatrix}}_{\boldsymbol{B}_c} \boldsymbol{u}_\tau \tag{5.43}$$

gegeben, wobei die Zeitkonstanten T_{f1}, T_{f2} und T_{f3} zusätzliche Tuningparameter kennzeichnen. Die neue Eingangsgröße für das nichtlineare Modell wird mit $\boldsymbol{u}_\tau$ bezeichnet. Im Folgenden ist mit dem Begriff **Stellgröße** die tatsächliche Stellgröße $\boldsymbol{\tau}_c$ gemeint, während $\boldsymbol{u}_\tau$ als **Eingangsgröße** bezeichnet wird.

Mit einer nichtlinearen Zustandstransformation wird aus den Größen $\boldsymbol{\nu}$, $\boldsymbol{\tau}$ und $\boldsymbol{\tau}_c$ ein transformierter Zustandsvektor $\boldsymbol{z}$ berechnet, mit dessen Hilfe das nichtlineare Geschwindigkeitsmodell (5.2) exakt linearisiert wird. Auf Geschwindigkeitsebene entsteht ein lineares System. Das *Guidance*-Gesetz berechnet aus dem gegebenen Pfad $\boldsymbol{\Pi}$, den Hindernissen $\mathcal{O}$ und der aktuellen Pose $\boldsymbol{\eta}$ eine kommandierte Pose $\boldsymbol{\eta}_c$, die auf dem Pfad liegt. Anschließend wird die Posendifferenz aus $\boldsymbol{\eta}_c$ und $\boldsymbol{\eta}$ in das körperfeste Koordinatensystem transformiert und zusammen mit $\boldsymbol{z}$ an den Posenregler übergeben, der unter Verwendung einer linearen Zustandsrückführung die Posendifferenz auf Null bringt. Die berechnete Eingangsgröße $\boldsymbol{u}_\tau$ wird auf den erlaubten Stellgrößenbereich beschränkt. Da das Eingangsfilter aus einem Verzögerungsglied erster Ordnung mit einer Verstärkung von Eins besteht, liegt die Stellgröße $\boldsymbol{\tau}_c$ ebenfalls im Bereich der realisierbaren Kräfte und Momente. Auf die einzelnen Teilblöcke wird in den folgenden Unterabschnitten eingegangen.

In Algorithmus 17 ist die *ExplorePath*-Prozedur gezeigt. Sie besteht im Wesentlichen aus einer Iterationsschleife, in der die numerische Integration des in Abbildung 5.17 dargestellten Systems durchgeführt wird. In Zeile 3 werden die Zustandsgrößen mit dem Anfangszustand $\boldsymbol{x}_0 = (\boldsymbol{\eta}_0^\mathrm{T}, \boldsymbol{\nu}_0^\mathrm{T}, \boldsymbol{\tau}_0^\mathrm{T})^\mathrm{T}$ initialisiert und in Zeile 4 wird $\boldsymbol{\tau}_c$ mit der Anfangsstellgröße $\boldsymbol{\tau}_{c,0}$ initialisiert. Anschließend wird in Zeile 5 die Iterationsschleife mit maximal k_{max} Schritten ausgeführt. In den Zeilen 6-11 erfolgt die algebraische Berechnung der Eingangsgröße $\boldsymbol{u}_\tau$, wie in Abbildung 5.17 dargestellt ist. In Zeile 12 wird die numerische Integration des nichtlinearen Modells durchgeführt, welches sich aus dem Eingangsfilter nach (5.43) sowie dem nichtlinearen Bewegungsmodell nach (5.1)-(5.3) zusammensetzt. Als Integrationsverfahren wird das klassische Runge-Kutta-Verfahren (RK4) mit einer konstanten Abtastzeit T_s verwendet. Schließlich werden die Zustandsgrößen sowie die Stellgröße zur Trajektorie hinzugefügt (Zeile 13). Falls mindestens $k_{min} \geq 1$ Integrationsschritte berechnet wurden und die Pose sowie die Geschwindigkeit und die Kraft in einem definierten Zielgebiet liegen (Zeile 14), wird die Prozedur abgebrochen. Das Zielgebiet gilt als erreicht, wenn alle der folgenden Bedingungen erfüllt sind.

$$\begin{array}{lll} |x - x_N| < \Delta x_{th} & |u| < u_{th} & |X| < X_{th} \\ |y - y_N| < \Delta y_{th} \quad , & |v| < v_{th} \quad , & |Y| < Y_{th} \\ |\psi - \psi_N| < \Delta \psi_{th} & |r| < r_{th} & |N| < N_{th} \end{array} \tag{5.44}$$

Algorithmus 17: *ExplorePath*-Prozedur der Bewegungsplanung

1: **procedure** $(\boldsymbol{\xi}, k) \leftarrow$ ExplorePath$(\boldsymbol{x}_0, \boldsymbol{\tau}_{c,0}, \boldsymbol{\Pi}, \mathcal{O})$
2: $\quad \boldsymbol{\xi} \leftarrow \emptyset$;
3: $\quad (\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau}) \leftarrow \boldsymbol{x}_0$;
4: $\quad \boldsymbol{\tau}_c \leftarrow \boldsymbol{\tau}_{c,0}$;
5: $\quad$ **for** $k = 1$ **to** k_{max} **do**
6: $\quad\quad \boldsymbol{\eta}_c \leftarrow$ GuidanceLaw$(\boldsymbol{\eta}, \boldsymbol{\Pi}, \mathcal{O})$; `// see algorithm 18`
7: $\quad\quad \Delta\boldsymbol{\eta}^b \leftarrow$ TransformToBodyFixedState$(\boldsymbol{\eta_c}, \boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau}, \boldsymbol{\tau}_c)$;
8: $\quad\quad \boldsymbol{z} \leftarrow$ NonlinearStateTransformation$(\boldsymbol{\nu}, \boldsymbol{\tau}, \boldsymbol{\tau}_c)$;
9: $\quad\quad \boldsymbol{x}_\eta \leftarrow (\Delta\boldsymbol{\eta}^b, \boldsymbol{z})$;
10: $\quad\quad \boldsymbol{u}_z \leftarrow$ PoseControl$(\boldsymbol{x}_\eta)$;
11: $\quad\quad \boldsymbol{u}_\tau \leftarrow$ FeedbackLinearization$(\boldsymbol{x}_\nu, \boldsymbol{u}_z)$;
12: $\quad\quad (\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau}, \boldsymbol{\tau}_c) \leftarrow$NonlinearModelSimulation$(\boldsymbol{u}_\tau, T_s)$;
13: $\quad\quad \boldsymbol{\xi}$.Add$((\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau}), \boldsymbol{\tau}_c)$;
14: $\quad\quad$ **if** $k \geq k_{min} \wedge$ InsideGoalRegion$(\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau})$ **then**
15: $\quad\quad\quad$ **break**
16: $\quad\quad$ **end**
17: $\quad$ **end**
18: **end**

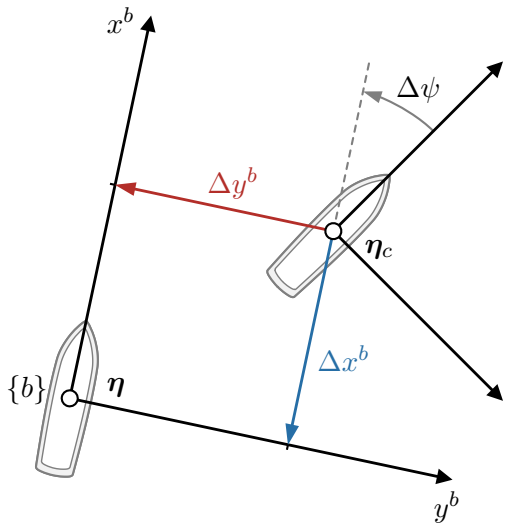


Abbildung 5.18.: Darstellung der Posendifferenz zwischen $\boldsymbol{\eta}$ und $\boldsymbol{\eta}_c$ bezogen auf das körperfeste Koordinatensystem $\{b\}$.

Hierbei ist $(x_N, y_N, \psi_N)^{\mathrm{T}}$ die letzte Pose des Pfades und Δx_{th}, Δy_{th} sowie $\Delta\psi_{th}$ geben die Größe des Zielgebietes in Form einer dreidimensionalen Box in $SE(2)$ an. Gleichzeitig muss die körperfeste Geschwindigkeit betragsmäßig unter einem Schwellwert liegen, welcher mit u_{th}, v_{th} und r_{th} vorgegeben wird. Ähnliches gilt für den Kräfte-Momente-Vektor. Die Verwendung eines Zielgebietes ist notwendig, da bei der vorgestellten Reglerstruktur eine Zielpose mit $\boldsymbol{\nu} = \mathbf{0}$ und $\boldsymbol{\tau} = \mathbf{0}$ niemals exakt in endlicher Zeit erreicht werden kann.

5.4.3.1. Nichtlineare Dynamik der körperfesten Posendifferenz

Zunächst wird ein mathematisches Modell abgeleitet, welches den Zusammenhang zwischen der Eingangsgröße $\boldsymbol{u}_\tau$ und der körperfesten Posendifferenz herstellt. Basierend auf der vom *Guidance*-Gesetz berechneten Sollpose $\boldsymbol{\eta}_c = (x_c, y_c, \psi_c)^{\mathrm{T}}$ sowie der aktuellen Pose $\boldsymbol{\eta}$ wird die Posendifferenz bestimmt, welche im *n-frame* mit

$$\Delta\boldsymbol{\eta}^n = \begin{pmatrix} \Delta x^n \\ \Delta y^n \\ \Delta\psi \end{pmatrix} = \begin{pmatrix} x - x_c \\ y - y_c \\ \sigma(\psi - \psi_c) \end{pmatrix} \tag{5.45}$$

gegeben ist. Unter Verwendung der transponierten Transformationsmatrix aus (5.5) wird diese Posendifferenz vom Navigationskoordinatensystem in das körperfeste Koordinatensystem transformiert.

$$\Delta\boldsymbol{\eta}^b = \begin{pmatrix} \Delta x^b \\ \Delta y^b \\ \Delta\psi \end{pmatrix} = \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\Delta\boldsymbol{\eta}^n \tag{5.46}$$

Die körperfeste Posendifferenz entspricht der Pose des Fahrzeuges bezogen auf eine gegebene Referenzpose $\boldsymbol{\eta}_c$ in den Koordinaten des körperfesten Koordinatensystems, wie in Abbildung 5.18 dargestellt ist. Um das dynamische Verhalten von $\Delta\boldsymbol{\eta}^b$ beschreiben zu können, wird die zeitliche Ableitung von (5.46) berechnet.

$$\begin{aligned} \Delta\dot{\boldsymbol{\eta}}^b &= \dot{\boldsymbol{T}}_b^{n,\mathrm{T}}(\psi)\Delta\boldsymbol{\eta}^n + \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\Delta\dot{\boldsymbol{\eta}}^n \\ &= \boldsymbol{Q}^{\mathrm{T}}(r)\boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\Delta\boldsymbol{\eta}^n + \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\left(\dot{\boldsymbol{\eta}} - \dot{\boldsymbol{\eta}}_c\right) \\ &= \boldsymbol{Q}^{\mathrm{T}}(r)\Delta\boldsymbol{\eta}^b + \boldsymbol{\nu} - \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\dot{\boldsymbol{\eta}}_c \end{aligned} \tag{5.47}$$

Hierbei ist

$$\boldsymbol{Q}(r) = \begin{pmatrix} 0 & -r & 0 \\ r & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{5.48}$$

eine schiefsymmetrische Matrix, welche die aktuelle Drehrate r enthält. Anhand von (5.47), des Geschwindigkeits- und Kraftmodells (5.2) bzw. (5.3) sowie dem Eingangsfilter (5.43) lässt sich ein mathematisches Modell in Form eines nichtlinearen Differenzialgleichungssystems ableiten. Es ist gegeben mit

$$\Delta\dot{\boldsymbol{\eta}}^b = \boldsymbol{Q}^{\mathrm{T}}(r)\Delta\boldsymbol{\eta}^b + \boldsymbol{\nu} - \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\dot{\boldsymbol{\eta}}_c \tag{5.49}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \tag{5.50}$$

$$\dot{\boldsymbol{\tau}} = \boldsymbol{A}_\tau\boldsymbol{\tau} + \boldsymbol{B}_\tau\boldsymbol{\tau}_c \tag{5.51}$$

$$\dot{\boldsymbol{\tau}}_c = \boldsymbol{A}_c\boldsymbol{\tau}_c + \boldsymbol{B}_c\boldsymbol{u}_\tau \ . \tag{5.52}$$

5.4.3.2. Exakte Linearisierung

Der Grundgedanke für den Einsatz einer exakten Linearisierung ist, dass bekannte nichtlineare Effekte im Geschwindigkeitsmodell (5.2) durch eine nichtlineare Zustandsrückführung kompensiert werden. Die exakte Linearisierung wird nicht für das gesamte nichtlineare Differenzialgleichungssystem (5.49)-(5.52) durchgeführt. Der Grund dafür ist, dass keine Beschreibung für die zeitliche Änderung von $\boldsymbol{\eta}_c$ in (5.49) existiert, da $\boldsymbol{\eta}_c$ nicht mit einer differenzierbaren Funktion, sondern anhand von Fallunterscheidungen berechnet wird. Das Teilsystem (5.50)-(5.52) hängt nicht von $\Delta\boldsymbol{\eta}^b$ ab, sodass für die exakte Linearisierung ausschließlich dieses Teilsystem betrachtet wird, um Nichtlinearitäten in $\boldsymbol{f}(\boldsymbol{\nu})$ zu kompensieren. Der Zustandsvektor des nichtlinearen Teilsystems ist mit

$$\boldsymbol{x}_\nu = \left(\boldsymbol{\nu}^{\mathrm{T}},\ \boldsymbol{\tau}^{\mathrm{T}},\ \boldsymbol{\tau}_c^{\mathrm{T}}\right)^{\mathrm{T}} \tag{5.53}$$

gegeben. Er besitzt die Dimension $n = 9$. Der Ausgangsvektor

$$\boldsymbol{y}_\nu = \begin{pmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{pmatrix} \boldsymbol{x}_\nu \tag{5.54}$$

entspricht der körperfesten Geschwindigkeit $\boldsymbol{\nu}$. Nach dem Entwurfsverfahren der exakten Linearisierung [12] lässt sich eine nichtlineare Zustandstransformation der Form

$$\boldsymbol{z} = \boldsymbol{t}(\boldsymbol{x}_\nu)$$

finden, indem die zeitlichen Ableitungen des Ausgangsvektors berechnet werden. Das nichtlineare Zustandsraummodell setzt sich aus einer Kette mit drei vektoriellen Integratoren zusammen, wobei die Eingangsgröße $\boldsymbol{u}_\tau \in \mathbb{R}^m$ mit $m = 3$ nur auf den Eingang dieser Integratorkette wirkt. Demzufolge erscheint die Eingangsgröße erst in der dritten zeitlichen Ableitung der Ausgangsgröße. Das zeitliche Differenzieren von (5.54) ergibt

$$\begin{aligned} \dot{\boldsymbol{y}}_\nu &= \frac{\partial \boldsymbol{y}_\nu}{\partial \boldsymbol{x}_\nu}\frac{d\boldsymbol{x}_\nu}{dt} = \begin{pmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\nu}} \\ \dot{\boldsymbol{\tau}} \\ \dot{\boldsymbol{\tau}}_c \end{pmatrix} \\ &= \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \ . \end{aligned} \tag{5.55}$$

Ein erneutes Ableiten liefert

$$\begin{aligned} \ddot{\boldsymbol{y}}_\nu &= \frac{\partial \dot{\boldsymbol{y}}_\nu}{\partial \boldsymbol{x}_\nu}\frac{d\boldsymbol{x}_\nu}{dt} = \begin{pmatrix} \frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}} & \boldsymbol{B} & \boldsymbol{0}_{3\times3} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\nu}} \\ \dot{\boldsymbol{\tau}} \\ \dot{\boldsymbol{\tau}}_c \end{pmatrix} \\ &= \frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}}\Big(\boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau}\Big) + \boldsymbol{B}\left(\boldsymbol{A}_\tau\boldsymbol{\tau} + \boldsymbol{B}_\tau\boldsymbol{\tau}_c\right) \ . \end{aligned} \tag{5.56}$$

Die dritte zeitliche Ableitung ergibt schließlich

$$\dddot{\boldsymbol{y}}_\nu = \frac{\partial \ddot{\boldsymbol{y}}_\nu}{\partial \boldsymbol{x}_\nu}\frac{d\boldsymbol{x}_\nu}{dt} = \mathcal{A}(\boldsymbol{x}_\nu) + \mathcal{B}\boldsymbol{u}_\tau \ , \tag{5.57}$$

mit

$$\begin{aligned}\mathcal{A}(\boldsymbol{x}_\nu) = &\left[\frac{\partial}{\partial \boldsymbol{\nu}}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}}\Big(\boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau}\Big)\right)\right](\boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau}) + \ldots \\ &\ldots\left(\frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}_\tau\right)(\boldsymbol{A}_\tau\boldsymbol{\tau} + \boldsymbol{B}_\tau\boldsymbol{\tau}_c) + \boldsymbol{B}\boldsymbol{B}_\tau\boldsymbol{A}_c\boldsymbol{\tau}_c\end{aligned} \tag{5.58}$$

und

$$\mathcal{B} = \boldsymbol{B}\boldsymbol{B}_\tau\boldsymbol{B}_c \ . \tag{5.59}$$

Diese Ableitung hängt von der Eingangsgröße $\boldsymbol{u}_\tau$ ab. Für die nichtlineare Zustandstransformation resultiert

$$\boldsymbol{z} = \begin{pmatrix}\boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \boldsymbol{z}_3\end{pmatrix} = \begin{pmatrix}\boldsymbol{y}_\nu \\ \dot{\boldsymbol{y}}_\nu \\ \ddot{\boldsymbol{y}}_\nu\end{pmatrix} \ . \tag{5.60}$$

Basierend auf dem transformierten Zustandsvektor $\boldsymbol{z}$ lässt sich das System mit

$$\begin{pmatrix}\dot{\boldsymbol{z}}_1 \\ \dot{\boldsymbol{z}}_2 \\ \dot{\boldsymbol{z}}_3\end{pmatrix} = \begin{pmatrix}\boldsymbol{z}_2 \\ \boldsymbol{z}_3 \\ \mathcal{A}(\boldsymbol{x}_\nu)\end{pmatrix} + \begin{pmatrix}\boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} \\ \mathcal{B}\end{pmatrix}\boldsymbol{u}_\tau \tag{5.61}$$

beschreiben. Diese Darstellungsform wird auch als nichtlineare Regelungsnormalform bezeichnet [11]. Der Ausgangsvektor entspricht hierbei $\boldsymbol{y}_v = \boldsymbol{z}_1$. Durch die Wahl der Eingangsgröße mit

$$\boldsymbol{u}_\tau = \mathcal{B}^{-1}\left(-\mathcal{A}(\boldsymbol{x}_\nu) + \boldsymbol{u}_z\right) \tag{5.62}$$

wird der nichtlineare Term $\mathcal{A}(\boldsymbol{x}_\nu)$ kompensiert und es entsteht ein lineares System bezüglich der transformierten Zustandsgrößen mit der neuen Eingangsgröße $\boldsymbol{u}_z$. Die Matrix $\mathcal{B}$ muss regulär und somit invertierbar sein. Anhand von (5.59) ist zu erkennen, dass diese Bedingung erfüllt ist, wenn $\boldsymbol{B}$ ebenfalls invertierbar ist und $\boldsymbol{B}_\tau$ und $\boldsymbol{B}_c$ beispielsweise Diagonalmatrizen wie in (5.6) und (5.43) sind. Gleichzeitig wird für die Berechnung von $\mathcal{A}(\boldsymbol{x}_\nu)$ nach (5.58) die zweite partielle Ableitung von $\boldsymbol{f}(\boldsymbol{\nu})$ nach $\boldsymbol{\nu}$ benötigt. Diese beiden Anforderungen an das dynamische Bewegungsmodell wurden in Abschnitt 5.2 bereits erwähnt. In Appendix A.4 werden die zeitlichen Ableitungen (5.55), (5.56) und (5.57) anhand eines konkreten Beispielmodells berechnet.

5.4.3.3. Posenregelung

Mithilfe der Posenregelung soll die körperfeste Posendifferenz $\Delta\boldsymbol{\eta}^b$ nach (5.46) minimiert werden. Durch die exakte Linearisierung werden Nichtlinearitäten auf Geschwindigkeitsebene mit dem Regelgesetz (5.62) kompensiert. Die neue Eingangsgröße für das linearisierte System ist $\boldsymbol{u}_z$. Der Zusammenhang zwischen $\Delta\boldsymbol{\eta}^b$ und $\boldsymbol{u}_z$ lässt sich aus (5.49), (5.61) sowie (5.62) herleiten und resultiert zu

$$\begin{aligned}\Delta\dot{\boldsymbol{\eta}}^b &= \boldsymbol{Q}^{\mathrm{T}}(\boldsymbol{z}_{1,3})\Delta\boldsymbol{\eta}^b + \boldsymbol{z}_1 - \boldsymbol{T}_b^{n,\mathrm{T}}(\psi)\dot{\boldsymbol{\eta}}_c \\ \dot{\boldsymbol{z}}_1 &= \boldsymbol{z}_2 \\ \dot{\boldsymbol{z}}_2 &= \boldsymbol{z}_3 \\ \dot{\boldsymbol{z}}_3 &= \boldsymbol{u}_z \ ,\end{aligned} \tag{5.63}$$

wobei $\boldsymbol{z}_{1,3} = r$ ist. Die Posendifferenz und die transformierten Zustandsgrößen werden in dem Zustandsvektor

$$\boldsymbol{x}_\eta = \left(\Delta\boldsymbol{\eta}^{b,\mathrm{T}},\ \boldsymbol{z}_1^{\mathrm{T}},\ \boldsymbol{z}_2^{\mathrm{T}},\ \boldsymbol{z}_3^{\mathrm{T}}\right)^{\mathrm{T}} \tag{5.64}$$

zusammengefasst. Für den Entwurf einer linearen Zustandsrückführung wird das System (5.63) zunächst in einem Arbeitspunkt linearisiert. Als Arbeitspunkt wird die letzte Pose des Pfades gewählt, an der das Fahrzeug zum Stehen kommen soll. Der Grund dafür ist, dass beispielsweise das Anlegen mit hoher Präzision erfolgen muss, wohingegen Modellungenauigkeiten während der Fahrt entlang eines Pfades tolerierbar sind. Infolgedessen wird (5.63) in einem Arbeitspunkt linearisiert, der mit

$$\begin{aligned} \Delta\boldsymbol{\eta}^b &= \mathbf{0} \\ \dot{\boldsymbol{\eta}}_c &= \mathbf{0} \\ \boldsymbol{\nu} &= \mathbf{0} \\ \boldsymbol{\tau} &= \mathbf{0} \\ \boldsymbol{\tau}_c &= \mathbf{0} \end{aligned} \tag{5.65}$$

definiert ist. Für die linearisierte Zustandsraumdarstellung resultiert

$$\underbrace{\begin{pmatrix} \Delta\dot{\boldsymbol{\eta}}^b \\ \dot{\boldsymbol{z}}_1 \\ \dot{\boldsymbol{z}}_2 \\ \dot{\boldsymbol{z}}_3 \end{pmatrix}}_{\dot{\boldsymbol{x}}_\eta} = \underbrace{\begin{pmatrix} \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix}}_{\boldsymbol{A}_\eta} \underbrace{\begin{pmatrix} \Delta\boldsymbol{\eta}^b \\ \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \boldsymbol{z}_3 \end{pmatrix}}_{\boldsymbol{x}_\eta} + \underbrace{\begin{pmatrix} \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} \\ \boldsymbol{I}_{3\times3} \end{pmatrix}}_{\boldsymbol{B}_\eta} \boldsymbol{u}_z \ . \tag{5.66}$$

Es ist zu erkennen, dass sich eine Integratorkette aus den einzelnen Zustandsgrößen $\Delta\boldsymbol{\eta}^b$, $\boldsymbol{z}_1$, $\boldsymbol{z}_2$ und $\boldsymbol{z}_3$ bildet. Damit die Posendifferenz zu Null geht, muss der gesamte Zustandsvektor $\boldsymbol{x}_\eta$ für $t \to \infty$ zu Null gehen. Physikalisch lässt sich das damit erklären, dass eine konstante Pose nur dann gehalten werden kann, wenn Geschwindigkeiten, Beschleunigungen und alle weiteren zeitlichen Ableitungen Null sind. Es ist also nicht möglich, gleichzeitig eine konstante Pose mit einer konstante Geschwindigkeit $\boldsymbol{\nu} \neq \mathbf{0}$ beizubehalten. Infolgedessen stellt sich das Regelungsproblem als ein Regulatorproblem dar, bei dem der Zustandsvektor $\boldsymbol{x}_\eta$ zu Null gebracht werden muss. Alle Zustandsgrößen werden während der numerischen Integration des nichtlinearen Systems berechnet und sind somit bekannt. Die transformierten Zustandsgrößen werden mit (5.60) berechnet. Der Rang der Steuerbarkeitsmatrix

$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{B}_\eta & \boldsymbol{A}_\eta\boldsymbol{B}_\eta & \boldsymbol{A}_\eta^2\boldsymbol{B}_\eta & \cdots & \boldsymbol{A}_\eta^{n-1}\boldsymbol{B}_\eta \end{pmatrix}$$

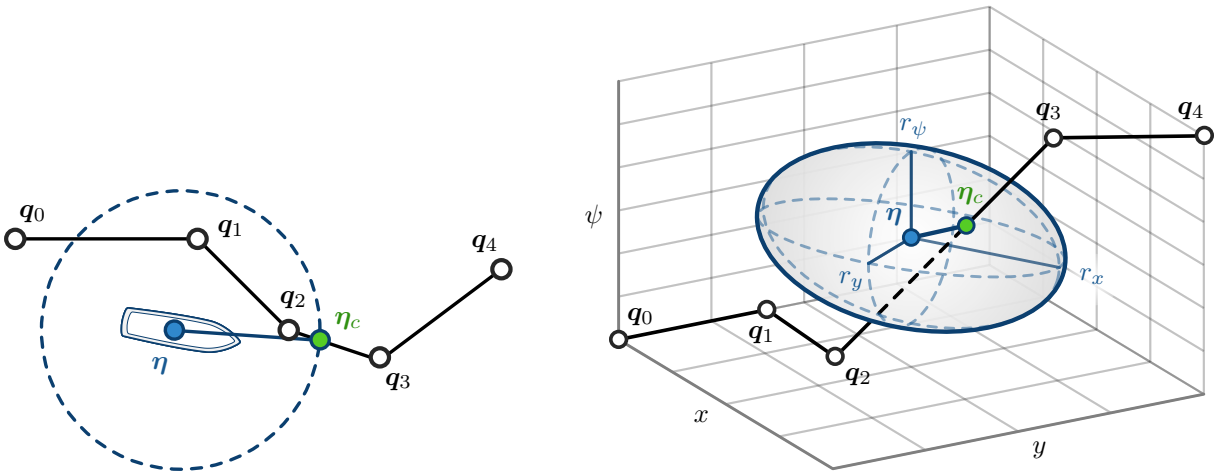
entspricht der Dimension des Zustandsvektors $n = 12$. Demzufolge ist das System steuerbar. Für den Posenregler wird eine Zustandsrückführung mit dem Regelgesetz

$$\boldsymbol{u}_z = -\boldsymbol{K}_\eta \boldsymbol{x}_\eta \tag{5.67}$$

verwendet. Hierbei ist $\boldsymbol{K}_\eta \in \mathbb{R}^{3\times12}$ die Verstärkungsmatrix, deren Parametrisierung mit klassischen Entwurfsmethoden aus der linearen Regelungstheorie wie beispielsweise LQR [34] oder *pole-placement* [112] erfolgt.

5.4.3.4. Guidance-Gesetz

Mithilfe des *Guidance*-Gesetzes erfolgt die Berechnung der Sollpose $\boldsymbol{\eta}_c$ für das Regelsystem auf Basis des gegebenen Pfades $\boldsymbol{\Pi}$ und den gegebenen Hindernissen $\mathcal{O}$. Das *Guidance*-Gesetz basiert auf dem *line-of-sight* (LOS) Verfahren [102]. Bei dem klassischen LOS-Algorithmus wird ein Kreis um das Fahrzeug definiert, sodass ein Schnittpunkt des Kreises mit dem Pfad in $\mathbb{R}^2$ entsteht, wie in Abbildung 5.19a illustriert ist. Prinzipiell entstehen zwei Schnittpunkte, jedoch wird der Schnittpunkt gewählt, welcher dichter am Ziel bzw. am Ende des Pfades liegt. Der Winkel der Sichtlinie vom Fahrzeug zum Schnittpunkt ausgehend von der x^n-Achse wird dann üblicherweise als Sollwert für eine Kursregelung kommandiert.



(a) Klassisches LOS in $\mathbb{R}^2$: $\boldsymbol{\eta}_c$ entspricht Schnittpunkt des Pfades mit einem Kreis.

(b) Modifiziertes LOS für $SE(2)$: $\boldsymbol{\eta}_c$ entspricht Schnittpunkt des Pfades mit einem Ellipsoid.

Abbildung 5.19.: Illustration der Sollposenbestimmung auf Basis des *line-of-sight*-Verfahrens.

In der vorliegenden Arbeit wird der Schnittpunkt direkt als Sollpose verwendet. Der LOS-Algorithmus wird dahingehend erweitert, dass der dichteste Punkt des Kreises zum Pfad gewählt wird, falls kein Schnittpunkt existiert. Da Pfade in $SE(2)$ liegen, entspricht die Sollpose $\boldsymbol{\eta}_c$ dem Schnittpunkt des Pfades mit einem Ellipsoid, wie in Abbildung 5.19b zu erkennen ist. Die Größe des Ellipsoids ist anhand der drei Radien r_x, r_y und r_ψ definiert, wobei r_x in Richtung der x^b-Achse und r_y in Richtung der y^b-Achse des körperfesten Koordinatensystems verläuft. Der maximale Abstand zwischen $\boldsymbol{\eta}$ und $\boldsymbol{\eta}_c$ ist durch diese drei Radien begrenzt. Um den Schnittpunkt des Ellipsoids mit einem Pfadsegment zu berechnen, wird der Konfigurationsraum derart auf einen euklidischen Raum $\mathbb{R}^3$ abgebildet, dass sich die Schnittpunktberechnung auf die einer Geraden mit einer Einheitskugel vereinfacht. Mit

$$\boldsymbol{p}_i = \begin{pmatrix} p_{x,i} \\ p_{y,i} \\ p_{z,i} \end{pmatrix} = \begin{pmatrix} \frac{1}{r_x} & 0 & 0 \\ 0 & \frac{1}{r_y} & 0 \\ 0 & 0 & \frac{1}{r_\psi} \end{pmatrix} \boldsymbol{T}_b^{n,\mathrm{T}}(\psi) \begin{pmatrix} x_i - x \\ y_i - y \\ \sigma\left(\psi_i - \psi\right) \end{pmatrix} \tag{5.68}$$

wird eine beliebige Pose $\boldsymbol{q}_i = (x_i, y_i, \psi_i)^\mathrm{T} \in SE(2)$ auf einen dreidimensionalen euklidischen Raum abgebildet, wobei der Koordinatenursprung identisch mit der aktuellen Pose $\boldsymbol{\eta} = (x, y, \psi)^\mathrm{T}$ ist. Alle Posen des Pfades werden transformiert, sodass eine Liste aus Liniensegmenten in $\mathbb{R}^3$ entsteht. Anschließend wird der Schnittpunkt $\boldsymbol{p}_c$ eines Liniensegmentes mit der Einheitskugel im Koordinatenursprung berechnet. Gibt es mehrere Schnittpunkte, dann wird wie beim LOS-Algorithmus der Schnittpunkt gewählt, welcher dichter am Ziel liegt. Existiert kein Schnittpunkt, dann entspricht $\boldsymbol{p}_c$ dem dichtesten Punkt der Einheitskugel zu allen Liniensegmenten. Falls sich die Endposition des letzten Liniensegmentes innerhalb der Einheitskugel befindet, wird diese Endposition als $\boldsymbol{p}_c$ gewählt. Durch die Invertierung von (5.68) wird aus $\boldsymbol{p}_c$ die Sollpose mit

$$\boldsymbol{\eta}_c = \boldsymbol{\sigma}\left(\boldsymbol{\eta} + \boldsymbol{T}_b^n(\psi) \begin{pmatrix} r_x & 0 & 0 \\ 0 & r_y & 0 \\ 0 & 0 & r_\psi \end{pmatrix} \boldsymbol{p}_c\right) \tag{5.69}$$

berechnet. Die drei Radien r_x, r_y und r_ψ stellen die Parameter für das *Guidance*-Gesetz dar. Je größer die Radien gewählt werden, desto größer ist die Posendifferenz für die Posenregelung. Aufgrund der Zustandsrückführung über die Verstärkungsmatrix $\boldsymbol{K}_\eta$ verhält sich die Stellgröße nach dem Regelgesetz (5.67) proportional zum Zustand $\boldsymbol{x}_\eta$ und somit proportional zur körperfesten Posendifferenz $\Delta\eta^b$. Als Folge führen große Werte für r_x, r_y und r_ψ zu entsprechend großen

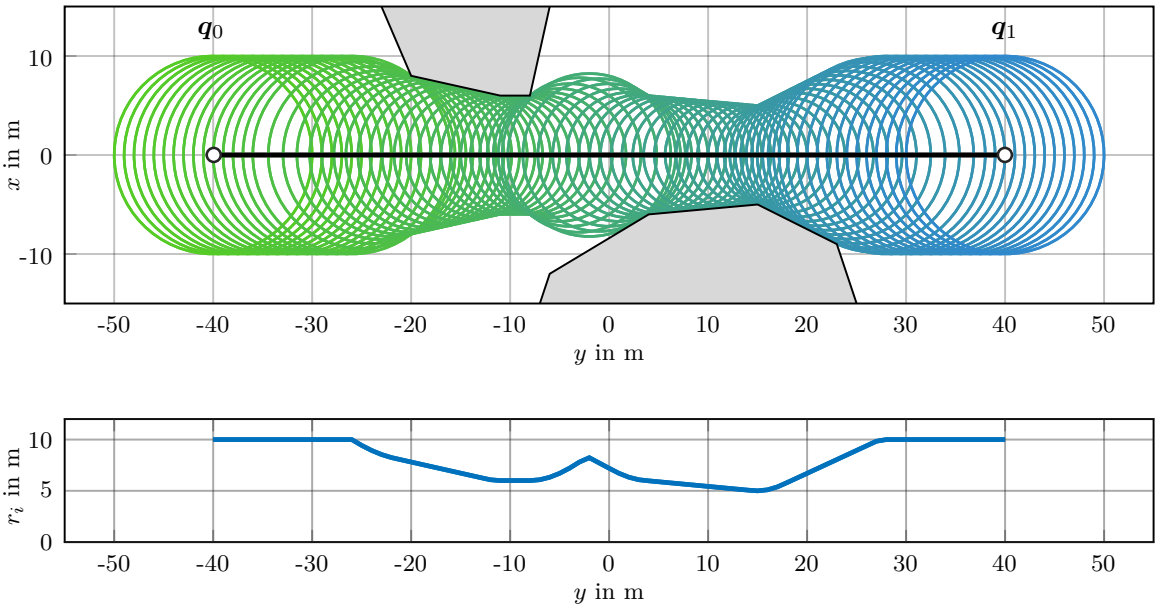


Abbildung 5.20.: Beispiel für die resultierenden Positionsradien r_i mit $i = \{x, y\}$ für den Fall $r_x = r_y$ entlang einer geraden Strecke (schwarz) zwischen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ bei zwei gegebenen Hindernissen (grau) mit $r_{p,min} = 1\,\text{m}$ und $r_{i,max} = 10\,\text{m}$.

Geschwindigkeiten u, v und r. Für ein sicheres Manövrieren in der Nähe von Hindernissen ist es von Vorteil, die Geschwindigkeit mit geringer werdendem Abstand zu Hindernissen zu reduzieren. Um das zu erreichen, werden die Positionsradien r_x und r_y für das *Guidance*-Gesetz mit

$$r_i = \begin{cases} r_{i,max} & \text{falls } \delta(x,y) \geq r_{i,max} \\ \delta(x,y) & \text{falls } r_{p,min} < \delta(x,y) < r_{i,max} \\ r_{p,min} & \text{falls } \delta(x,y) \leq r_{p,min} \end{cases} \tag{5.70}$$

für $i = \{x, y\}$ nach oben und unten beschränkt, wobei $r_{p,min} > 0$ und $r_{i,max} > r_{p,min}$ eine untere bzw. obere Beschränkung für r_i festlegen. Hierbei ist $\delta(x,y)$ die in Abschnitt 5.3.2 bereits eingeführte Funktion, mit deren Hilfe der kleinste Abstand der Position $(x,y)^{\mathrm{T}}$ zu allen Polygonen aus $\mathcal{O}$ bestimmt wird. Das bedeutet, dass die Radien r_x und r_y in Abhängigkeit von der Position des Fahrzeuges sowie den Polygonen aus $\mathcal{O}$ berechnet werden. Der Radius r_ψ bleibt unverändert. Die Tuningparameter für das *Guidance*-Gesetz sind $r_{p,min}$, $r_{x,max}$, $r_{y,max}$ und r_ψ. Die Vorgabe eines minimalen Radius $r_{p,min}$ ist notwendig, da während der Simulation die Positionen der Trajektorien innerhalb von Hindernissen liegen können. In diesem Fall wäre $\delta(x,y) = 0$ und der Positionsfehler würde ebenfalls Null werden, wodurch die Trajektorie nicht die Endpose des Pfades erreichen würde. Für kleine Werte für r_x und r_y ergibt sich eine geringere Geschwindigkeit und es muss für einen längeren Zeitraum simuliert werden. Demzufolge kann $r_{p,min}$ einen signifikanten Einfluss auf die Rechenzeit haben.

In der Abbildung 5.20 ist ein Beispiel für den Fall $r_x = r_y$ dargestellt. In der oberen Grafik ist der Pfad anhand einer geraden schwarzen Linie zwischen zwei Konfigurationen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ abgebildet. Die grau gekennzeichneten Flächen stellen die Hindernisse dar. Entlang des Pfades sind in regelmäßigen Abständen Kreise mit den resultierenden Radien r_i entsprechend der Gleichung (5.70) gezeigt. Hierbei ist $r_{p,min} = 1\,\text{m}$ und $r_{x,max} = r_{y,max} = 10\,\text{m}$. In der unteren Grafik ist der zugehörige Wert für r_i in Abhängigkeit der y-Position dargestellt. Es ist zu erkennen, dass r_i ab etwa $y = -26\,\text{m}$ reduziert wird und im gesamten Verlauf zwischen $r_{p,min}$ und $r_{x,max}$ bzw. $r_{y,max}$ liegt.

In Algorithmus 18 ist das *Guidance*-Gesetz zusammengefasst. In Zeile 2 werden die Radien basierend auf dem Abstand zu Hindernissen nach (5.70) gesetzt. Alle Posen des Pfades $\boldsymbol{\Pi}$ werden in Zeile 3 auf den euklidischen Raum abgebildet. Anschließend wird der dichteste Punkt aus $\boldsymbol{P}$ zum Koordinatenursprung berechnet (Zeile 4) und ein neuer Pfad $\boldsymbol{P}_s$ definiert, der den Ursprung, den dichtesten Punkt sowie alle Punkte aus $\boldsymbol{P}$ enthält, welche die nachfolgenden Punkte des dichtesten Punktes darstellen (Zeile 5). In den Zeilen 7-14 werden alle Liniensegmente aus $\boldsymbol{P}_s$ der Reihe nach auf Schnittpunkte mit der Einheitskugel getestet. Da die Liniensegmente im Ursprung beginnen, gibt es entweder genau einen oder keinen Schnittpunkt. Sobald ein Schnittpunkt eines Liniensegmentes mit der Einheitskugel um den Ursprung existiert (Zeile 9), wird dieser Schnittpunkt für $\boldsymbol{p}_c$ gesetzt (Zeile 10) und die Schleife kann unterbrochen werden. Andernfalls wird in Zeile 13 der Endpunkt $\boldsymbol{p}_i$ des Liniensegmentes der aktuellen Iteration für $\boldsymbol{p}_c$ gewählt und die Iteration wird fortgesetzt. Der gefundene Schnittpunkt wird in Zeile 15 nach (5.69) in den Konfigurationsraum transformiert und entspricht der Sollpose $\boldsymbol{\eta}_c$.

Algorithmus 18: *Guidance*-Gesetz für die *ExplorePath*-Prozedur der Bewegungsplanung

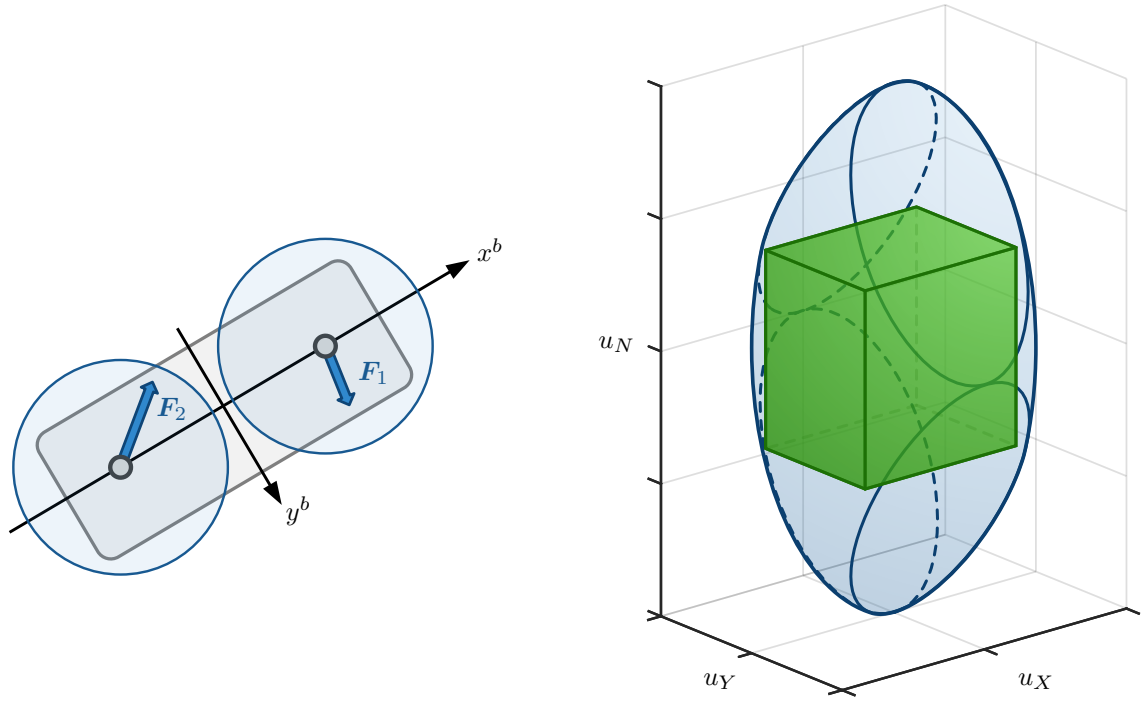
```
1:  procedure η_c ← GuidanceLaw(η, Π, O)
2:      SetRadii(η, O);                                   // according to (5.70)
3:      P ← {TransformToEuclideanSpace(q), ∀q ∈ Π};       // according to (5.68)
4:      p_closest ← ClosestPointToOrigin(P);
5:      P_s ← {0, p_closest, P.Subset(p_closest)};
6:      p_c ← 0;
7:      for i = 2 to P_s.NumPoints() do
8:          ε ← LineSegment(p_{i−1}, p_i);
9:          if IntersectsWithUnitSphere(ε) then
10:             p_c ← IntersectionPointWithUnitSphere(ε);
11:             break
12:         end
13:         p_c ← p_i;
14:     end
15:     η_c ← TransformToCSpace(p_c);                     // according to (5.69)
16: end
```

5.4.3.5. Stellgrößenbeschränkung

Um realisierbare Trajektorien zu generieren, müssen die technischen Beschränkungen des Fahrzeuges berücksichtigt werden. Typischerweise sind die Stellwerte der Antriebsorgane durch Minimal- und Maximalwerte begrenzt. Da im Rahmen der Bewegungsplanung jedoch Kräfte und Momente die Eingangsgrößen bilden, wird die Stellgrößenbeschränkung in der Form

$$\boldsymbol{u}_{\tau,min} \leq \boldsymbol{u}_{\tau} \leq \boldsymbol{u}_{\tau,max} \tag{5.71}$$

umgesetzt. Im Allgemeinen muss eine Untermenge der durch die Antriebsorgane realisierbaren Kräfte und Momente gewählt werden, wie die Abbildung 5.21 an einem Beispiel zeigt. In der linken Grafik ist die Antriebskonfiguration eines ASVs mit zwei drehbaren Antriebsorganen dargestellt, die sich entlang der x^b-Achse befinden und die Antriebskräfte $\boldsymbol{F}_1$ und $\boldsymbol{F}_2$ erzeugen. Die Schubkraft jedes Antriebs ist durch einen Maximalwert beschränkt. Die Gesamtkraft sowie das Moment, was beide Antriebe in Summe auf das Fahrzeug ausüben können, ist in der Abbildung 5.21b durch die blaue Form illustriert. Sie gleicht einem Ellipsoid mit vier ebenen Schnittflächen. Der grüne Quader kennzeichnet die realisierbaren Kräfte und Momente entsprechend der sogenannten *box constraints* nach (5.71) und bildet eine Untermenge der blauen Form. Bei der



(a) Darstellung der Antriebskonfiguration eines ASVs mit zwei drehbaren Antrieben.

(b) Illustration der Menge an realisierbaren Kräften und Momenten (blau) sowie einer beschränkten Eingangsgröße $\boldsymbol{u}_\tau$ (grün).

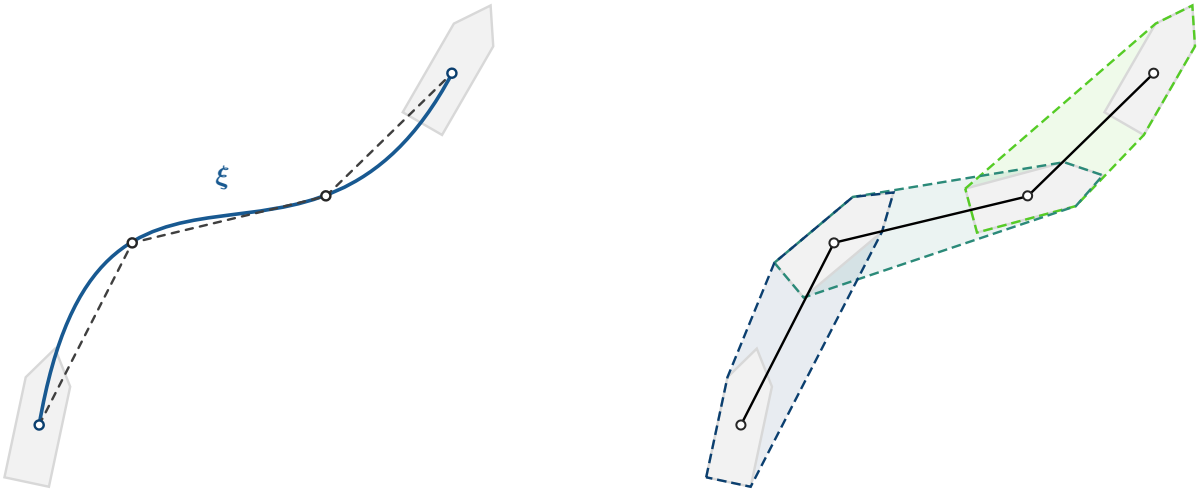
Abbildung 5.21.: Schematische Darstellung der Menge an realisierbaren Kräften und Momenten bei einer Stellgrößenbeschränkung anhand einer gewählten Antriebskonfiguration.

Wahl von $\boldsymbol{u}_{\tau,min} = (X_{min}, Y_{min}, N_{min})^{\mathrm{T}}$ und $\boldsymbol{u}_{\tau,max} = (X_{max}, Y_{max}, N_{max})^{\mathrm{T}}$ muss darauf geachtet werden, dass eine Menge an Kräften und Momenten entsteht, die mit den Antriebsorgangen entsprechend der spezifischen Antriebskonfiguration realisierbar ist. Darüber hinaus muss der Bereich der Beschränkung für die Bewegungsplanung weiter reduziert werden, damit eine Stellgrößenreserve für das *Control-System* des realen Fahrzeuges bleibt. Solange die Eingangsgröße in einem unbeschränkten Bereich liegt, ist die Stabilität des geschlossenen Regelkreises durch den Reglerentwurf sichergestellt. In dieser Arbeit wird auf eine Stabilitätsuntersuchung für den Fall einer beschränkten Eingangsgröße verzichtet. Da die vorgestellte Reglerstruktur keinen Integralanteil enthält, sind keine weiteren Maßnahmen wie beispielsweise die Entwicklung einer Anti-Windup-Strategie notwendig.

5.4.4. IsFeasible-Funktion

Die *IsFeasible*-Prozedur der Bewegungsplanung ist in Algorithmus 19 abgebildet. In Zeile 2 wird der zu überprüfende Zweig $\boldsymbol{\Pi}_{branch}$ mithilfe der *ExplorePath*-Prozedur aus Algorithmus 17 in eine Trajektorie $\boldsymbol{\xi}$ transformiert. Die Anzahl der numerischen Integrationsschritte, die in der Trajektorie enthalten sind, ist mit k gekennzeichnet. Falls in der *ExplorePath*-Prozedur das Zielgebiet nicht innerhalb von k_{max} Integrationsschritten erreicht werden konnte (Zeile 3), wird die Trajektorie als ungültig betrachtet, da nicht sichergestellt werden kann, dass sich der Endzustand der Trajektorie in einem sicheren Zustand mit einer Geschwindigkeit von $\boldsymbol{\nu} \approx \mathbf{0}$ und einer Kraft $\boldsymbol{\tau} \approx \mathbf{0}$ befindet. In Zeile 6 wird die Trajektorie $\boldsymbol{\xi}$ auf Kollisionen mit $\mathcal{O}$ getestet.

Der Algorithmus für den Kollisionstest entlang einer Kurve in $SE(2)$ ist in Algorithmus 20 beschrieben und in Abbildung 5.22 schematisch dargestellt. Zunächst wird die Trajektorie mit-



(a) Approximation der Trajektorie $\boldsymbol{\xi}$ (blau) durch mehrere Liniensegmente (gestrichelte Linie).

(b) Separater Kollisionstest für jedes Liniensegment.

Abbildung 5.22.: Schematische Darstellung der *CheckCollisionCurve*-Prozedur.

Algorithmus 19: *IsFeasible*-Prozedur der Bewegungsplanung

```
1: procedure feasible ← IsFeasible(Π_branch, O)
2:     (ξ, k) ← ExplorePath(x_0, τ_{c,0}, Π_branch, O);        // see algorithm 17
3:     if k ≥ k_max then
4:         return FALSE;
5:     end
6:     return ¬CheckCollisionCurve(ξ, O);                      // see algorithm 20
7: end
```

hilfe des Douglas-Peucker-Algorithmus [113] vereinfacht (Zeile 2). Dabei wird die Kurve, welche durch die Posen der Trajektorie gegeben ist, durch mehrere Liniensegmente approximiert, wobei die maximale Abweichung der Approximation von der Kurve durch zwei Parameter Δp_{max} und $\Delta\psi_{max}$ festgelegt ist, welche die maximale Positions- bzw. Winkelabweichung bezeichnen. In Zeile 4 erfolgt der Kollisionstest für jedes Liniensegment nach Algorithmus 10. Innerhalb der Bewegungsplanung muss ein größeres Fahrzeugpolygon verwendet werden, um einen sicheren Kollisionstest bei einer gewählten Approximation auf Basis der zwei Parameter Δp_{max} und $\Delta\psi_{max}$ gewährleisten zu können. Auf die Wahl eines größeren Fahrzeugpolygons wird detailliert in Appendix A.3 eingegangen.

Algorithmus 20: *CheckCollisionCurve*-Prozedur der Bewegungsplanung

```
1: procedure collision ← CheckCollisionCurve(ξ, O)
2:     Π_approx ← DouglasPeuckerSE2(ξ, Δp_max, Δψ_max);
3:     for i = 2 to Π_approx.NumPoses() do
4:         if CheckCollision(q_{i−1}, q_i, O) then
5:             return TRUE;
6:         end
7:     end
8:     return FALSE;
9: end
```

5.4.5. Rewire

In der vorgestellten Methode zur Bewegungsplanung von ASVs wird die Dynamik innerhalb der *IsFeasible*-Prozedur berücksichtigt. Für einen gegebenen Anfangszustand wird eine Trajektorie entlang eines Zweiges des Baums generiert, welche anschließend auf Kollision getestet wird. Innerhalb des RRT*-Algorithmus wird eine Kante zwischen zwei Knoten des Baums neu verbunden, falls die resultierende Kante gültig ist und sich dadurch geringere Gesamtkosten ergeben. Das hat zur Folge, dass alle Teilzweige, welche vom *rewire* einer Kante betroffen sind, nochmals auf Gültigkeit untersucht werden müssen. Aus diesem Grund wird die *Rewire*-Prozedur für die Bewegungsplanung modifiziert.

In Algorithmus 21 ist die angepasste *Rewire*-Prozedur zusammengefasst, wobei die Zeilen 6-14 die Modifikation darstellen. Eine neue Verbindung von $\boldsymbol{q}_{new}$ zu einem Knoten $\boldsymbol{q} \in V_r$ ist dann gültig, wenn alle Zweige vom Wurzelknoten zu allen Blattknoten von $\boldsymbol{q}$ gültig sind. In Zeile 7 wird der Zweig $\boldsymbol{\Pi}_A$ vom Wurzelknoten zu $\boldsymbol{q}_{new}$ ermittelt und in Zeile 8 werden alle Blattknoten von $\boldsymbol{q}$ bestimmt und in der Menge $\boldsymbol{Q}_{leaves}$ zusammengefasst. Anschließend wird für jeden Blattknoten $\boldsymbol{q}_{leaf}$ der Zweig $\boldsymbol{\Pi}_B$ von $\boldsymbol{q}$ zu diesem Blattknoten erzeugt (Zeile 10). Der gesamte Zweig $\boldsymbol{\Pi}_{branch}$ vom Wurzelknoten bis zum Blattknoten über die potenziell neue Verbindung von $\boldsymbol{q}_{new}$ zu $\boldsymbol{q}$ wird in Zeile 12 auf Gültigkeit getestet. Erst wenn alle betroffenen Zweige gültig sind (Zeile 14), erfolgt in den Zeilen 15 und 16 die Aktualisierung der Kosten sowie die Umstrukturierung des Baums.

Algorithmus 21: *Rewire*-Prozedur der Bewegungsplanung

```
1:  procedure Rewire(𝒯, V_r, q_new)
2:      foreach q ∈ V_r do
3:          c_i ← Cost(q_new) + c(q_new, q);
4:          (q_s, ε) ← Steer(q_new, q);                          // see algorithm 8
5:          if q_s = q ∧ c_i < Cost(q) then
6:              feasible ← TRUE;
7:              Π_A ← 𝒯.GetBranchToRoot(q_new);
8:              Q_leaves ← 𝒯.GetAllLeafNodes(q);
9:              foreach q_leaf ∈ Q_leaves do
10:                 Π_B ← 𝒯.GetBranchToNode(q, q_leaf);
11:                 Π_branch ← {Π_A, Π_B};
12:                 feasible ← feasible ∧ IsFeasible(Π_branch, 𝒪);   // see algorithm 19
13:             end
14:             if feasible then
15:                 Δc_i ← c_i − Cost(q);
16:                 𝒯.Reconnect(q, q_new, Δc_i);
17:             end
18:         end
19:     end
20: end
```

5.5. Online-Algorithmus für die sequenzielle Bewegungsplanung

Die sequenzielle Bewegungsplanung ergibt sich aus der Kombination der Pfadplanung aus Abschnitt 5.3 mit der Bewegungsplanung aus Abschnitt 5.4 und ist im Algorithmus 22 dargestellt. Der Algorithmus berechnet eine Trajektorie $\boldsymbol{\xi}$ für einen gegebenen Anfangszustand $\boldsymbol{x}_I = (\boldsymbol{\eta}^{\mathrm{T}}, \boldsymbol{\nu}^{\mathrm{T}}, \boldsymbol{\tau}^{\mathrm{T}})^{\mathrm{T}}$, eine initiale Stellgröße $\boldsymbol{\tau}_{c,I}$, eine vorgegebene Zielpose $\boldsymbol{q}_G$ sowie Hindernisse

Algorithmus 22: Sequenzielle Bewegungsplanung

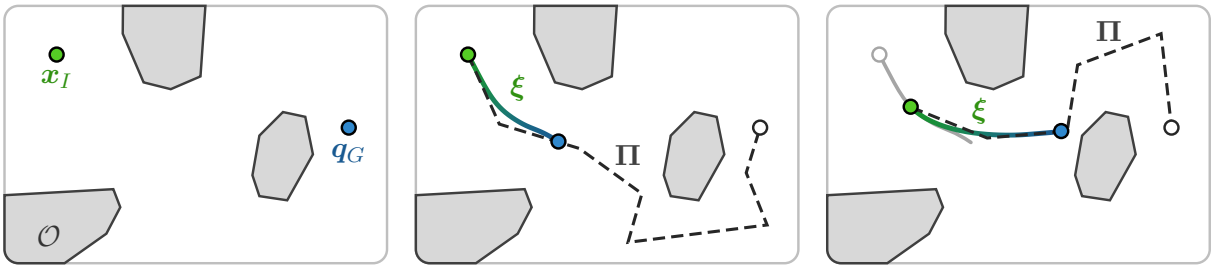
1: **procedure** $\boldsymbol{\xi} \leftarrow$ SequentialPlanner$(\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p}, t^p_{max}, t^m_{max})$
2: $\quad \boldsymbol{q}_I \leftarrow$ Pose$(\boldsymbol{x}_I)$;
3: $\quad \boldsymbol{\Pi} \leftarrow$ PathPlanner$(\boldsymbol{q}_I, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p}, t^p_{max})$; `// see algorithm 5`
4: $\quad \boldsymbol{\xi} \leftarrow$ MotionPlanner$(\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}, \boldsymbol{\Pi}, \mathcal{O}, \Delta\boldsymbol{p}, t^m_{max})$; `// see algorithm 13`
5: **end**

$\mathcal{O}$. Die initiale Konfiguration $\boldsymbol{q}_I$ entspricht der Pose von $\boldsymbol{x}_I$. In Zeile 3 wird mithilfe der Pfadplanung nach Algorithmus 5 der Pfad $\boldsymbol{\Pi}$ berechnet, welcher in Zeile 4 für die Bewegungsplanung nach Algorithmus 13 genutzt wird, um $\boldsymbol{\xi}$ zu berechnen. Anhand der zwei Parameter t^p_{max} und t^m_{max} wird eine Beschränkung der Rechenzeit für die Pfad- bzw. Bewegungsplanung definiert. Die Gesamtrechenzeit der sequenziellen Bewegungsplanung ist demzufolge mindestens die Summe dieser zwei Zeiten.

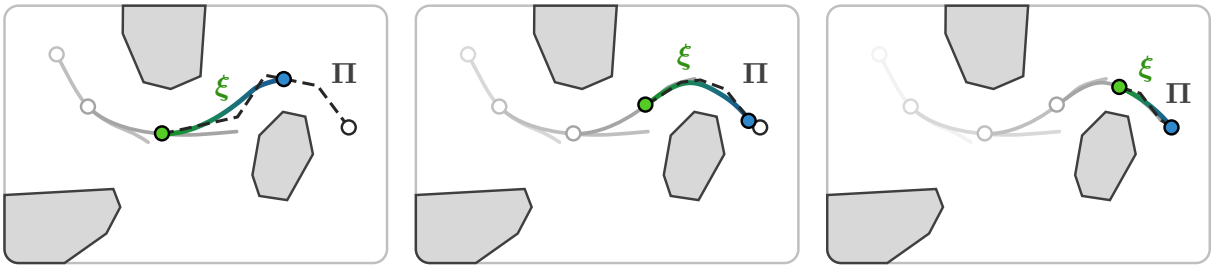
Mit Algorithmus 22 wird ein einziges Planungsproblem mit gegebenen Eingangsdaten in begrenzter Rechenzeit gelöst. Jedoch verändern sich die Eingangsdaten zur Laufzeit, da beispielsweise mit einer Umfelderkennung neue Hindernisse detektiert werden oder die vorgegebene Zielpose von einem übergeordneten System angepasst wird. Für den praktischen Einsatz muss die sequenzielle Bewegungsplanung daher fortlaufend ausgeführt werden. Zudem ist eine Neuberechnung notwendig, da für die Bewegungsplanung nur ein Teil des gefundenen Pfades der Pfadplanung verwendet wird und die resultierende Trajektorie in der Endpose dieses Teilpfades endet. Um die vorgegebene Zielpose zu erreichen, muss die Bewegungsplanung mit neuen, in der Zukunft liegenden Anfangswerten basierend auf der jeweils zuvor generierten Trajektorie ausgeführt werden. Durch diese fortwährende Berechnung ergibt sich darüber hinaus der Vorteil, dass ein zuvor gefundener Pfad mithilfe der vorgestellten *WarmStart*-Prozedur kontinuierlich verbessert wird.

In dieser Arbeit wird ein Online-Algorithmus für die sequenzielle Bewegungsplanung vorgestellt, mit dem die genannten Punkte beachtet werden. Die Berechnung einer Trajektorie nach Algorithmus 22 benötigt eine nicht zu vernachlässigende Rechenzeit. Während dieser Zeit bewegt sich das reale Fahrzeug, sodass der aktuelle Bewegungszustand nach einer abgeschlossenen Berechnung der Trajektorie nicht mehr mit dem Anfangszustand dieser geplanten Trajektorie übereinstimmt. Aus diesem Grund erfolgt die Generierung einer Trajektorie nicht ausschließlich auf Basis des aktuellen Bewegungszustands. Die grundsätzliche Idee des entwickelten Verfahrens ist, dass für die Planung einer neuen Trajektorie ein Anfangszustand gewählt wird, der einem zukünftigen Zustand der aktuellen Trajektorie entspricht. Während der Berechnungszeit, die für das Planen einer neuen Trajektorie notwendig ist, wird vom *Control-System* des ASVs weiterhin die aktuelle Trajektorie verwendet. Ist die Planung der neuen Trajektorie abgeschlossen, wird diese an die aktuelle Trajektorie angefügt und dem *Control-System* übergeben. Anschließend erfolgt die Planung der nächsten Trajektorie ausgehend von einem zukünftigen Zeitpunkt.

Dieses Prinzip der fortlaufenden Berechnung ist in der Abbildung 5.23 schematisch dargestellt. In der Teilabbildung 5.23a ist das Ausgangsproblem gezeigt. Während der abgebildeten Iterationen bleiben $\boldsymbol{q}_G$ und $\mathcal{O}$ konstant. Jede Iteration zeigt den resultierenden Pfad $\boldsymbol{\Pi}$ der Pfadplanung als eine gestrichelte Linie sowie die resultierende Trajektorie $\boldsymbol{\xi}$ der Bewegungsplanung in Form einer grün-blauen Kurve. Die Trajektorien werden nur für einen Teilpfad geplant und enden in den blauen Punkten. Die erste Iteration produziert in diesem Beispiel aufgrund der begrenzten Rechenzeit einen Pfad, der signifikante Abweichungen zu einer optimalen Lösung aufweist, wie in Abbildung 5.23b illustriert ist. Während sich das Fahrzeug entlang des ersten Teilabschnittes der geplanten Trajektorie bewegt, berechnet der Online-Algorithmus eine Lösung für den zweiten Iterationsschritt, welcher in der Abbildung 5.23c dargestellt ist. Es ist zu erkennen, dass die sequenzielle Bewegungsplanung an dem grünen Punkt auf der vorangegangenen Trajektorie beginnt. Der resultierende Pfad stellt in dieser Iteration ($k = 2$) bereits eine bessere Lösung dar.



(a) Gegebenes Ausgangsproblem. (b) Ergebnis der Iteration $k = 1$. (c) Ergebnis der Iteration $k = 2$.



(d) Ergebnis der Iteration $k = 3$. (e) Ergebnis der Iteration $k = 4$. (f) Ergebnis der Iteration $k = 5$.

Abbildung 5.23.: Schematische Darstellung der resultierenden Pfade $\boldsymbol{\Pi}$ (gestrichelte Linien) sowie Trajektorien $\boldsymbol{\xi}$ (grün-blaue Kurven) für mehrere Iterationen des Online-Algorithmus der sequenziellen Bewegungsplanung.

Dieses Prinzip setzt sich in den weiteren Teilabbildungen mit den Ergebnissen der Iterationen $k = \{3, 4, 5\}$ fort. Der Pfad wird kontinuierlich verbessert. Mit der Iteration $k = 5$ erreicht die Trajektorie schließlich die vorgegebene Zielkonfiguration.

Im Algorithmus 23 ist die Vorgehensweise detailliert beschrieben. Die aktuelle Trajektorie wird in $\boldsymbol{\xi}$ gespeichert. Der zugehörige Startzeitpunkt von $\boldsymbol{\xi}$ wird mit t_ξ bezeichnet und $\boldsymbol{\Omega} = (\varphi, \lambda, h)^{\mathrm{T}}$ enthält den geografischen Ursprung der Tangentialebene auf dem Erdellipsoid, in der sich $\boldsymbol{\xi}$ befindet. Innerhalb der Prozedur werden fortlaufend Iterationen durchgeführt (Zeile 2). In Zeile 3 werden die aktuellen Eingangsdaten gesetzt, die beispielsweise vom *Navigation-System* des ASVs kommen. Diese Eingangsdaten beziehen sich ebenfalls auf eine bestimmte Tangentialebene, die sich im Ursprung $\boldsymbol{\Omega}_{new}$ befindet. In Zeile 4 wird die Positionsdifferenz vom aktuellen Ursprung

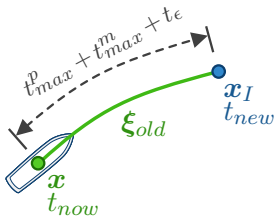
Algorithmus 23: Online-Algorithmus der sequenziellen Bewegungsplanung

Data: $t_\xi \leftarrow 0$; $\boldsymbol{\xi} \leftarrow \varnothing$; $\boldsymbol{\Omega} \leftarrow \mathbf{0}$; $reset \leftarrow$ TRUE

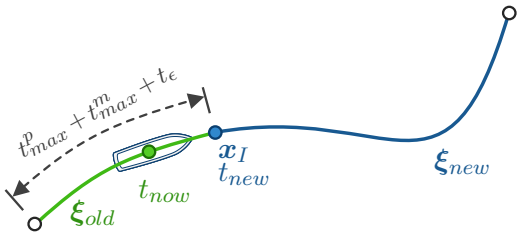
```
1:  procedure RealtimePlanner
2:      for k = 1 to ∞ do
3:          (x, τ_c, q_G, O, Ω_new) ← GetInputData();
4:          Δp ←PositionDifference(Ω_new, Ω);
5:          Ω ← Ω_new;
6:          if reset then
7:              (t_ξ, ξ) ← SolveInitialProblem(x, τ_c, q_G, O, Δp);      // see algorithm 24
8:              reset ← FALSE;
9:          else
10:             (t_ξ, ξ) ← SolveNextProblem(t_ξ, ξ, q_G, O, Δp);         // see algorithm 25
11:         end
12:         Publish(t_ξ, ξ, Ω);
13:     end
14: end
```

Algorithmus 24: *SolveInitialProblem*-Prozedur des Online-Algorithmus

1: **procedure** $(t_\xi, \boldsymbol{\xi}) \leftarrow$ SolveInitialProblem$(\boldsymbol{x}, \boldsymbol{\tau}_c, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p})$
2: $\quad t_{now} \leftarrow$ TimeNow();
3: $\quad t_{new} \leftarrow t_{now} + t^p_{max} + t^m_{max} + t_\epsilon$;
4: $\quad \boldsymbol{\xi}_{old} \leftarrow$ PredictMotion$(\boldsymbol{x}, \boldsymbol{\tau}_c, t_{new})$;
5: $\quad (\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}) \leftarrow \boldsymbol{\xi}_{old}(t_{new})$;
6: $\quad \boldsymbol{\xi}_{new} \leftarrow$ SequentialPlanner$(\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p}, t^p_{max}, t^m_{max})$; `// see algorithm 22`
7: $\quad t_{now} \leftarrow$ TimeNow();
8: $\quad \boldsymbol{\xi} \leftarrow \{\boldsymbol{\xi}_{old}([t_{now}, t_{new}]), \boldsymbol{\xi}_{new}\}$;
9: $\quad t_\xi \leftarrow t_{now}$;
10: **end**



(a) Berechnung des Anfangszustandes auf Basis der Bewegungsprädiktion (Zeile 5).



(b) Zusammenfügen der zwei Trajektorien $\boldsymbol{\xi}_{old}$ und $\boldsymbol{\xi}_{new}$ (Zeile 8).

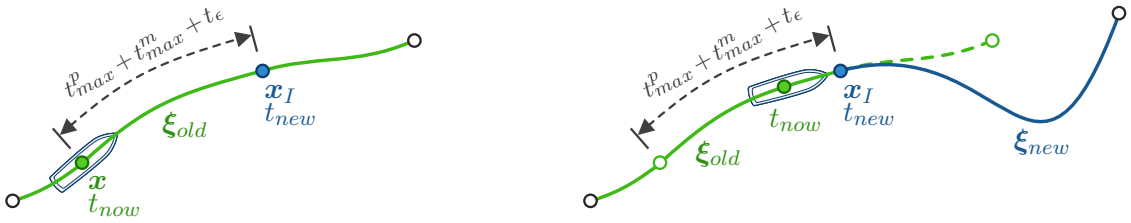
Abbildung 5.24.: Schematische Darstellung der *SolveInitialProblem*-Prozedur zu zwei Zeitpunkten vor und nach der sequenziellen Bewegungsplanung.

$\boldsymbol{\Omega}$ zu $\boldsymbol{\Omega}_{new}$ ermittelt, wobei lediglich die horizontale Positionsdifferenz von Interesse ist. Die Berechnung der Positionsdifferenz zwischen zwei geografischen Koordinaten kann der Literatur entnommen werden [13, 114]. Anschließend wird in Zeile 7 das initiale Planungsproblem mithilfe der *SolveInitialProblem*-Prozedur gelöst, falls das *reset*-Flag gesetzt ist (Zeile 6). Dieses Flag ist initial gesetzt und kann darüber hinaus jederzeit gesetzt werden, um einen Neustart zu erzwingen, sodass die Planung basierend auf dem aktuellen Bewegungszustand $\boldsymbol{x}$ gestartet wird. Das initiale Planungsproblem entspricht dem ersten Planungsproblem nach einem Neustart. Für alle weiteren Iterationen, bei denen das *reset*-Flag nicht gesetzt ist, werden die Planungsprobleme auf Basis der vorangegangenen Trajektorie mithilfe der *SolveNextProblem*-Prozedur gelöst (Zeile 10). Am Ende jeder Iteration wird die berechnete Trajektorie in Zeile 12 dem *Control-System* des ASVs zur Verfügung gestellt.

In Algorithmus 24 ist die *SolveInitialProblem*-Prozedur dargestellt. Zu Beginn wird in Zeile 2 die aktuelle Zeit t_{now} ermittelt. Anschließend wird in Zeile 3 die Zeit t_{new} berechnet, welche den Startzeitpunkt angibt, für den die sequenzielle Bewegungsplanung durchgeführt werden soll. Diese Zeit liegt in der Zukunft und setzt sich aus den gegebenen Rechenzeiten der Pfad- bzw. Bewegungsplanung t^p_{max} und t^m_{max} sowie einer weiteren Zeit t_ϵ zusammen. Hierbei ist $t_\epsilon > 0$ ein zusätzliches Zeitintervall, welches so gewählt werden muss, damit die Summe $t^p_{max} + t^m_{max} + t_\epsilon$ größer ist, als die tatsächliche Rechenzeit der sequenziellen Bewegungsplanung. Das Zeitintervall t_ϵ ist aus rein praktischen Gründen notwendig, da die Pfadplanung und die Bewegungsplanung erst abgebrochen werden, wenn die gemessene Rechenzeit die angegebenen Grenzwerte überschreitet. Die Berechnung einer Trajektorie muss jedoch vor dem Zeitpunkt t_{new} abgeschlossen sein. In Zeile 4 wird die Bewegung des ASVs nach dem Modell (5.1)-(5.3) vom aktuellen Anfangszustand bis zum zukünftigen Zeitpunkt t_{new} prädiziert und die sich daraus ergebene Trajektorie wird in $\boldsymbol{\xi}_{old}$ gespeichert. In der Abbildung 5.24a ist die prädizierte Trajektorie illustiert. Wäh-

Algorithmus 25: *SolveNextProblem*-Prozedur des Online-Algorithmus

1: **procedure** $(t_\xi, \boldsymbol{\xi}) \leftarrow$ SolveNextProblem$(t_\xi, \boldsymbol{\xi}, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p})$
2: $\quad t_{now} \leftarrow$ TimeNow();
3: $\quad t_{new} \leftarrow t_{now} + t^p_{max} + t^m_{max} + t_\epsilon$;
4: $\quad \boldsymbol{\xi}_{old} \leftarrow$ TransformToCurrentFrame$(\boldsymbol{\xi}, \Delta\boldsymbol{p})$; // according to (5.72)
5: $\quad (\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}) \leftarrow \boldsymbol{\xi}_{old}(t_{new})$;
6: $\quad \boldsymbol{\xi}_{new} \leftarrow$ SequentialPlanner$(\boldsymbol{x}_I, \boldsymbol{\tau}_{c,I}, \boldsymbol{q}_G, \mathcal{O}, \Delta\boldsymbol{p}, t^p_{max}, t^m_{max})$; // see algorithm 22
7: $\quad t_{now} \leftarrow$ TimeNow();
8: $\quad \boldsymbol{\xi} \leftarrow \{\boldsymbol{\xi}_{old}([t_{now}, t_{new}]), \boldsymbol{\xi}_{new}\}$;
9: $\quad t_\xi \leftarrow t_{now}$;
10: **end**



(a) Berechnung des Anfangszustandes auf Basis der Trajektorie $\boldsymbol{\xi}_{old}$ (Zeile 5).

(b) Zusammenfügen der zwei Trajektorien $\boldsymbol{\xi}_{old}$ und $\boldsymbol{\xi}_{new}$ (Zeile 7).

Abbildung 5.25.: Schematische Darstellung der *SolveNextProblem*-Prozedur zu zwei Zeitpunkten vor und nach der sequenziellen Bewegungsplanung.

rend der Prädiktion wird angenommen, dass $\boldsymbol{\tau}_c = const.$ gilt. In Zeile 5 wird $\boldsymbol{x}_I$ sowie $\boldsymbol{\tau}_{c,I}$ aus der prädizierten Trajektorie zum Zeitpunkt t_{new} ermittelt. Ausgehend von diesen Anfangswerten, die in Abbildung 5.24 durch einen blauen Punkt gekennzeichnet sind, erfolgt in Zeile 6 die sequenzielle Bewegungsplanung nach Algorithmus 22. Das Ergebnis ist eine Trajektorie $\boldsymbol{\xi}_{new}$, die zum Zeitpunkt t_{new} startet, wie in Abbildung 5.24b dargestellt ist. Anschließend wird t_{now} aktualisiert (Zeile 7). In Zeile 8 werden $\boldsymbol{\xi}_{old}$ und $\boldsymbol{\xi}_{new}$ zusammengefügt, wobei aus $\boldsymbol{\xi}_{old}$ nur das Zeitintervall $[t_{now}, t_{new}]$ verwendet wird. Hierbei wird sich zunutze gemacht, dass eine Trajektorie per Definition nach (2.4) nicht den Anfangszustand enthält. Dadurch lassen sich $\boldsymbol{\xi}_{old}$ und $\boldsymbol{\xi}_{new}$ nahtlos miteinander verbinden, ohne dass $\boldsymbol{x}$ und $\boldsymbol{\tau}_c$ zum Zeitpunkt t_{new} mehrfach enthalten sind. Der Startzeitpunkt der Trajektorie wird in Zeile 9 gesetzt und entspricht der aktuellen Zeit t_{now}.

Die *SolveNextProblem*-Prozedur ist in Algorithmus 25 beschrieben und eine schematische Darstellung ist in der Abbildung 5.25 gezeigt. Die grundsätzliche Vorgehensweise ist dabei vergleichbar mit der der *SolveInitialProblem*-Prozedur. Der Unterschied besteht in der Bestimmung der Anfangswerte $\boldsymbol{x}_I$ und $\boldsymbol{\tau}_{c,I}$ für die sequenzielle Bewegungsplanung. In Zeile 4 wird die aktuelle Trajektorie mit der Positionsdifferenz $\Delta\boldsymbol{p} = (\Delta x, \Delta y)^\mathrm{T}$ mit

$$\begin{aligned} x_{old,i} &= x_i - \Delta x \\ y_{old,i} &= y_i - \Delta y \end{aligned} \tag{5.72}$$

für alle $\boldsymbol{x}_i \in \boldsymbol{\xi}$ in das neue Koordinatensystem transformiert. Die Anfangswerte $\boldsymbol{x}_I$ und $\boldsymbol{\tau}_{c,I}$ werden in Zeile 5 aus Trajektorie $\boldsymbol{\xi}_{old}$ zum Zeitpunkt t_{new} ermittelt, wie in Abbildung 5.25a illustriert ist. Ausgehend von diesen Anfangswerten wird in Zeile 6 das Problem der sequenziellen Bewegungsplanung gelöst. Die Zeilen 7-9 sind identisch zu denen aus Algorithmus 24.

In der Abbildung 5.25b ist das Zusammenfügen von Trajektorien grafisch dargestellt. Die grün gestrichelte Linie entspricht dem Teil der Trajektorie $\boldsymbol{\xi}_{old}$, der verworfen wird und durch die blau markierte neu geplante Trajektorie $\boldsymbol{\xi}_{new}$ ersetzt wird.

6. Simulative Verfahrensanalyse

In diesem Kapitel wird das entwickelte Verfahren zur sequenziellen Bewegungsplanung autonomer Wasserfahrzeuge in mehreren Simulationsexperimenten analysiert. Nach der Beschreibung der Testumgebung, in welcher die Simulationen durchgeführt werden, wird die sequenzielle Bewegungsplanung für ein Beispielszenario mit einem festgelegten Parametersatz ausgeführt. Im Anschluss wird der Einfluss verschiedener Tuningparameter auf die resultierende Lösung der Pfad- und Bewegungsplanung untersucht. Für die Darstellung von Parameterwerten wird im weiteren Verlauf dieser Arbeit auf die Angabe von Einheiten verzichtet. Alle Zahlenwerte beziehen sich grundsätzlich auf das internationale Einheitensystem, sofern nicht anders gekennzeichnet.

6.1. Beschreibung der Simulationsumgebung

Für sämtliche Simulationsexperimente wird das ASV des deutschen Forschungsprojektes A-SWARM (Autonome elektrische Schifffahrt auf Wasserstraßen in Metropolregionen) [115] verwendet, welches in Abbildung 6.1 gezeigt ist. Im Nachfolgenden werden die fahrzeugspezifischen Eigenschaften beschrieben, die für die Bewegungsplanung von Bedeutung sind. Neben der geometrischen Form des Fahrzeuges wird ein konkretes Bewegungsmodell vorgestellt und auf die Parametrisierung der *ExplorePath*-Prozedur eingegangen.

6.1.1. Fahrzeugform

Das ASV hat eine rechteckige Grundform mit einer Länge von 6.0 m und einer Breite von 2.5 m. Wie in den Abschnitten 5.3.5 und 5.4.4 beschrieben wurde, muss die geometrische Fahrzeugform für einen sicheren Kollisionstest innerhalb der Pfad- und Bewegungsplanung größer als die reale geometrische Form sein. Während des Kollisionstests wird eine beliebige Kurve durch zahlreiche Liniensegmente approximiert, wobei der maximale Positionsfehler der Approximation mit Δp_{max} und der maximale Approximationsfehler für den Heading-Winkel mit $\Delta\psi_{max}$ festgelegt wird. Im Rahmen der folgenden Untersuchungen wird $\Delta p_{max} = 0.1$ und $\Delta\psi_{max} = 5^\circ$ gewählt. Nach der in Appendix A.3.2 beschriebenen Vorgehensweise wird das Fahrzeugpolygon als ein Rechteck mit der Länge von 6.5 m und Breite von 3.3 m repräsentiert. Der Ursprung des körperfesten Koordinatensystems befindet sich im Mittelpunkt des Rechteckes.

6.1.2. Bewegungsmodell

Das ASV verfügt über zwei um die Hochachse verdrehbare Pod-Antriebe, deren Propeller so ausgelegt sind, dass der Betrag der Schubkraft bei gleicher Drehzahl in beiden Drehrichtungen identisch ist. Die Antriebskonfiguration entspricht der aus Abbildung 5.21a und der Bereich der realisierbaren Kräfte hat die in Abbildung 5.21b gezeigte Form. Ein einzelner Pod-Antrieb produziert eine Schubkraft von maximal 500 N. Die maximale Stellgröße auf Kraft-Momente-Ebene liegt bei $X_{max} = 700$, $Y_{max} = 550$ und $N_{max} = 750$. Für die Bewegungsplanung werden jedoch nur 90% der maximalen Stellgröße benutzt, damit das *Control-System* des Fahrzeuges die Möglichkeit hat, externe Störungen und Modellungenauigkeiten auszuregeln. Dementsprechend wird die maximale Eingangsgröße mit $X_{max} = 630$, $Y_{max} = 495$ und $N_{max} = 675$ festgelegt.

Abbildung 6.1.: ASV des A-SWARM-Projektes.

Parameter	Wert
$\boldsymbol{F}$	$\begin{pmatrix} -31.61 & 0 & 0 & 233.74 & 0 & 0 & 133.46 & -20.79 & 0 & 0 \\ 0 & -41.71 & 17.24 & 0 & -251.24 & -0.48 & 5.3 & 0 & -167.98 & 497.44 \\ 0 & 0.018 & -23.53 & 0 & 0.11 & 0.65 & -0.0023 & 0 & 0.0724 & -678.80 \end{pmatrix} \times 10^{-3}$
$\boldsymbol{B}$	$\begin{pmatrix} 0.2379 & -0.0046 & 0.01 \\ -0.0093 & 0.2152 & -0.025 \\ -0.0022 & -0.0029 & 0.043 \end{pmatrix} \times 10^{-3}$
$\{T_X,\ T_Y,\ T_N\}$	$\{0.2,\ 0.2,\ 0.2\}$
$\{X_{max},\ Y_{max},\ N_{max}\}$	$\{630,\ 495,\ 675\}$
$\{X_{min},\ Y_{min},\ N_{min}\}$	$\{-630,\ -495,\ -675\}$

Tabelle 6.1.: Modellparameter für das ASV des A-SWARM-Projektes.

Das Bewegungsmodell des ASVs entspricht den Gleichungen (5.1)-(5.3), wobei

$$\boldsymbol{f}(\boldsymbol{\nu}) = \underbrace{\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1,10} \\ f_{21} & f_{22} & \cdots & f_{2,10} \\ f_{31} & f_{32} & \cdots & f_{3,10} \end{pmatrix}}_{\boldsymbol{F}} \underbrace{\begin{pmatrix} u & v & r & vr & ur & uv & r^2 & u^3 & v^3 & r^3 \end{pmatrix}^{\mathrm{T}}}_{\boldsymbol{n}(\boldsymbol{\nu})} \tag{6.1}$$

ist. In der Tabelle 6.1 sind alle Parameter des Bewegungsmodells zusammengefasst. Die Werte für $\boldsymbol{F}$ und $\boldsymbol{B}$ resultieren aus Identifikationsfahrten mit dem realen Fahrzeug. Die Matrix $\boldsymbol{B}$ besitzt vollen Rang. Demzufolge kann sich das ASV unabhängig in allen drei Freiheitsgraden bewegen.

6.1.3. Parametrisierung der ExplorePath-Prozedur

Innerhalb der *ExplorePath*-Prozedur der Bewegungsplanung wird ein Eingangsfilter zur Vermeidung sprunghafter Stellgrößen verwendet. Die zugehörigen Zeitkonstanten werden in den Simulationsexperimenten mit $T_{f1} = T_{f2} = T_{f3} = 0.5$ gewählt. Auf Basis der Modellparameter sowie der gewählten Zeitkonstanten für das Eingangsfilter wird der Posenregler parametrisiert. Die Systemmatrix $\boldsymbol{A}_\eta$ der linearisierten Zustandsraumdarstellung (5.66) besitzt 12 Eigenwerte. Mit dem *pole-placement*-Verfahren wird die Verstärkungsmatrix

$$\boldsymbol{K}_\eta = \begin{pmatrix} \boldsymbol{K}_{\eta,1} & \boldsymbol{K}_{\eta,2} & \boldsymbol{K}_{\eta,3} & \boldsymbol{K}_{\eta,4} \end{pmatrix} \tag{6.2}$$

berechnet, sodass die Polstellen des geschlossenen Regelkreises den gewünschten Polstellen entsprechen. Das System entspricht einer Integratorkette aus vier Integratoren für jeden der drei

Parameter	Wert
$\{T_{f1},\ T_{f2},\ T_{f3}\}$	$\{0.5,\ 0.5,\ 0.5\}$
$\boldsymbol{K}_{\eta,1}$	$\begin{pmatrix} 0.054 & 0 & 0 \\ 0 & 0.054 & 0 \\ 0 & 0 & 0.054 \end{pmatrix}$
$\boldsymbol{K}_{\eta,2}$	$\begin{pmatrix} 0.531 & 0 & 0 \\ 0 & 0.531 & 0 \\ 0 & 0 & 0.531 \end{pmatrix}$
$\boldsymbol{K}_{\eta,3}$	$\begin{pmatrix} 1.785 & 0 & 0 \\ 0 & 1.785 & 0 \\ 0 & 0 & 1.785 \end{pmatrix}$
$\boldsymbol{K}_{\eta,4}$	$\begin{pmatrix} 2.35 & 0 & 0 \\ 0 & 2.35 & 0 \\ 0 & 0 & 2.35 \end{pmatrix}$
$r_{x,max}$	10
$r_{y,max}$	6
r_{ψ}	0.6
$r_{p,min}$	2
$\{\Delta x_{th},\ \Delta y_{th},\ \Delta\psi_{th}\}$	$\{0.25,\ 0.25,\ 0.15\}$
$\{u_{th},\ v_{th},\ r_{th}\}$	$\{0.1,\ 0.1,\ 0.01\}$
$\{X_{th},\ Y_{th},\ N_{th}\}$	$\{10,\ 10,\ 10\}$

Tabelle 6.2.: Tuningparameter für die *ExplorePath*-Prozedur.

Freiheitsgrade. Aus diesem Grund werden vier dreifach-Pole für die Gesamtdynamik vorgegeben. Dabei wird gefordert, dass der geschlossene Regelkreis stabil ist und kein Überschwingen entsteht. Infolgedessen müssen alle Polstellen in der offenen linken Halbebene des Pol-Nullstellen-Diagramms liegen und reellwertig sein. Für die Simulationsuntersuchungen werden die folgenden Werte für die gewünschten Polstellen gewählt.

$$\left\{(-1.2)^3,\ (-0.6)^3,\ (-0.3)^3,\ (-0.25)^3\right\}$$

Hierbei kennzeichnet der Exponent 3, dass es sich um eine dreifache Polstelle handelt. Die resultierenden Teilmatrizen der Verstärkungsmatrix (6.2) können der Tabelle 6.2 entnommen werden.

Eine Pose gilt innerhalb der *ExplorePath*-Prozedur als erreicht, wenn (5.44) erfüllt ist, also die Beträge der Posendifferenz sowie der Geschwindigkeiten und Kräfte unterhalb der festgelegten Schwellwerte liegen. In der Tabelle 6.2 sind ebenfalls die für die Simulationsuntersuchungen verwendeten Schwellwerte sowie die für das *Guidance*-Gesetz gewählten Tuningparameter dargestellt.

6.1.4. Hardware und Testdaten

Sämtliche Untersuchungen werden auf einem Industrie-PC durchgeführt, der mit einer Intel Core i9-13900E CPU und 32 GB Arbeitsspeicher ausgestattet ist. Dieser Industrie-PC wird ebenfalls für die Applizierung des entwickelten Verfahrens auf einem realen Versuchsträger eingesetzt. Das in Kapitel 5 vorgestellte Verfahren zur sequenziellen Bewegungsplanung autonomer Wasserfahrzeuge ist in C++ implementiert.

Um den Einfluss einzelner Parameter analysieren und bewerten zu können, wird das entwickelte Verfahren anhand zahlreicher, unterschiedlicher Szenarien getestet, die realen Situationen entsprechen. Zu diesem Zweck wird ein Testdatensatz mit 100 Szenarien generiert, die sich in ihrem Schwierigkeitsgrad unterscheiden. Sie reichen von trivialen Planungsproblemen, bei denen

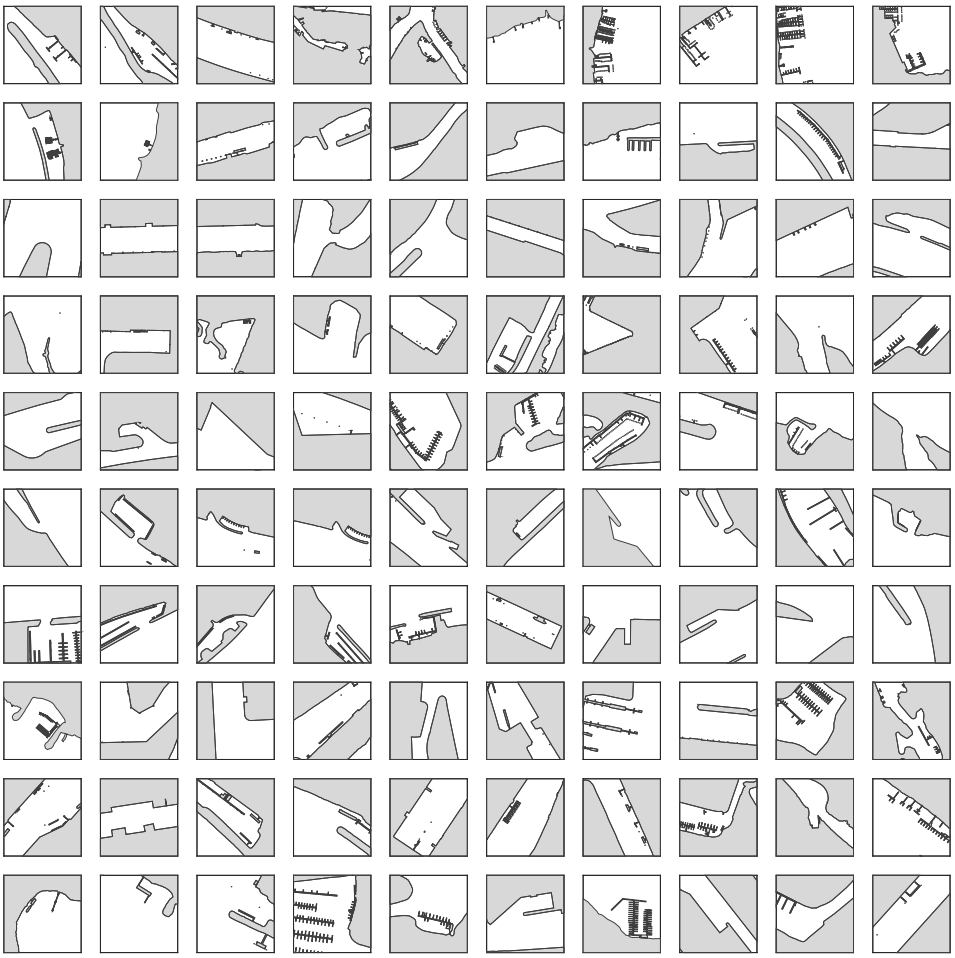


Abbildung 6.2.: Kartenausschnitte mit unbewegten Hindernissen (grau) des generierten Szenarien-Datensatzes SD100.

sich Start- und Zielpunkt direkt miteinander verbinden lassen, bis hin zu Planungsproblemen mit engen Passagen und zahlreichen Hindernissen, die umfahren werden müssen. Als Datengrundlage dienen IENCs, die von der Wasserstraßen- und Schifffahrtsverwaltung des Bundes (WSV) frei online zur Verfügung stehen [103] und eine Vielzahl von Karten deutscher Binnengewässer beinhalten. Basierend auf diesem Kartenmaterial werden 100 verschiedene Punkte entlang von Wasserstraßen ausgewählt. Um jeden Punkt wird ein gleichgroßes Gebiet von $200\,\mathrm{m} \times 200\,\mathrm{m}$ definiert. Unbewegte Hindernisse wie beispielsweise Landflächen, Stege und Pfeiler werden anhand von konvexen Polygonen abgebildet. Zusätzlich wird für jedes Szenario eine Start- und Zielpose derart festgelegt, dass eine Verbindung beider möglich ist. Die Initialwerte für die Geschwindigkeiten, Kräfte und Stellgrößen sind in allen Szenarien auf null gesetzt, damit $\boldsymbol{x}_I \in \mathcal{X}_{free}$ gilt. Alle 100 Szenarien werden in einem Datensatz zusammengefasst, der im Folgenden mit **SD100** bezeichnet wird. Abbildung 6.2 zeigt den generierten Datensatz. Jedes Quadrat entspricht hierbei einem Szenario und zeigt einen $200\,\mathrm{m} \times 200\,\mathrm{m}$ großen Kartenausschnitt, wobei unbewegte Hindernisse grau dargestellt sind.

6.2. Sequenzielle Bewegungsplanung für ein Beispielszenario

In diesem Abschnitt wird das Verfahren der sequenziellen Bewegungsplanung entsprechend dem Algorithmus 22 für ein Beispielszenario aus SD100 appliziert, wobei der Kartenausschnitt auf $60\,\mathrm{m} \times 60\,\mathrm{m}$ reduziert wird, damit das ASV und die Resultate der Pfad- und Be-

Parameter	Wert	Beschreibung
T_s	0.05	Abtastzeit der Lösungstrajektorie.
n^p_{max}	1000	Maximale Anzahl an Knoten in $\mathcal{T}$ während der Pfadplanung.
n^m_{max}	500	Maximale Anzahl an Knoten in $\mathcal{T}$ während der Bewegungsplanung.
w_ψ	3	Wichtungsfaktor für ψ im Kostenterm c_ρ.
w_v	2	Wichtungsfaktor für Querbewegungen im Kostenterm c_v.
w_α	0	Wichtungsfaktor für Quer- und Rückwärtsbewegungen in c_v.
w_β	1	Stauchungsfaktor der Gewichtungsfunktion für die Rückwärtsfahrt.
α	5	Wichtungsfaktor für das Skalarfeld f_ε im Kostenterm c_μ.
β	0.02	Exponentieller Abfall des Skalarfeldes f_ε.
g	0.05	Auflösung der LUT des Skalarfeldes.
m_g	10	Modulofaktor, der angibt, für welche LUT-Einträge f_ε berechnet wird.
λ_{max}	50	Maximale Schrittweite innerhalb des RRT*-Algorithmus.
g_{max}	100	Periode für das *goal sampling*.
k_{max}	10^6	Anzahl der im Vorfeld generierten Pseudozufallszahlen.
D_{xy}	10	Dimension einer Sampling-Box in Positionskoordinaten.
D_ψ	$\pi/4$	Dimension einer Sampling-Box in Winkelkoordinaten.
L_{max}	50	Maximale Weglänge eines Teilpfades für die Bewegungsplanung.
$\{\boldsymbol{\kappa}^b_1, \boldsymbol{\kappa}^b_2\}$	$\left\{\begin{pmatrix}3\\0\end{pmatrix}, \begin{pmatrix}-3\\0\end{pmatrix}\right\}$	Körperfeste Punkte an denen der Kostenterm c_μ ausgewertet wird.

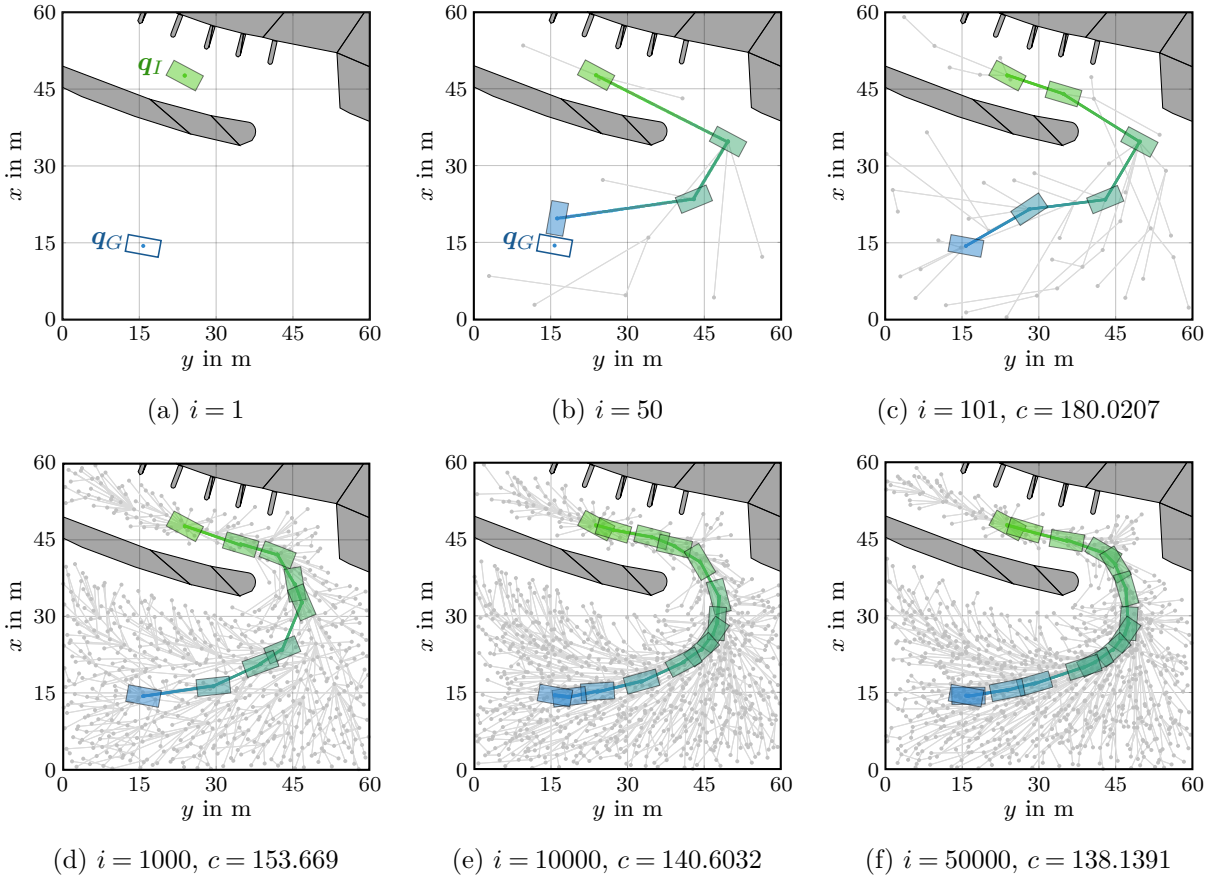
Tabelle 6.3.: Gewählte Tuningparameter für das Simulationsexperiment zur sequenziellen Bewegungsplanung.

wegungsplanung zu unterschiedlichen Zeitpunkten der Iterationen besser zu erkennen sind. Das Beispielszenario beinhaltet 16 konvexe Polygone und die Start- und Zielpose sind mit $\boldsymbol{q}_I = (47.6421,\ 23.8694,\ 2.0608)^{\mathrm{T}}$ und $\boldsymbol{q}_G = (14.3894,\ 15.7678,\ -1.3897)^{\mathrm{T}}$ gegeben. In der Tabelle 6.3 sind die Tuningparameter der sequenziellen Bewegungsplanung zusammengefasst, die für dieses Simulationsexperiment verwendet werden.

6.2.1. Pfadplanung

In der Abbildung 6.3 sind die Resultate der Pfadplanung nach unterschiedlicher Anzahl von Iterationen dargestellt. Die unbewegten Hindernisse sind durch graue Flächen gekennzeichnet und die Start- und Zielpose sind durch grüne bzw. blaue Rechtecke illustriert, welche die Posen mithilfe der geometrischen Fahrzeugform repräsentieren. Der Baum $\mathcal{T}$, welcher während der Iterationen aufgebaut wird, ist durch graue Punkte dargestellt, die über graue Linien miteinander verbunden sind. Dabei ist zu beachten, dass diese Punkte lediglich einer Projektion der dreidimensionalen Posen aus $SE(2)$ auf eine zweidimensionale x-y-Ebene entsprechen. Die farbige Linie zeigt den aktuellen Lösungspfad der Pfadplanung mit einem Farbverlauf von der grünen Startpose zur blauen Zielpose. An jedem Knotenpunkt des Lösungspfades ist zudem das Fahrzeugpolygon dargestellt, um den Heading-Winkel zu verdeutlichen.

Die Teilabbildung 6.3a zeigt das Ergebnis nach der ersten Iteration der Pfadplanung entsprechend Algorithmus 5. Die vorgegebene Zielpose $\boldsymbol{q}_G$ ist als eine blau umrandete Box illustriert. Während der ersten Iteration wird in der *Sample*-Prozedur das *goal sampling* durchgeführt. Die direkte Verbindung von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ ist aufgrund von Hindernissen jedoch nicht möglich, sodass in dieser Iteration kein neuer Knoten dem Baum hinzugefügt wird. Als Resultat enthält der Baum ausschließlich die Startpose $\boldsymbol{q}_I$, welche gleichzeitig den aktuellen Lösungspfad darstellt. In den nachfolgenden Iterationsschritten werden fortlaufend zufällige Samples generiert, von denen einige erfolgreich dem Baum hinzugefügt werden können. Das Ergebnis nach 50 Iterationen ist in Teilabbildung 6.3b dargestellt. Der Baum besteht aus 14 Knoten und enthält bereits Zweige, die um das Hindernis verlaufen, welches sich zwischen $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ befindet. Der aktuelle Lösungspfad



(a) $i = 1$ (b) $i = 50$ (c) $i = 101$, $c = 180.0207$

(d) $i = 1000$, $c = 153.669$ (e) $i = 10000$, $c = 140.6032$ (f) $i = 50000$, $c = 138.1391$

Abbildung 6.3.: Resultate der Pfadplanung nach unterschiedlicher Anzahl an Iterationen i mit den Kostenwerten c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau).

ist farblich markiert und entspricht dem Zweig des Baums, dessen Blattknoten der Zielpose am dichtesten kommt. Die Zielpose selbst wird nicht erreicht. Nach 51 weiteren Iterationen kommt es während der Iteration $i = 101$ erneut zum *goal sampling*, da die Periode für das *goal sampling* entsprechend der Tabelle 6.3 auf $g_{max} = 100$ gesetzt ist. Aus dem Baum, der bereits 50 Knoten enthält, wird ein Knoten ausgewählt, der eine optimale, kollisionsfreie Verbindung zu $\boldsymbol{q}_G$ erzeugt. Ab diesem Zeitpunkt ist die Zielpose $\boldsymbol{q}_G$ mit einem Kostenwert von $c = 180.0207$ im Baum enthalten. Der Kostenwert c beschreibt die Gesamtkosten von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ entlang des Zweiges, welcher die zwei Posen miteinander verbindet, und wird mithilfe der Kostenfunktion (5.13) berechnet.

Die weiteren Teilabbildungen 6.3d, 6.3e und 6.3f zeigen das Ergebnis der Pfadplanung nach 1000, 10000 und 50000 Iterationen. Es ist zu erkennen, dass die Dichte der Knoten des Baums im freien Konfigurationsraum weiter zunimmt. Die maximale Anzahl an Knoten, die im Baum gespeichert werden können, ist auf $n^p_{max} = 1000$ beschränkt. Bereits nach 1418 Iterationen ist diese maximale Anzahl erreicht. In allen nachfolgenden Iterationen können weitere Knoten nur hinzugefügt werden, indem zufällige Blattknoten entfernt werden, wie es Algorithmus 6 vorsieht. Dadurch ist es möglich, den Kostenwert c für einen Lösungspfad fortlaufend zu minimieren. Nach 10000 Iterationen wird auf diese Weise ein Kostenwert von $c = 140.6032$ erreicht und nach 50000 Iterationen ergibt sich $c = 138.1391$. Mit zunehmender Anzahl an Iterationen konvergiert der Lösungspfad gegen den optimalen Pfad.

In der Abbildung 6.4 ist die Konvergenz des Kostenwertes c in Abhängigkeit von der Rechenzeit über einen Zeitraum von einer Sekunde dargestellt. Zum Zeitpunkt $t = 0$ existiert noch kein Lösungspfad. Nach einer Rechenzeit von etwa 2.1 ms wird während der Iteration $i = 101$ eine Lösung gefunden, die einen Kostenwert von $c = 180.0207$ aufweist. Dieser Zeitpunkt entspricht

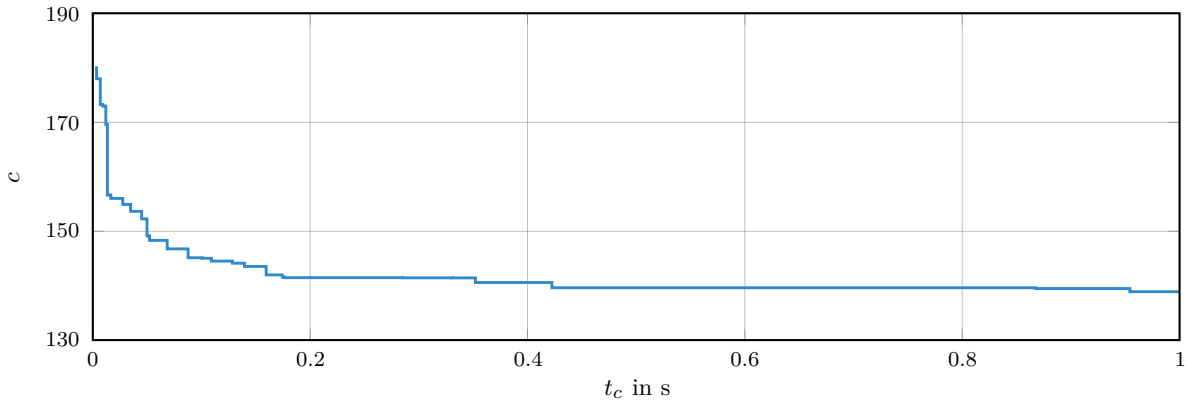


Abbildung 6.4.: Konvergenz des Kostenwertes c von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ in Abhängigkeit von der Rechenzeit t_c.

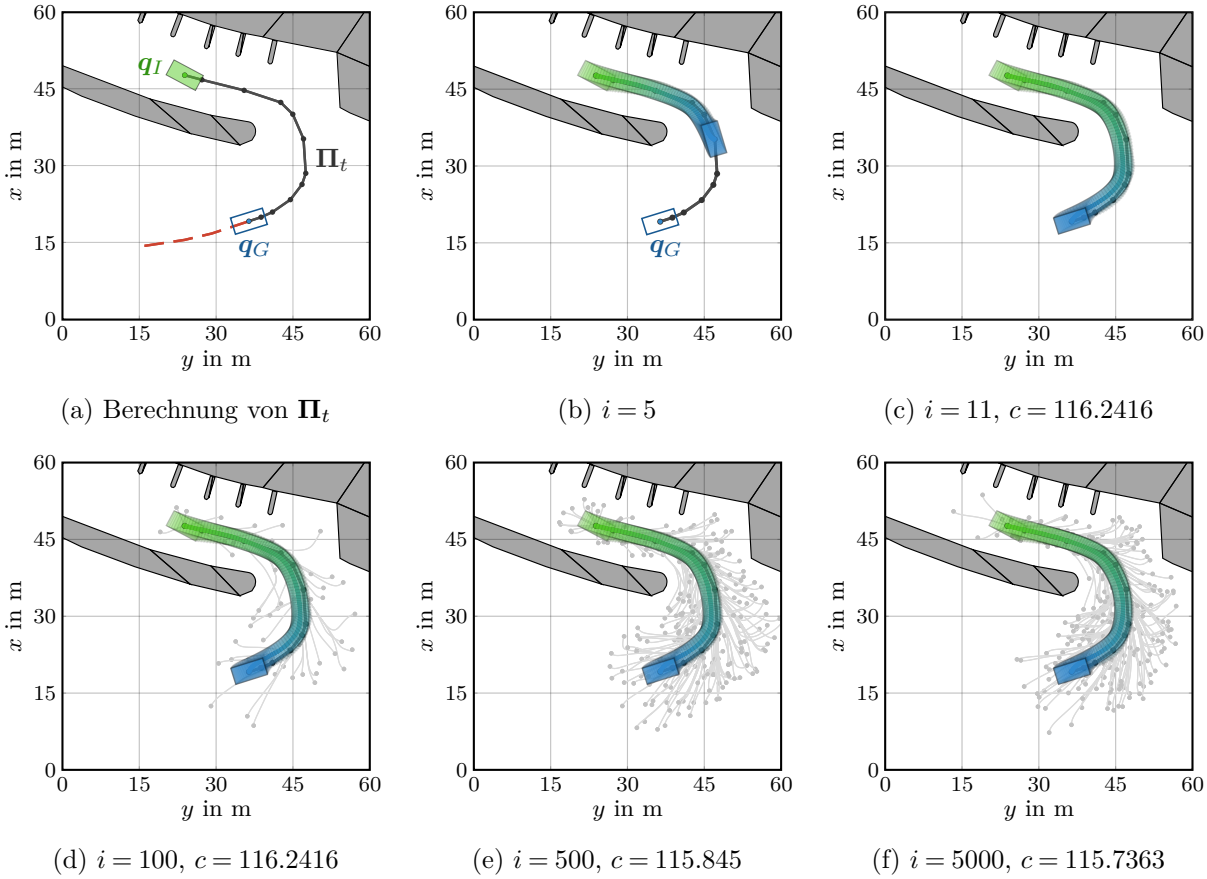
der Abbildung 6.3c. Bereits nach 200 ms wird ein Kostenwert von $c = 141.4604$ erzielt. Ab diesem Zeitpunkt wird zunehmend mehr Rechenzeit benötigt, um den Kostenwert weiter zu minimieren, wobei keine signifikante Verbesserung zu erkennen ist. Nach einer Rechenzeit von einer Sekunde wird nach insgesamt 24779 Iterationen ein Kostenwert von $c = 138.9152$ erreicht. Dies entspricht etwa der Hälfte an Iterationen, die für das Ergebnis in Abbildung 6.3f notwendig sind, bei dem der Kostenwert bei 138.1391 liegt. Die Abbildung 6.4 zeigt, dass nach einer Rechenzeit im Bereich einiger Hundert Millisekunden praktikable Ergebnisse erzielt werden. Die Zeit für das Konvergieren hängt jedoch von der Wahl einiger Tuningparameter sowie dem Szenario ab, worauf in den nachfolgenden Abschnitten dieses Kapitels eingegangen wird.

6.2.2. Bewegungsplanung

Im Anschluss an die Pfadplanung erfolgt die Bewegungsplanung nach Algorithmus 13. In der Abbildung 6.5 sind die Resultate der Bewegungsplanung nach einer unterschiedlichen Anzahl von Iterationen dargestellt. Die Farbgebung ist identisch zu der aus Abbildung 6.3. Es werden jedoch nicht die Knoten und Zweige des Baums gezeigt, sondern die Trajektorien, die sich durch die *ExplorePath*-Prozedur entlang der Zweige ergeben. Ebenso ist nicht der Lösungspfad, sondern die Lösungstrajektorie $\boldsymbol{\xi}$ mit einem Farbverlauf von der grünen Startpose zur blauen Zielpose illustriert. Entlang dieser Lösungstrajektorie ist das Fahrzeugpolygon zu mehreren Zeitpunkten mit einem zeitlichen Abstand von einer Sekunde abgebildet.

Die erste Teilabbildung 6.5a verdeutlicht die Berechnung des Teilpfades $\boldsymbol{\Pi}_t$. Der resultierende Pfad der Pfadplanung wird entsprechend der *TrimPath*-Prozedur aus Algorithmus 15 auf eine maximale Weglänge beschränkt, die nach Tabelle 6.3 mit $L_{max} = 50$ gegeben ist. In der Grafik ist der Teilpfad als schwarze Linie gekennzeichnet und die rot gestrichelte Linie stellt den Teil des Lösungspfades der Pfadplanung dar, der entfernt wurde. Die Zielpose für die Bewegungsplanung ist mit $\boldsymbol{q}_G$ beschriftet und entspricht der Endpose von $\boldsymbol{\Pi}_t$. Der gewählte Datensatz aus SD100 enthält den Anfangszustand $\boldsymbol{x}_I$ mit den Initialwerten $\boldsymbol{\eta}_I = (47.6421,\ 23.8694,\ 2.0608)^{\mathrm{T}}$, $\boldsymbol{\nu}_I = \mathbf{0}$ und $\boldsymbol{\tau}_I = \mathbf{0}$. Die initiale Stellgröße ist mit $\boldsymbol{\tau}_{c,I} = \mathbf{0}$ gegeben.

In der Teilabbildung 6.5b ist das Ergebnis der Bewegungsplanung nach der fünften Iteration dargestellt. Die *Sample*-Prozedur aus Algorithmus 16 wählt während der ersten Iterationen die Posen aus $\boldsymbol{\Pi}_t \setminus \boldsymbol{q}_I$, bevor zufällige Samples generiert werden. Aus diesem Grund wird für die Iteration $i = 5$ bereits eine Lösungstrajektorie bis zur sechsten Pose aus $\boldsymbol{\Pi}_t$ generiert. Alle bisherigen Samples konnten erfolgreich dem Baum hinzugefügt werden, sodass dieser einschließlich des Wurzelknotens sechs Knoten beinhaltet. Die Teilabbildung 6.5c zeigt das Resultat nach sechs weiteren Iterationen. Während der Iteration $i = 11$ wird die Zielpose $\boldsymbol{q}_G$ dem Baum hinzugefügt. Der Kostenwert c beschreibt die Gesamtkosten des Referenzpfades von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ nach der Kostenfunktion (5.13) und resultiert zu $c = 116.2416$. Dieser Kostenwert entspricht dem Kostenwert



(a) Berechnung von $\mathbf{\Pi}_t$ (b) $i = 5$ (c) $i = 11$, $c = 116.2416$

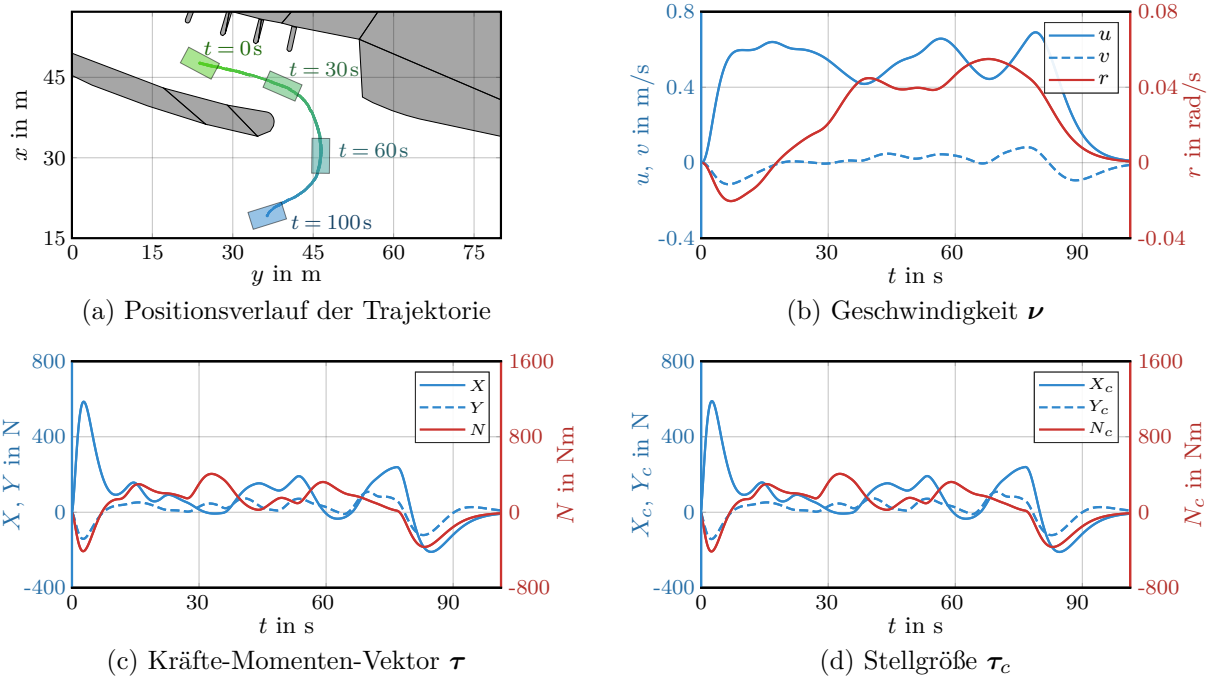
(d) $i = 100$, $c = 116.2416$ (e) $i = 500$, $c = 115.845$ (f) $i = 5000$, $c = 115.7363$

Abbildung 6.5.: Resultate der Bewegungsplanung nach unterschiedlicher Anzahl an Iterationen i mit den Kostenwerten c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau).

des Teilpfades $\mathbf{\Pi}_t$. Der Lösungspfad der zuvor durchgeführten Pfadplanung hat einen Kostenwert von $c = 138.1391$. Da für die Bewegungsplanung nur ein Teilpfad dieses Lösungspfades verwendet wird, ergeben sich folglich Kostenwerte, die unter diesem Wert liegen.

Nach der Iteration $i = 11$ wurden alle Samples aus $\mathbf{\Pi}_t \setminus \boldsymbol{q}_I$ verwendet und alle nachfolgenden Samples werden zufällig aus Sampling-Boxen um $\mathbf{\Pi}_t$ gezogen, wie in Abschnitt 5.4.2 beschrieben wurde. Die Teilabbildung 6.5d zeigt das Resultat der Bewegungsplanung nach 100 Iterationen. Es ist zu erkennen, dass dem Baum erfolgreich weitere Knoten hinzugefügt werden konnten. Der Kostenwert bleibt im Vergleich zur Iteration $i = 11$ unverändert, da keiner der neu hinzugefügten Knoten zu einem Referenzpfad mit geringeren Kosten geführt hat. Mit steigender Anzahl von Iterationen nimmt die Dichte an Knoten in der Umgebung von $\mathbf{\Pi}_t$ zu. Die Teilabbildung 6.5e stellt das Ergebnis nach 500 Iterationen dar. Durch das fortlaufende Sampling wurde ein Referenzpfad mit einem geringeren Kostenwert von $c = 115.845$ gefunden. Diese Verbesserung ist minimal und macht sich in der grafischen Darstellung nicht bemerkbar. Die maximale Anzahl von Knoten, die im Baum gespeichert werden können, ist auf $n^m_{max} = 500$ beschränkt und wird nach 658 Iterationen erreicht. Um weitere Knoten hinzufügen zu können, werden zufällige Blattknoten entfernt, wie in Algorithmus 14 beschrieben wurde. Der Referenzpfad wird infolgedessen kontinuierlich verbessert. Die Teilabbildung 6.5f zeigt das Ergebnis nach 5000 Iterationen. Der resultierende Kostenwert des Referenzpfades liegt bei $c = 115.7363$ und zeigt keine signifikante Verbesserung gegenüber der Iteration $i = 500$. Das liegt vor allem daran, dass alle Posen aus $\mathbf{\Pi}_t \setminus \boldsymbol{q}_I$ zur initialen Lösung während der Iteration $i = 11$ beigetragen haben und $\mathbf{\Pi}_t$ selbst ein Teil des optimierten Pfades der Pfadplanung ist, der nach 50000 Iterationen entstanden ist.

Die zeitlichen Verläufe der Zustands- und Stellgrößen der Lösungstrajektorie sind in Abbildung 6.6 gezeigt. Die Trajektorie wurde mit einer Abtastzeit von $T_s = 0.05$ berechnet und enthält



(a) Positionsverlauf der Trajektorie

(b) Geschwindigkeit $\boldsymbol{\nu}$

(c) Kräfte-Momenten-Vektor $\boldsymbol{\tau}$

(d) Stellgröße $\boldsymbol{\tau}_c$

Abbildung 6.6.: Zeitlicher Verlauf der Zustands- und Stellgrößen der resultierenden Trajektorie.

zeitdiskrete Zustands- und Stellgrößen über eine Dauer von etwa 100 s. In der Abbildung 6.6a sind der Positionsverlauf der Trajektorie in der x-y-Ebene und das Fahrzeugpolygon zu vier verschiedenen Zeitpunkten illustriert. Die Teilabbildungen 6.6b, 6.6c und 6.6d zeigen den zeitlichen Verlauf der Geschwindigkeit $\boldsymbol{\nu}$, des Kräfte-Momenten-Vektors $\boldsymbol{\tau}$ sowie der Stellgröße $\boldsymbol{\tau}_c$. Da jeder Freiheitsgrad des Kraftmodells (5.3) aus einem Verzögerungsglied erster Ordnung mit einer Zeitkonstante von $T = 0.2$ besteht, ist über den dargestellten Zeitraum kein signifikanter Unterschied zwischen $\boldsymbol{\tau}$ und $\boldsymbol{\tau}_c$ zu erkennen. Jedoch ist in der Teilabbildung 6.6d zu sehen, dass es keine sprunghaften Änderungen der Stellgröße gibt, was auf die Wirkung des Eingangsfilters (5.43) zurückzuführen ist. Während der Beschleunigungsphase zum Zeitpunkt $t = 0$ steigt die geforderte Längskraft innerhalb von 2.5 s kontinuierlich auf einen Wert von etwa $X_c = 590$. Am Ende der Trajektorie gehen $\boldsymbol{\nu}$, $\boldsymbol{\tau}$ und $\boldsymbol{\tau}_c$ zu null. Die Zielpose gilt als erreicht, sobald sich alle Zustandsgrößen innerhalb des Zielgebietes nach (5.44) befinden.

Die mittlere Längsgeschwindigkeit im Zeitraum von $t = 5$ bis $t = 75$ liegt bei etwa 0.55 m/s und ist kleiner als die maximale Längsgeschwindigkeit des ASVs von etwa 1.8 m/s, da die Position einen geringen Abstand zu Hindernissen aufweist. Entsprechend der Tabelle 6.2 ist der Radius des Ellipsoids, welches für das *Guidance*-Gesetz verwendet wird, in Längsrichtung auf $r_{x,max} = 10$ beschränkt. Die Größe des Ellipsoids wird je nach Abstand zu Hindernissen nach (5.70) reduziert, sodass der Betrag des maximalen Positionsfehlers bis etwa zum Zeitpunkt $t = 60$ kleiner als $r_{x,max}$ ist. Demzufolge beschränken die Hindernisse in der unmittelbaren Umgebung des ASVs dessen Geschwindigkeit. Erst etwa zum Zeitpunkt $t = 65$ wird der Abstand zu Hindernissen größer und das Fahrzeug beschleunigt, was am kontinuierlichen Anstieg der Stellgröße X_c in der Teilabbildung 6.6d sowie der Längsgeschwindigkeit u in der Teilabbildung 6.6b zu erkennen ist. Ab dem Zeitpunkt $t = 80$ wird die Geschwindigkeit jedoch reduziert, da der Abstand des ASVs zur Zielpose und somit der Regelfehler der Posenregelung kleiner wird.

Die Teilabbildung 6.6d zeigt deutliche Änderungen der Stellgröße $\boldsymbol{\tau}_c$ über den gesamten Zeitverlauf der Trajektorie, was auf mehrere Ursachen zurückzuführen ist. Das in der Teilabbildung 6.6a dargestellte Manöver ist durch Beschleunigungsphasen und signifikante Änderungen des Heading-Winkels gekennzeichnet, sodass kein stationärer Zustand erreicht wird, in dem $\dot{\boldsymbol{\tau}}_c = 0$ gilt. In jedem Zeitschritt wird das Ellipsoid innerhalb des *Guidance*-Gesetzes basierend auf dem Abstand zur Hindernisstruktur angepasst. Durch diese fortwährende Anpassung des Ellipsoids

Parameter	Wertebereich	Beschreibung
w_ψ	> 0	Wichtungsfaktor für ψ im Kostenterm c_ρ.
w_v	≥ 0	Wichtungsfaktor für Querbewegungen im Kostenterm c_v.
w_α	≥ 0	Wichtungsfaktor für Quer- und Rückwärtsbewegungen in c_v.
w_β	> 0	Stauchung der Gewichtungsfunktion für Quer- und Rückwärtsfahrt.
α	≥ 0	Wichtungsfaktor für das Skalarfeld f_ε im Kostenterm c_μ.
β	> 0	Exponentieller Abfall des Skalarfeldes f_ε.

Tabelle 6.4.: Erlaubter Wertebereich der Tuningparameter für die Kostenfunktion.

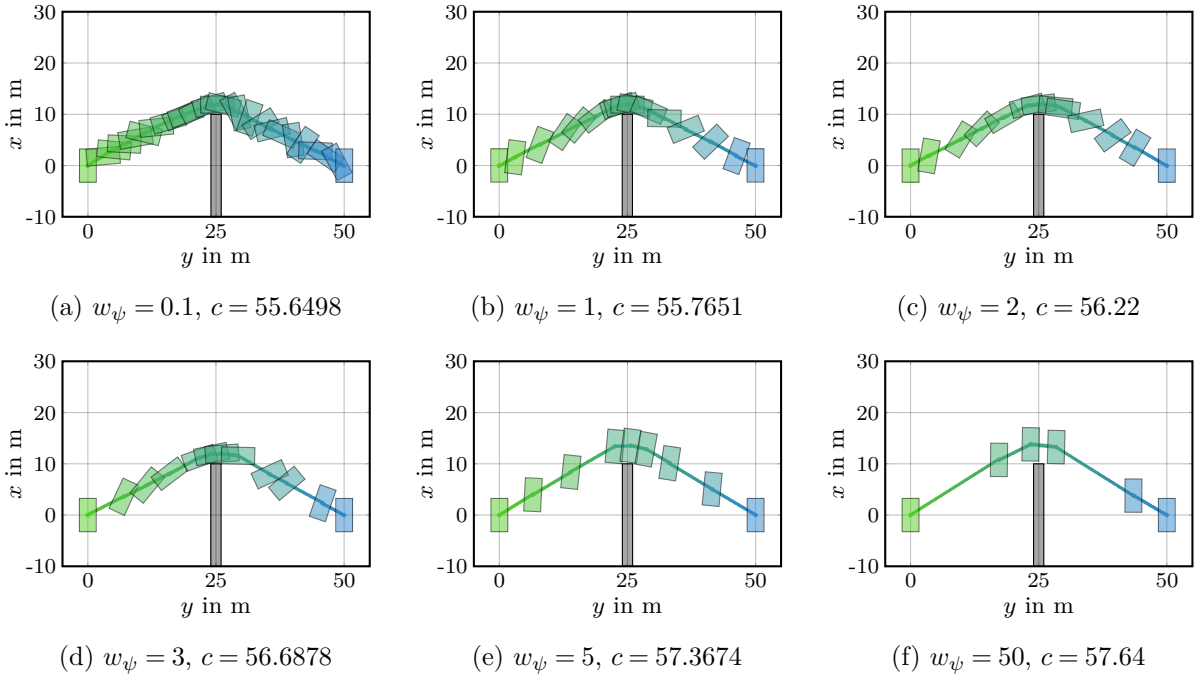
entlang des geplanten Referenzpfades wird die Posendifferenz und die damit einhergehende Stellgröße kontinuierlich verändert. Die Posendifferenz für die Posenregelung wird aus dem Schnittpunkt des Ellipsoids mit dem Referenzpfad berechnet. Da das Ellipsoid unterschiedliche Radien in Längs- und Querrichtung besitzt, hat ein mit ψ gedrehtes Ellipsoid einen Einfluss auf die berechnete Posendifferenz. Darüber hinaus wird der Posenfehler innerhalb der Posenregelung in körperfesten Koordinaten betrachtet, sodass dieser nach (5.46) nichtlinear von ψ abhängt. Da ein linearer Regler verwendet wird, dessen Entwurf auf der Linearisierung des nichtlinearen Modells (5.47) beruht, ergeben sich Abweichungen, auf die der Posenregeler reagiert.

6.3. Einfluss der Tuningparameter der Kostenfunktion

Sowohl in der Pfadplanung als auch in der Bewegungsplanung wird nach einem optimalen Pfad gesucht, entlang dessen die Kostenfunktion (5.13) einen minimalen Wert annimmt. Die Kostenfunktion besteht aus drei Termen, die über Tuningparameter gegeneinander gewichtet werden, wie in Abschnitt 5.3.2 beschrieben wurde. In der Tabelle 6.4 sind die zu wählenden Tuningparameter w_ψ, w_v, w_α, w_β, α und β mit den zugehörigen Wertebereichen dargestellt. Die Wahl der Parameter hat einen signifikanten Einfluss auf den resultierenden Verlauf der Posen innerhalb des Lösungspfades. Um diesen Einfluss analysieren zu können, wird zunächst ein geeignetes Testszenario definiert. Die Start- und Zielpose werden mit $\boldsymbol{q}_I = (0,0,0)^\mathrm{T}$ und $\boldsymbol{q}_G = (0,50,0)^\mathrm{T}$ festgelegt. Zwischen diesen Posen wird ein rechteckiges Hindernis generiert, sodass die Änderung des zu untersuchenden Tuningparameters erkennbare Auswirkungen auf den resultierenden Lösungspfad hat. Der Lösungspfad wird mithilfe des Pfadplanungsalgorithmus berechnet, wobei eine maximale Anzahl an Iterationen anstelle einer maximalen Rechenzeit definiert wird. Für die nachfolgenden Simulationsexperimente sind die verwendeten Tuningparameter der Pfadplanung identisch zu denen aus Tabelle 6.3, während w_ψ, w_v, w_α, w_β, α und β modifiziert werden. Für jeden Versuch werden 10^6 Iterationen durchgeführt.

6.3.1. Tuningparameter w_ψ

Zunächst wird $w_v = w_\alpha = \alpha = 0$ gewählt, um den spezifischen Einfluss von w_ψ auf den resultierenden Lösungspfad zu untersuchen. Es werden sechs Versuche mit unterschiedlichen Werten für w_ψ durchgeführt. Die Ergebnisse sind in der Abbildung 6.7 gezeigt, wobei die Farbgebung identisch zu der aus Abschnitt 6.2 ist. In der Teilabbildung 6.7a ist zu erkennen, dass der Heading-Winkel entlang des Pfades stark variiert. Die Ursache dafür ist, dass Winkeländerungen zwischen zwei Posen gegenüber der Positionsänderung aufgrund des vergleichsweise geringen Wichtungsfaktors $w_\psi = 0.1$ keinen signifikanten Beitrag zur Kostenfunktion liefern. Es tragen vorrangig Konfigurationen mit kurzer Weglänge in x und y zum Lösungspfad bei und der Heading-Winkel dieser Konfigurationen spielt nur eine untergeordnete Rolle. Würde die Anzahl an Iterationen größer gewählt werden, dann entstünden auch Samples mit geringeren Winkelabweichungen. Jedoch erzeugt selbst eine maximale Winkeländerung von π bei einem Wichtungsfaktor von $w_\psi = 0.1$ nur eine Veränderung des Kostenwertes von 0.314,



(a) $w_\psi = 0.1$, $c = 55.6498$ (b) $w_\psi = 1$, $c = 55.7651$ (c) $w_\psi = 2$, $c = 56.22$

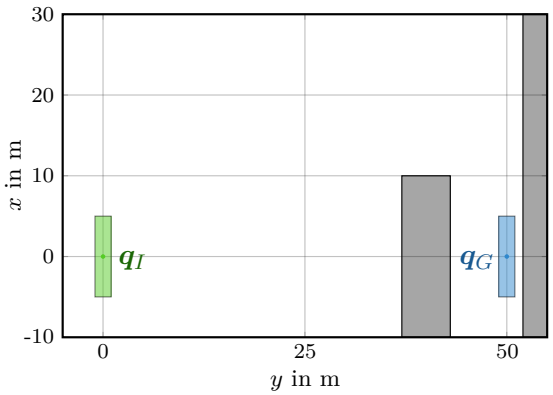
(d) $w_\psi = 3$, $c = 56.6878$ (e) $w_\psi = 5$, $c = 57.3674$ (f) $w_\psi = 50$, $c = 57.64$

Abbildung 6.7.: Resultierender Lösungspfad mit dem Kostenwert c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau) für verschiedene Werte von w_ψ mit $w_v = w_\alpha = \alpha = 0$.

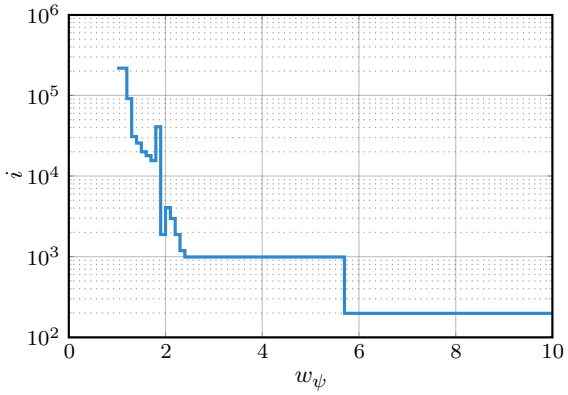
was lediglich etwa 0.6% der Gesamtkosten entspricht. Demzufolge führt weiteres Iterieren zu keiner signifikanten Reduktion des Kostenwertes. In den nachfolgenden Teilabbildungen 6.7b bis 6.7f ist zu erkennen, dass mit größeren Werten für w_ψ Winkeländerungen entlang des Pfades bestraft werden. Der Kostenwert c wird dementsprechend größer. Für Wichtungsfaktoren bis $w_\psi = 3$ wird das rechteckige Hindernis mit einem Heading-Winkel von 90° passiert, da dadurch die Positionsänderungen vergleichsweise klein sind. Ab einem Wichtungsfaktor von $w_\psi = 5$ führen Winkeländerungen zu entsprechend großen Kostenwerten, sodass der optimale Pfad kaum Winkeländerungen aufweist, wie in den Teilabbildungen 6.7e und 6.7f zu erkennen ist.

In der metrischen Distanzfunktion (5.12) wird mit w_ψ der Winkelabstand gegenüber dem Positionsabstand zwischen zwei Konfigurationen gewichtet. Diese Metrik wird unter anderem in der *NearestNeighbor*-Funktion der Pfadplanung aus Algorithmus 6 verwendet, um während einer Iteration den dichtesten Knoten des Baums zu einer zufälligen Konfiguration auszuwählen. Durch zufälliges Sampeln kann w_ψ einen Einfluss auf die Performance der Pfadplanung haben, was an einem konkreten Beispiel verdeutlicht wird.

Gegeben sei ein Szenario entsprechend der Abbildung 6.8a mit zwei rechteckigen Hindernissen in der Umgebung von $\boldsymbol{q}_G$, sodass keine direkte Verbindung von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ möglich ist. Das Fahrzeug ist mit einem Rechteck der Dimension $10\,\mathrm{m} \times 2\,\mathrm{m}$ gegeben. In der Abbildung 6.8b ist die Anzahl der notwendigen Iterationen i in Abhängigkeit von w_ψ dargestellt, die durchgeführt werden müssen, um die Zielkonfiguration mithilfe der *goal-sampling*-Heuristik, die in Abschnitt 5.3.3 beschrieben wurde, exakt zu erreichen. Es ist zu erkennen, dass für Werte von $w_\psi < 2$ die Anzahl an Iterationen zunimmt, während für $w_\psi \geq 5.7$ die Zielpose aufgrund des *goal samplings* bereits in der Iteration $i = 201$ erreicht werden kann. Demzufolge erfordern Werte von $w_\psi < 2$ für dieses konkrete Beispiel eine hohe Rechenzeit. Der Grund für diesen Effekt ist, dass durch zufälliges Sampling eine Konfiguration $\boldsymbol{q}_i$ nahe der Zielkonfiguration $\boldsymbol{q}_G$ generiert und dem Baum hinzugefügt werden kann, die eine hohe Winkelabweichung bei einer vergleichsweise geringen Positionsabweichung aufweist und die durch die geringe Distanz zu $\boldsymbol{q}_G$ als Kandidat für das *goal sampling* verwendet wird. Ist keine kollisionsfreie Verbindung von $\boldsymbol{q}_i$ zu $\boldsymbol{q}_G$ möglich, muss solange iteriert werden, bis ein besserer Kandidat als $\boldsymbol{q}_i$ mit einem geringeren Abstand zu $\boldsymbol{q}_G$ erzeugt



(a) Beispielszenario mit Hindernissen nahe der Zielkonfiguration $\boldsymbol{q}_G$.



(b) Anzahl der Iterationen k bis zum Erreichen der Zielpose in Abhängigkeit von w_ψ.

Abbildung 6.8.: Einfluss von w_ψ auf die Performance für ein konkretes Beispielszenario.

wird. Dieser Sachverhalt wird an einem Zahlenbeispiel veranschaulicht. Es werden zwei Konfigurationen $\boldsymbol{q}_1 = (1,\ 50,\ \pi)^{\mathrm{T}}$ und $\boldsymbol{q}_2 = (10,\ 50,\ 0)^{\mathrm{T}}$ betrachtet, deren Distanz zu $\boldsymbol{q}_G = (0,\ 50,\ 0)^{\mathrm{T}}$ entsprechend der metrischen Distanzfunktion (5.12) mit

$$\rho(\boldsymbol{q}_1, \boldsymbol{q}_G) = \sqrt{1 + (w_\psi \pi)^2}$$

und

$$\rho(\boldsymbol{q}_2, \boldsymbol{q}_G) = 10$$

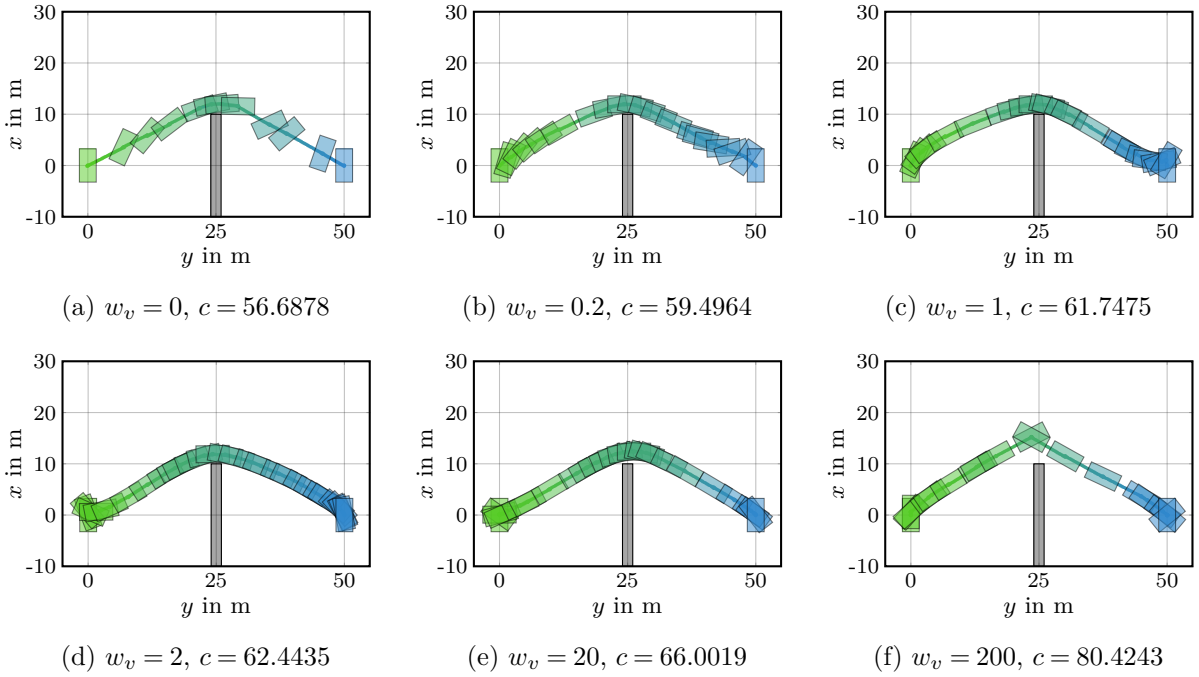
berechnet werden. Ist nun

$$w_\psi = \frac{1}{\pi}\sqrt{10^2 - 1} \approx 3.1671\ ,$$

dann sind $\boldsymbol{q}_1$ und $\boldsymbol{q}_2$ im selben Abstand zu $\boldsymbol{q}_G$. Daraus folgt, dass für $w_\psi < 3.1671$ die Konfiguration $\boldsymbol{q}_1$ dichter an $\boldsymbol{q}_G$ ist, als $\boldsymbol{q}_2$. Somit wird $\boldsymbol{q}_1$ für das *goal sampling* ausgewählt. Da jedoch keine kollisionsfreie Verbindung von $\boldsymbol{q}_1$ zu $\boldsymbol{q}_G$ möglich ist, müssen weitere Iterationen durchgeführt werden. Für $w_\psi > 3.1671$ liegt hingegen $\boldsymbol{q}_2$ dichter an $\boldsymbol{q}_G$ und kann als Kandidat für das *goal sampling* verwendet werden. Die Verbindung von $\boldsymbol{q}_2$ zu $\boldsymbol{q}_G$ ist zudem kollisionsfrei und die Zielkonfiguration kann dem Baum hinzugefügt werden. Es ist zu erwähnen, dass der beschriebene Effekt signifikant vom gewählten Szenario abhängt. Dennoch wird vorgeschlagen $w_\psi > 2$ zu wählen. Ob sich w_ψ prinzipiell auf Basis von Kenngrößen wie der Fahrzeuggröße wählen lässt, wird an dieser Stelle nicht untersucht.

6.3.2. Tuningparameter w_v

Mit dem Tuningparameter w_v wird die Querbewegung des Fahrzeuges entlang des Pfades bestraft. Um den Einfluss von w_v auf den resultierenden Lösungspfad untersuchen zu können, wird $w_\psi = 3$ und $w_\alpha = \alpha = 0$ gewählt. Es werden sechs Versuche mit unterschiedlichen Werten für w_v durchgeführt. Das Ergebnis ist in Abbildung 6.9 gezeigt. Die erste Teilabbildung 6.9a zeigt das Ergebnis für $w_v = 0$ und ist identisch zu Abbildung 6.7d. In der Teilabbildung 6.9b ist der Lösungspfad für einen Wichtungsfaktor von $w_v = 0.2$ dargestellt. Im Vergleich zur Teilabbildung 6.9a ist zu erkennen, dass sich Änderungen des Heading-Winkels hin zu $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ verlagern und dass entlang geradliniger Strecken weniger Änderungen des Heading-Winkels auftreten. Bei größeren Werten von w_v ist dieser Effekt in den Teilabbildungen 6.9c und 6.9d deutlicher zu sehen. Auf geraden Strecken wird eine Querbewegung vermieden, da diese zu hohen Kostenwerten führen würde. Anhand von (5.26) ist zu erkennen, dass der Kostenterm c_v von der Länge einer Wegstrecke abhängt und dass demzufolge das Drehen auf der Stelle $c_v = 0$ ergibt. Werden



(a) $w_v = 0$, $c = 56.6878$ (b) $w_v = 0.2$, $c = 59.4964$ (c) $w_v = 1$, $c = 61.7475$

(d) $w_v = 2$, $c = 62.4435$ (e) $w_v = 20$, $c = 66.0019$ (f) $w_v = 200$, $c = 80.4243$

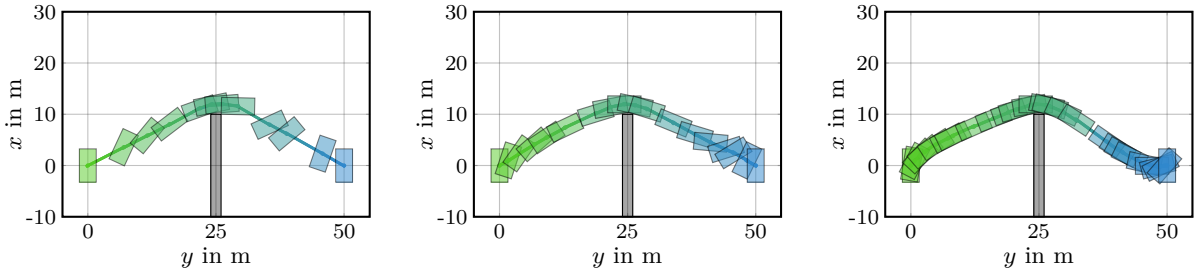
Abbildung 6.9.: Resultierender Lösungspfad mit dem Kostenwert c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau) für verschiedene Werte von w_v mit $w_\psi = 3$, $w_\alpha = 0$, $\alpha = 0$.

vergleichsweise große Werte für w_v gewählt, dann besteht der optimale Pfad nur noch aus Längsbewegungen und dem Drehen auf der Stelle wie Abbildung 6.9f an einem Beispiel für $w_v = 200$ zeigt.

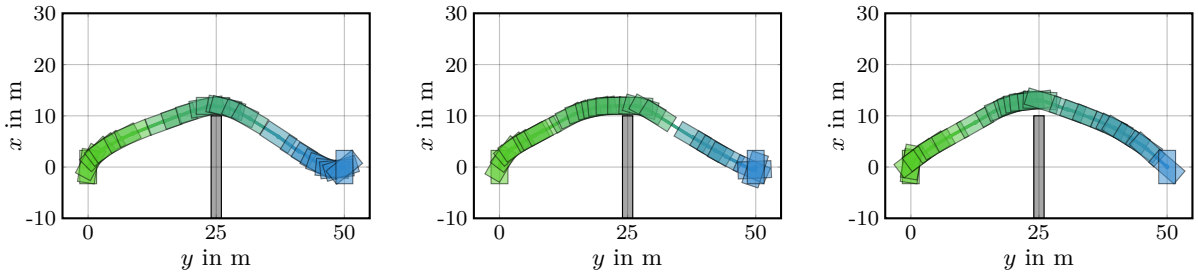
Im Vergleich der Teilabbildungen 6.9c und 6.9d ist zu erkennen, dass der Lösungspfad von einer Vorwärtsbewegung in eine Rückwärtsbewegung übergegangen ist. Der Grund dafür ist, dass das gegebene Pfadplanungsproblem symmetrisch ist. Es existieren zwei optimale Lösungspfade, die $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ miteinander verbinden und denselben Kostenwert haben. Werden $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ vertauscht, entsteht ein äquivalentes Pfadplanungsproblem, welches an der vertikalen Achse bei $y = 25$ gespiegelt ist. Da der Kostenwert von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$ identisch zu dem von $\boldsymbol{q}_1$ zu $\boldsymbol{q}_0$ ist, kann der Lösungspfad für dieses Beispielszenario an dieser vertikalen Achse gespiegelt werden. Bei symmetrischen Pfadplanungsproblemen kann der berechnete Pfad von einer zur nächsten Iteration zwischen unterschiedlichen Lösungspfaden wechseln.

6.3.3. Tuningparameter w_α und w_β

Die zwei Tuningparameter w_α und w_β parametrisieren die Gewichtungsfunktion (5.25), mit deren Hilfe Längsbewegungen gegenüber Quer- und Rückwärtsbewegungen begünstigt werden. Während w_α den zugehörigen Kostenterm gegenüber den anderen Termen der Kostenfunktion gewichtet, beeinflusst w_β die Stauchung der Gewichtungsfunktion (5.25). In den folgenden Simulationsexperimenten wird $w_\beta = 1$, $w_\psi = 3$, $w_v = 0$ und $\alpha = 0$ gewählt. Für den Parameter w_α werden sechs unterschiedliche Werte verwendet, die denen für w_v aus dem vorangegangenen Versuch aus Abschnitt 6.3.2 entsprechen. Das Ergebnis ist in der Abbildung 6.10 gezeigt. In der Grafik 6.10a ist der Lösungspfad für $w_\alpha = 0$ dargestellt, der identisch zu dem aus Abbildung 6.7d ist. Für $w_\alpha = 0.2$ und $w_\alpha = 1$ ergeben sich ähnliche Lösungspfade, wie für $w_v = 0.2$ bzw. $w_v = 1$, was im Vergleich der Teilabbildungen 6.10b und 6.10c mit 6.9b und 6.9c zu erkennen ist. Je größer w_α gewählt wird, desto weniger Änderungen des Heading-Winkels treten entlang geradliniger Strecken auf. Für $w_\alpha = 200$ besteht der Lösungspfad fast ausschließlich aus Längsbewegungen und dem Drehen auf der Stelle. Es ist zu erkennen, dass mit $w_\alpha > 0$ Quer- und Rückwärtsbewegungen vermieden werden.



(a) $w_\alpha = 0$, $c = 56.6878$ (b) $w_\alpha = 0.2$, $c = 59.1761$ (c) $w_\alpha = 1$, $c = 61.8422$



(d) $w_\alpha = 2$, $c = 62.4050$ (e) $w_\alpha = 20$, $c = 66.1418$ (f) $w_\alpha = 200$, $c = 87.3320$

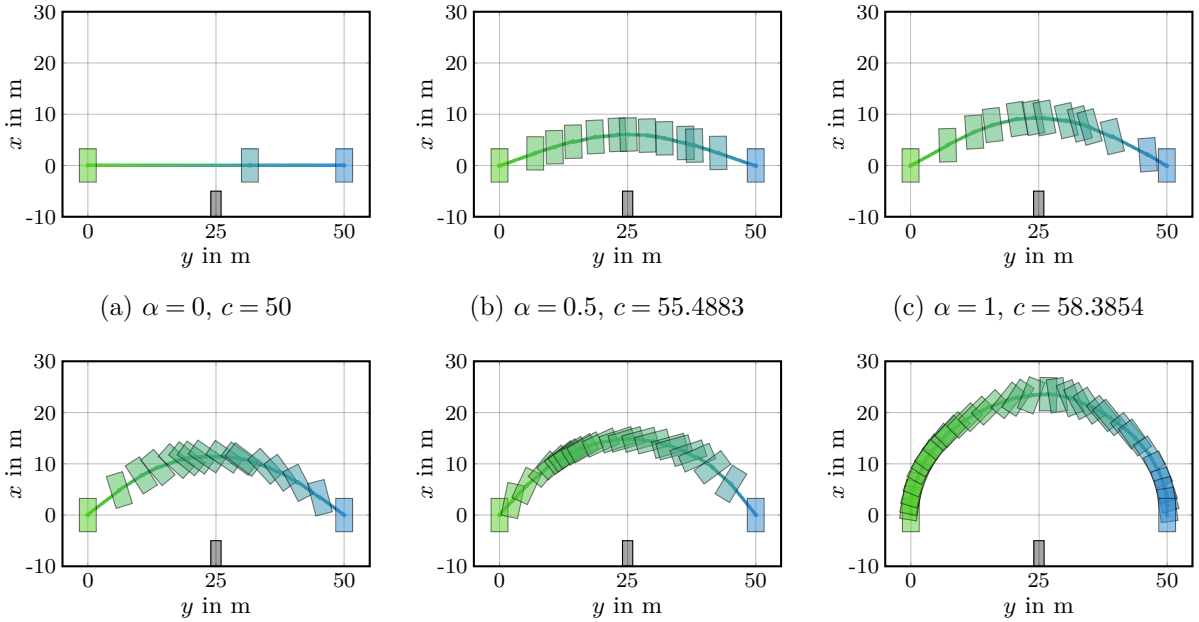
Abbildung 6.10.: Resultierender Lösungspfad mit dem Kostenwert c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau) für verschiedene Werte von w_α mit $w_\beta = 1$, $w_\psi = 3$, $\alpha = 0$.

6.3.4. Tuningparameter α und β

Die zwei Tuningparameter α und β legen das Skalarfeld $f_\varepsilon(x, y)$ des Kostenterms c_μ fest, mit dem ein geringer Abstand des Fahrzeuges zu Hindernissen bestraft wird, wobei β den exponentiellen Abfall des quadratischen Abstandes gewichtet und α die Stärke des Kostenterms c_μ kennzeichnet. Für die folgenden Simulationsexperimente wird das rechteckige Hindernis derart angepasst, dass eine kollisionsfreie Verbindung von $\boldsymbol{q}_I$ zu $\boldsymbol{q}_G$ möglich ist. Die Wichtungsparameter der zwei Kostenterme c_ρ und c_v werden auf $w_\psi = 3$ bzw. $w_v = w_\alpha = 0$ gesetzt und für β wird ein konstanter Wert von 0.01 vorgegeben. Es werden erneut sechs Versuche mit unterschiedlichen Werten für α durchgeführt. Die Abbildung 6.11 zeigt das Ergebnis.

In Abbildung 6.11a ist der Lösungspfad für $\alpha = 0$ dargestellt. Der optimale Pfad entspricht der direkten Verbindung von $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ und der Kostenwert $c = 50$ kommt dem Abstand dieser zwei Konfigurationen gleich. Es ist zu erkennen, dass der Pfad aus drei Posen besteht. Der Grund dafür ist, dass die Schrittweite während des *goal sampling* innerhalb der ersten Iteration nach (5.35) auf einen Wert von $\lambda_s \approx 31.6$ beschränkt wird. Die Zielkonfiguration wird im zweiten Iterationsschritt erreicht und der resultierende Lösungspfad entspricht dem globalen Optimum und wird durch weiteres Iterieren nicht mehr angepasst. Die Abbildungen 6.11b und 6.11c zeigen den Lösungspfad für $\alpha = 0.5$ bzw. $\alpha = 1$. Es ist zu erkennen, dass der Abstand zum Hindernis mit zunehmenden Werten für α größer wird. Der Kostenterm c_μ wird nach (5.18) an zwei körperfesten Punkten $\boldsymbol{\kappa}_1^b$ und $\boldsymbol{\kappa}_2^b$ ausgewertet, die der Tabelle 6.3 entnommen werden können. Dieser Kostenterm wird minimal, wenn beide Punkte einen möglichst großen Abstand zum Hindernis haben. Das ist dann der Fall, wenn der Heading-Winkel senkrecht zur Richtung des stärksten Gefälles des Skalarfeldes verläuft. Da eine Winkeländerung jedoch ebenfalls mit $w_\psi = 3$ bestraft wird, macht sich dieser Effekt erst ab einem Wert von etwa $\alpha = 2$ bemerkbar, wie in Abbildung 6.11d zu erkennen ist. Für $\alpha = 5$ bzw. $\alpha = 100$ ist dieser Effekt in den Abbildungen 6.11e und 6.11f deutlich zu sehen.

Der resultierende Pfad entspricht einem *trade-off* zwischen der kürzesten Verbindung der zwei Konfigurationen $\boldsymbol{q}_I$ und $\boldsymbol{q}_G$ sowie der Maximierung des Abstandes zwischen Fahrzeug und Hindernis. Mit den Tuningparametern α und β wird indirekt festgelegt, wie viel Abstand zu



(a) $\alpha = 0$, $c = 50$ (b) $\alpha = 0.5$, $c = 55.4883$ (c) $\alpha = 1$, $c = 58.3854$

(d) $\alpha = 2$, $c = 61.5207$ (e) $\alpha = 5$, $c = 65.9374$ (f) $\alpha = 100$, $c = 83.6941$

Abbildung 6.11.: Resultierender Lösungspfad mit dem Kostenwert c von der Startpose $\boldsymbol{q}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau) für verschiedene Werte von α mit $w_\psi = 3$, $w_v = 0$, $\beta = 0.01$.

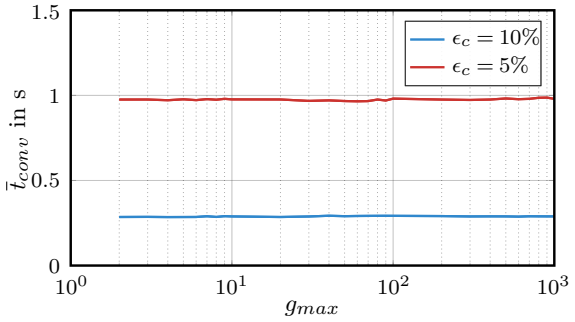
Hindernissen gehalten werden soll und wie groß der Umweg ist, der dafür in Kauf genommen werden muss.

6.4. Einfluss ausgewählter Tuningparameter auf die Performance

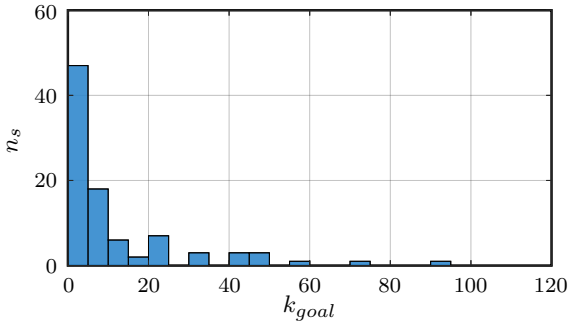
In diesem Abschnitt wird der Einfluss der drei Tuningparameter g_{max}, n^p_{max} und λ_{max} auf die Performance der Pfadplanung untersucht. Der Tuningparameter g_{max} wird innerhalb der *Sample*-Prozedur in Algorithmus 7 verwendet und legt die Periode für das *goal sampling* fest. Mit n^p_{max} wird die maximale Anzahl an Knoten vorgegeben, die im Baum gespeichert werden sollen und mit λ_{max} wird die maximale Schrittweite innerhalb der *Steer*-Prozedur in Algorithmus 8 definiert. In den folgenden Simulationsexperimenten werden für die übrigen Parameter die in Tabelle 6.3 angegebenen Werte verwendet. Zur Quantifizierung der Performance wird die Konvergenzzeit t_{conv} eingeführt, die angibt, wie viel Rechenzeit notwendig ist, damit die Differenz aus dem Kostenwert c des Lösungspfades und dem Kostenwert c^* des optimalen Pfades kleiner als ein vorgegebener Schwellwert wird. Wie in Abbildung 6.4 zu erkennen ist, hängt der Kostenwert c von der Rechenzeit t_c ab. Die Konvergenzzeit wird mit

$$t_{conv} = \min \left\{ t_c \;\middle|\; \frac{c(t_c) - c^*}{c^*} \leq \epsilon_c \right\} \tag{6.3}$$

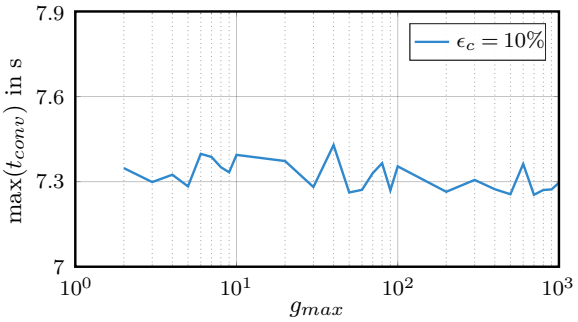
berechnet und entspricht der minimalen Rechenzeit, ab welcher der relative Fehler von c zu c^* unter den Schwellwert $\epsilon_c > 0$ fällt. Das globale Optimum besitzt den Kostenwert $c^* = c(\infty)$ und wird durch die Approximation $c^* \approx c(t_{opt})$ bestimmt, wobei $t_{opt} \gg t_{conv}$ gewählt wird. Für die folgenden Simulationsexperimente wird $t_{opt} = 600$ verwendet. Ausgehend von einem eingestellten Parametersatz wird t_{conv} für jedes Szenario aus SD100 ermittelt und aus allen Konvergenzzeiten die mittlere Konvergenzzeit $\bar{t}_{conv}$ berechnet. Diese stellt einen Indikator für die Performance dar, wobei ein kleinerer Wert einer höheren Performance entspricht.



(a) Mittlere Konvergenzzeit $\bar{t}_{conv}$ in Abhängigkeit von g_{max} für zwei unterschiedliche Fehlertoleranzen ϵ_c.



(b) Häufigkeitsverteilung für die Anzahl zufälliger Samples k_{goal} bis zum Erreichen der Zielpose mit $g_{max} = 2$.



(c) Maximale Konvergenzzeit $\max(t_{conv})$ aller Szenarien in Abhängigkeit von g_{max}.

Abbildung 6.12.: Simulationsergebnisse zum Einfluss des Tuningparameters g_{max} auf die Performance der Pfadplanung.

6.4.1. Tuningparameter g_{max}

Das Ergebnis zur Analyse des Einflusses von g_{max} auf die Performance der Pfadplanung ist in der Abbildung 6.12a gezeigt. Die blaue Kurve zeigt dabei die mittlere Konvergenzzeit $\bar{t}_{conv}$ in Abhängigkeit von g_{max} für eine Fehlertoleranz von $\epsilon_c = 10\%$. Die rote Kurve entspricht hingegen einer kleineren Fehlertoleranz von $\epsilon_c = 5\%$. Es ist zu erkennen, dass eine kleinere Fehlertoleranz zu einer größeren mittleren Konvergenzzeit führt. Jedoch zeigt der Tuningparameter $g_{max} \in [2, 1000]$ keinen signifikanten Einfluss auf die Performance. Der Grund dafür ist, dass das *goal sampling* nur dann durchgeführt wird, wenn die gegebene Zielkonfiguration noch nicht erreicht wurde. Existiert hingegen ein Lösungspfad zur Zielkonfiguration, dann wird das *goal sampling* nicht mehr verwendet. Es ist jedoch zu erwarten, dass für größere Werte von g_{max} die Konvergenzzeit steigt, da eine Zielkonfiguration nur durch das *goal sampling* exakt erreicht werden kann. Demnach müssen mindestens g_{max} Iterationen durchgeführt werden, um eine Verbindung zur Zielpose zu erhalten, sofern diese nicht bereits während der ersten Iteration möglich ist. Die damit einhergehende Rechenzeit für die Durchführung zahlreicher Iterationen hat dann einen zunehmenden Einfluss auf die Performance.

Anschließend wird für alle Szenarien aus SD100 untersucht, wie viele Iterationen mindestens notwendig sind, um die Zielpose zu erreichen. Dafür wird $g_{max} = 2$ gewählt, sodass nach jedem zufällig generierten Sample das *goal sampling* durchgeführt wird. Die Anzahl zufällig generierter Samples bis zum Erreichen der Zielpose wird mit k_{goal} bezeichnet. Abbildung 6.12b zeigt ein Histogramm, welches die Anzahl n_s der Szenarien aus SD100 in Abhängigkeit von k_{goal} dargestellt. Aus Gründen der Anschaulichkeit sind ausschließlich Szenarien bis $k_{max} = 120$ abgebildet, was einer Anzahl von 92 Szenarien entspricht. Die verbleibenden 8 Szenarien benötigen mehr als 120 zufällige Samples, um das Ziel zu erreichen. Der größte Wert für k_{goal} liegt bei 1582. Aus allen Szenarien resultiert ein Mittelwert von $\bar{k}_{max} = 55$. Der Median liegt bei $\tilde{k}_{goal} = 5$ und

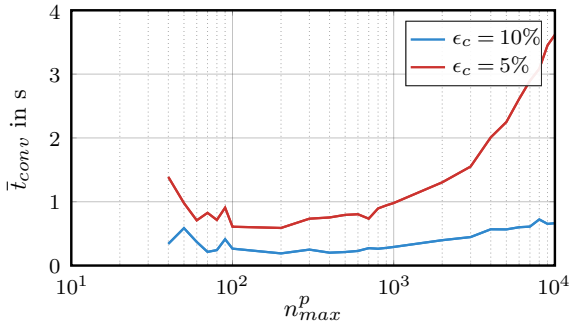


Abbildung 6.13.: Einfluss des Tuningparameters n^p_{max} auf die Performance der Pfadplanung.

entspricht dem ersten Balken des Diagramms. Das bedeutet, dass in 47% der Szenarien nach fünf Iterationen eine Verbindung zum Ziel gefunden wurde.

Es ist zu erwähnen, dass die Abbildung 6.12a nur die mittlere Konvergenzzeit zeigt. Demzufolge kann die Konvergenzzeit für einzelne Szenarien größer ausfallen. In der Abbildung 6.12c ist die maximale Konvergenzzeit $\max(t_{conv})$ aller Szenarien in Abhängigkeit von g_{max} für eine Fehlertoleranz von $\epsilon_c = 10\%$ dargestellt. Diese maximale Konvergenzzeit entspricht der Konvergenzzeit des Szenarios mit dem größten k_{goal}. Es ist zu erkennen, dass die maximale Konvergenzzeit bei etwa 7.3 Sekunden liegt und ebenfalls keine signifikante Abhängigkeit von g_{max} besteht. Daraus lässt sich deduzieren, dass für g_{max} ein beliebiger Wert im Bereich $[2, 1000]$ gewählt werden kann, ohne dass dies einen negativen Einfluss auf die Performance hat.

6.4.2. Tuningparameter n^p_{max}

Der Tuningparameter n^p_{max} legt fest, wie viele Knoten in $\mathcal{T}$ maximal gespeichert werden können. In der Abbildung 6.13 ist der Einfluss von n^p_{max} auf die Performance der Pfadplanung dargestellt. Es sind zwei Kurven gezeigt, die sich in der Wahl der Fehlertoleranz ϵ_c unterscheiden. Eine geringere Fehlertoleranz führt zu einer größeren mittleren Konvergenzzeit. Es ist zu erkennen, dass die mittlere Konvergenzzeit ein Minimum bei etwa $n^p_{max} = 200$ aufweist. Bei einer geringeren Anzahl an Knoten wird der Suchraum nicht ausreichend dicht mit zufälligen Samples belegt, da fortlaufend Knoten entfernt werden müssen, um neue Knoten hinzufügen zu können. Das erschwert das Finden eines optimalen Lösungspfades. Als Folge steigt die mittlere Konvergenzzeit. Ebenso ist eine geringe Anzahl an maximalen Knoten des Baums problematisch, da mitunter kein Pfad zum Ziel aufgrund dieser beschränkten Anzahl an Knoten gefunden werden kann. In diesem Simulationsexperiment konnten einige Szenarien für $n^p_{max} < 40$ nicht innerhalb einer Rechenzeit von 10 Sekunden gelöst werden.

In der Abbildung 6.13 ist zu erkennen, dass die mittlere Konvergenzzeit ab $n^p_{max} = 500$ mit zunehmender Anzahl maximaler Knoten weiter ansteigt. Der Grund dafür liegt in der *FindBestParent*- und *Rewire*-Prozedur des RRT*-Algorithmus. Je dichter die Samples beieinander liegen, desto mehr Knoten müssen innerhalb dieser zwei Prozeduren trotz der reduzierten Schrittweite λ_s untersucht werden, was sich signifikant auf die Rechenzeit einer Iteration auswirkt. Demnach lässt sich für Pfadplanungprobleme, deren Szenarien denen aus SD100 ähnlich sind, eine geringe mittlere Konvergenzzeit und somit hohe Performance für n^p_{max} im Bereich $[100, 500]$ erzielen.

6.4.3. Tuningparameter λ_{max}

Die Abbildung 6.14 zeigt den Einfluss der maximalen Schrittweite λ_{max} auf die Performance der Pfadplanung. Die mittlere Konvergenzzeit $\bar{t}_{conv}$ ist erneut für zwei Fehlertoleranzen $\epsilon_c = 10\%$ und $\epsilon_c = 5\%$ abgebildet. Es ist zu erkennen, dass für $\lambda_{max} < 10$ die mittlere Konvergenzzeit steigt, was darauf zurückzuführen ist, dass bei einer kleinen Schrittweite mehr Iterationen durchgeführt werden müssen, um die gleiche Wegstrecke zwischen zwei Konfigurationen zurückzulegen. Als Folge

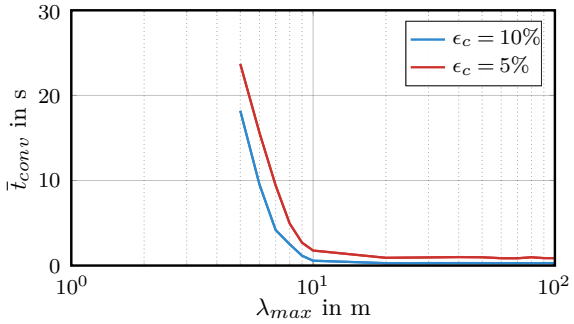
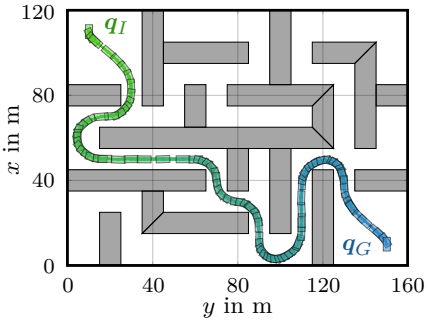
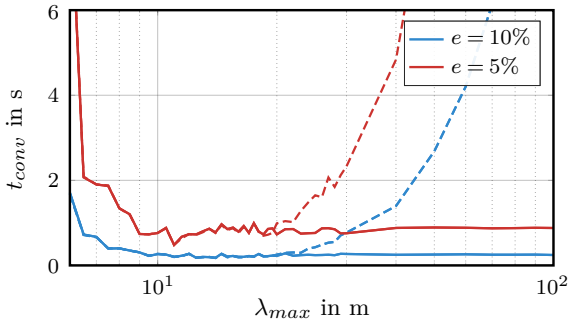


Abbildung 6.14.: Einfluss des Tuningparameters λ_{max} auf die Performance der Pfadplanung.



(a) Testszenario in Form eines Labyrinths sowie der resultierende Lösungspfad von der Startkonfiguration $\boldsymbol{q}_I$ (grün) zur Zielkonfiguration $\boldsymbol{q}_G$ (blau).



(b) Einfluss der maximalen Schrittweite λ_{max} auf die mittlere Konvergenzzeit $\bar{t}_{conv}$ (gestrichelt: ohne Begrenzung auf optimale Schrittweite).

Abbildung 6.15.: Ergebnis des Simulationsexperiments zur Analyse des Einflusses von λ_{max} auf die Performance für ein Szenario in Form eines Labyrinths.

sind mehr Iterationen notwendig, um einen Konfigurationsraum abdecken zu können, dessen Dimensionen signifikant größer als die gewählte Schrittweite sind. Mit dem Tuningparameter λ_{max} wird lediglich ein oberer Grenzwert für die Schrittweite vorgegeben. Die tatsächliche Schrittweite λ_s ergibt sich nach (5.35) aus dem Minimum einer optimalen Schrittweite und λ_{max}. Bei einem Baum mit $n^p_{max} = 1000$ Knoten entspricht die optimale Schrittweite für Szenarien mit einem Gebiet von $200\,\mathrm{m} \times 200\,\mathrm{m}$ einem Wert von etwa 23.7. Die Wahl einer großen Maximalschrittweite zeigt demzufolge ab $\lambda_{max} = 20$ keine signifikante Erhöhung der mittleren Konvergenzzeit.

In einem weiteren Simulationsexperiment wird untersucht, ob dieser Zusammenhang auch bei einem Szenario besteht, welches durch zahlreiche enge Passagen gekennzeichnet ist, sodass große Schrittweiten mit einer höheren Wahrscheinlichkeit dazu führen, dass Samples aufgrund von Kollisionen verworfen werden müssen. Dazu wird ein Testszenario in Form eines Labyrinths definiert. Auch wenn dieses Szenario keine praktische Relevanz für Wasserfahrzeuge hat, lässt sich damit dennoch der Einfluss von λ_{max} auf die Gesamtperformance analysieren. Das Ergebnis des Simulationsexperiments ist in Abbildung 6.15 gezeigt. Das Testszenario ist in der Teilabbildung 6.15a dargestellt und zeigt das Labyrinth sowie den Lösungspfad von der grün gekennzeichneten Startkonfiguration $\boldsymbol{q}_I$ zur blauen Zielkonfiguration $\boldsymbol{q}_G$. Die Teilabbildung 6.15b zeigt die mittlere Konvergenzzeit $\bar{t}_{conv}$ in Abhängigkeit von der maximalen Schrittweite λ_{max} für zwei Fehlertoleranzen. Anhand der durchgezogenen Kurven ist zu erkennen, dass für $\lambda_{max} < 10$ die mittlere Konvergenzzeit aus den bereits genannten Gründen steigt. Bei großen Maximalschrittweiten bis $\lambda_{max} = 100$ zeigt sich in diesem Fall der gleiche Zusammenhang wie in Abbildung 6.14. Wird hingegen auf eine optimale Schrittweite verzichtet und $\lambda_s = \lambda_{max}$ gewählt, dann ergibt sich der gestrichelte Verlauf in der Teilabbildung 6.15b. Eine große Schrittweite führt aufgrund der Hindernisstruktur des Labyrinths vermehrt zu Kollisionen, sodass zahlreiche Iterationen des RRT*-Algorithmus keine neuen Knoten zum Baum hinzufügen können. Demzufolge sind mehr

Iterationen notwendig, um durch zufälliges Sampling einen Pfad durch das Labyrinth zu finden, sodass aufgrund der damit einhergehenden Erhöhung der Rechenzeit die mittlere Konvergenzzeit steigt. Es bildet sich ein Optimum für eine Schrittweite von 10 bis 20 aus, was etwa dem Abstand der Wände des Labyrinths bzw. einer mittleren freien Weglänge innerhalb des Labyrinths entspricht.

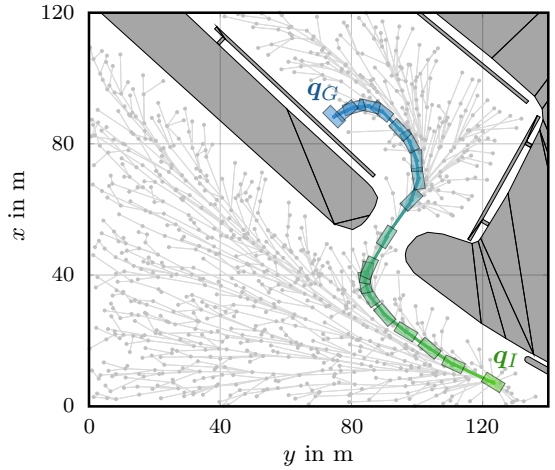
Da bei der Berechnung der Schrittweite nach (5.35) kein Vorteil bei der Festlegung einer obere Beschränkung der Schrittweite zu erkennen ist, kann die Wahl der Schrittweite λ_s direkt nach der optimalen Schrittweite erfolgen, indem $\lambda_{max} = \infty$ gesetzt wird.

6.5. Validierung der WarmStart-Prozedur an einem Beispiel

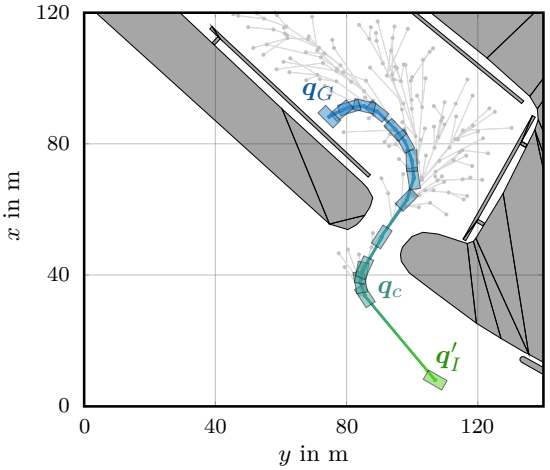
Mithilfe der *WarmStart*-Prozedur aus Algorithmus 11 wird eine existierende Lösung der Pfadplanung als Ausgangsbasis für ein nachfolgendes Pfadplanungsproblem verwendet. In diesem Abschnitt wird die *WarmStart*-Prozedur an einem Beispielszenario des SD100-Datensatzes validiert. Für die Tuningparameter werden die Werte entsprechend der Tabelle 6.3 festgelegt. In einem Simulationsexperiment wird zunächst das Pfadplanungsproblem für das gewählte Szenario gelöst. Anschließend wird ein Warmstart mit einer veränderten Startkonfiguration durchgeführt und für das neue Pfadplanungsproblem die Konvergenz des Kostenwertes analysiert. Das Ergebnis des Simulationsexperiments ist in Abbildung 6.16 dargestellt. Die Teilabbildung 6.16a zeigt das Ergebnis der Pfadplanung nach 50000 Iterationen. Für eine bessere Darstellung ist der Kartenausschnitt sowie der zugehörige Suchraum auf $120\,\mathrm{m} \times 140\,\mathrm{m}$ beschränkt. Der Lösungspfad verläuft von der grün dargestellten Startkonfiguration $\boldsymbol{q}_I$ zur blau gekennzeichneten Zielkonfiguration $\boldsymbol{q}_G$. Der Baum des RRT*-Algorithmus füllt den gesamten freien Raum aus und ist durch die grauen Punkte und Linien abgebildet.

Im Anschluss wird die Startkonfiguration verändert, um zu simulieren, dass sich das ASV bewegt hat. Die Startkonfiguration für das nachfolgende Pfadplanungsproblem wird mit $\boldsymbol{q}_I'$ bezeichnet und sowohl die Hindernisse als auch die Zielkonfiguration bleiben unverändert. Für das neue Pfadplanungsproblem wird ein Warmstart durchgeführt. Das Ergebnis der *WarmStart*-Prozedur ist in Abbildung 6.16b gezeigt. Es ist zu erkennen, dass $\boldsymbol{q}_I'$ der Wurzelknoten des neuen Baums ist und über den Knoten $\boldsymbol{q}_c$ mit dem Lösungspfad des vorangegangenen Pfadplanungsproblems verbunden ist. Dabei bleiben alle Knoten des Lösungspfades erhalten, die auf $\boldsymbol{q}_c$ folgen. Ebenso bleiben die Zweige des Baums bestehen, welche von den Knoten dieses neuen Lösungspfades ausgehen. Alle grau gekennzeichneten Zweige aus Abbildung 6.16a, die direkt mit Elternknoten von $\boldsymbol{q}_c$ verbunden waren, wurden entfernt.

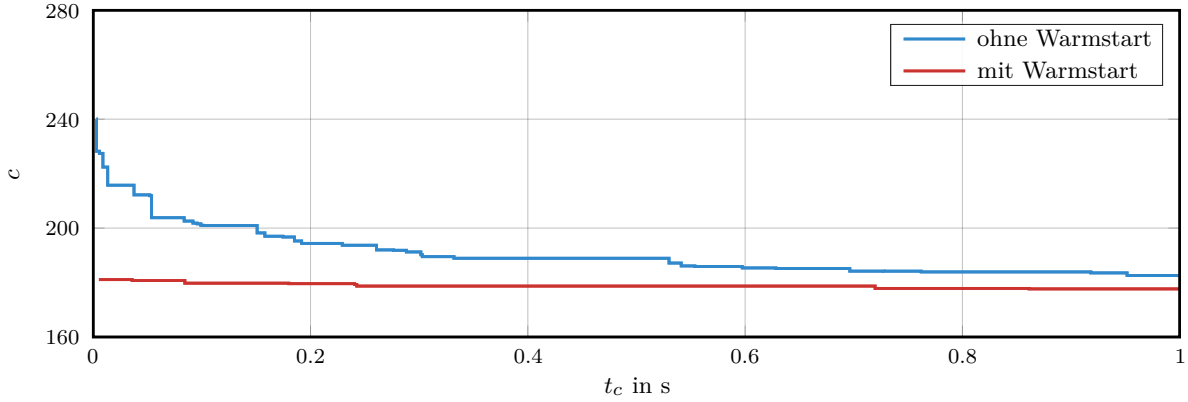
Nach der Durchführung des Warmstarts wird das neue Pfadplanungsproblem nach Algorithmus 5 gelöst. Der Vorteil der *WarmStart*-Prozedur zeigt sich in der Konvergenz des Kostenwertes c, welcher die Gesamtkosten des Lösungspfades von $\boldsymbol{q}_I'$ zu $\boldsymbol{q}_G$ entsprechend der Kostenfunktion (5.13) angibt. Die Abbildung 6.16c zeigt den Kostenwert c in Abhängigkeit der Rechenzeit t_c, wobei die blaue Kurve den Fall kennzeichnet, bei dem kein Warmstart durchgeführt wird, und die rote Kurve den Fall darstellt, bei dem der Warmstart verwendet wird. Wird kein Warmstart durchgeführt, muss der Baum von $\boldsymbol{q}_I'$ zu $\boldsymbol{q}_G$ komplett neu aufgebaut werden. In diesem Fall, welcher auch als Kaltstart bezeichnet wird, wird bereits nach einer Zeit von etwa 3 ms ein Lösungspfad zum Ziel gefunden, der jedoch noch einen Kostenwert von etwa 240 aufweist. Nach einer Rechenzeit von einer Sekunde wird nach insgesamt 23799 Iterationen ein Kostenwert von etwa 183 erzielt. Wird hingegen ein Warmstart durchgeführt, ergibt sich der in Abbildung 6.16c rot dargestellte Verlauf des Kostenwertes. Die *WarmStart*-Prozedur hat in diesem Simulationsversuch eine Rechenzeit von 2.6 ms benötigt. Der Warmstart beginnt mit einem Kostenwert von etwa 181 und erreicht nach einer Rechenzeit von einer Sekunde innerhalb von 22524 Iterationen einen Kostenwert von etwa 178. Im Vergleich der zwei Kurven aus Abbildung 6.16c ist erkennbar, dass bei einem Kaltstart mehr Rechenzeit erforderlich ist, um den gleichen Kostenwert wie bei einem Warmstart zu erreichen.



(a) Ergebnis der Pfadplanung von der Startkonfiguration $\boldsymbol{q}_I$ (grün) zur Zielkonfiguration $\boldsymbol{q}_G$ (blau).



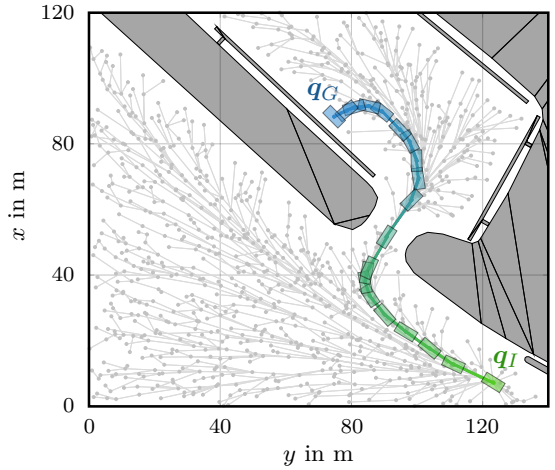
(b) Ergebnis der *WarmStart*-Prozedur für eine neue Startkonfiguration $\boldsymbol{q}'_I$ (grün).



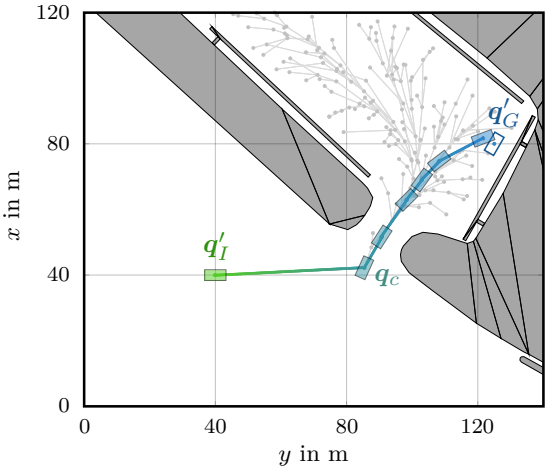
(c) Konvergenz des Kostenwertes c von $\boldsymbol{q}'_I$ zu $\boldsymbol{q}_G$ in Abhängigkeit der Rechenzeit t_c mit und ohne Warmstart.

Abbildung 6.16.: Ergebnis des Simulationsexperiments zur Validierung der *WarmStart*-Prozedur bei Veränderung der Startkonfiguration.

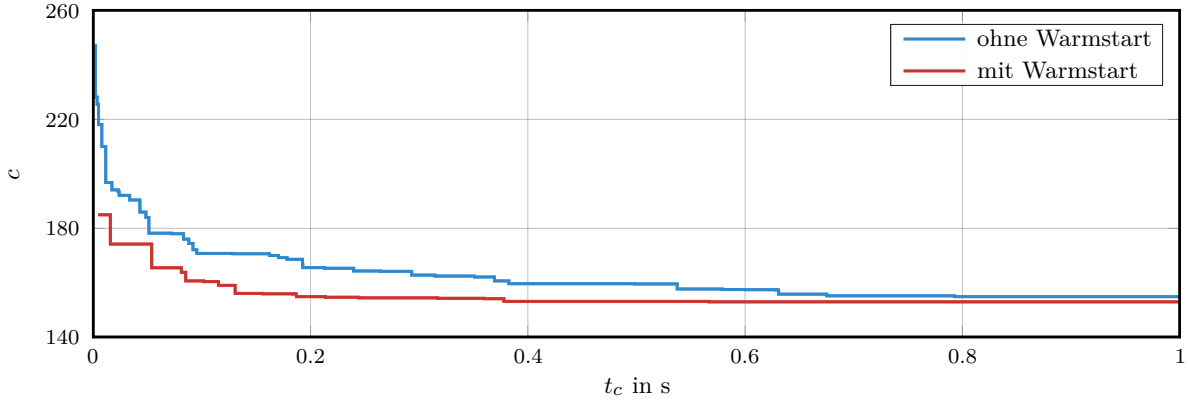
In einem zweiten Simulationsexperiment wird untersucht, wie sich der Warmstart gegenüber einem Kaltstart verhält, wenn Start- und Zielkonfiguration signifikant verändert werden. Das Ergebnis dieser Simulation ist in Abbildung 6.17 gezeigt, wobei das Ausgangsproblem identisch zu dem des ersten Simulationsexperimentes ist. In der Teilabbildung 6.17a ist erneut das Ergebnis der Pfadplanung nach 50000 Iterationen dargestellt. Im Anschluss wird eine veränderte Startkonfiguration $\boldsymbol{q}'_I$ sowie Zielkonfiguration $\boldsymbol{q}'_G$ vorgegeben und ein Warmstart durchgeführt. Die Teilabbildung 6.17b zeigt das Ergebnis der *WarmStart*-Prozedur. Aus dem bestehenden Lösungspfad wird ein Knoten $\boldsymbol{q}_c$ ausgewählt, sodass die Verbindung von $\boldsymbol{q}'_I$ zu $\boldsymbol{q}_c$ einen minimalen Kostenwert aufweist. Alle Zweige des bestehenden Baums aus Abbildung 6.17a, die mit Elternknoten von $\boldsymbol{q}_c$ verbunden waren, wurden entfernt und sämtliche Zweige, die in $\boldsymbol{q}_c$ starten und sich im oberen Teil des Hafenbeckens befinden, bleiben erhalten, wie es in Abbildung 6.17b zu erkennen ist. Die neue Zielkonfiguration $\boldsymbol{q}'_G$ befindet sich ebenfalls im oberen Hafenbecken und ist als blau umrandetes Rechteck illustriert. Der neue Lösungspfad ist als grün-blaue Kurve dargestellt und entspricht dem Zweig, dessen Endpose nach der Distanzmetrik (5.12) der neuen Zielkonfiguration $\boldsymbol{q}'_G$ am dichtesten liegt. Eine Verbindung zu $\boldsymbol{q}'_G$ wird innerhalb der *WarmStart*-Prozedur nicht vorgenommen, da während der ersten Iteration der Pfadplanung ohnehin die Zielkonfiguration aufgrund des *goal sampling* gewählt wird.



(a) Ergebnis der Pfadplanung von der Startkonfiguration $\boldsymbol{q}_I$ (grün) zur Zielkonfiguration $\boldsymbol{q}_G$ (blau).



(b) Ergebnis der *WarmStart*-Prozedur für eine neue Startkonfiguration $\boldsymbol{q}'_I$ (grün) und Zielkonfiguration $\boldsymbol{q}'_G$ (blau).



(c) Konvergenz des Kostenwertes c von $\boldsymbol{q}'_I$ zu $\boldsymbol{q}'_G$ in Abhängigkeit der Rechenzeit t_c mit und ohne Warmstart.

Abbildung 6.17.: Ergebnis des Simulationsexperiments zur Validierung der *WarmStart*-Prozedur bei signifikanter Veränderung der Start- und Zielkonfiguration.

In der Abbildung 6.17c ist die Konvergenz des Kostenwertes c für den Lösungspfad von $\boldsymbol{q}'_I$ zu $\boldsymbol{q}'_G$ in Abhängigkeit der Rechenzeit gezeigt, wobei die blaue Kurve den Kaltstart und die rote Kurve den Warmstart kennzeichnet. Die zwei Kurven konvergieren gegen ein Minimum. Jedoch startet der Warmstart nach einer Rechenzeit von 1.2 ms mit einem geringeren Kostenwert von $c = 185$ als der Kaltstart, welcher mit $c = 247$ beginnt. Nach einer Rechenzeit von einer Sekunde erreicht c im Falle des Kaltstarts nach 23865 Iterationen einen Wert von 155, während sich im Falle des Warmstarts nach 22444 Iterationen ein Kostenwert von 153 ergibt. Unter Verwendung des Warmstarts werden in gleicher Rechenzeit geringere Kostenwerte als bei einem Kaltstart erzielt. Ist kein Warmstart möglich, beispielsweise weil $\boldsymbol{q}'_I$ aufgrund von Hindernissen nicht mit dem bestehenden Lösungspfad verbunden werden kann, dann wird ein Kaltstart durchgeführt und der Baum muss komplett neu aufgebaut werden. Die für den Warmstart notwendige Rechenzeit im Bereich weniger Millisekunden stellt dabei keinen signifikanten Nachteil dar.

Parameter	Wert	Beschreibung
t^p_{max}	1	Maximale Rechenzeit für die Pfadplanung.
t^m_{max}	1	Maximale Rechenzeit für die Bewegungsplanung.
t_ϵ	18	Zusätzliches Zeitintervall für die Wahl neuer Initialwerte.

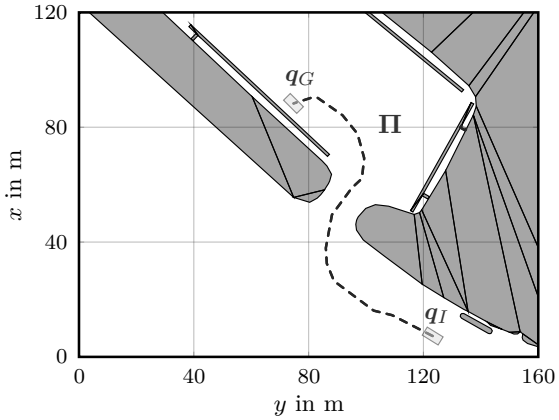
Tabelle 6.5.: Tuningparameter für den Online-Algorithmus.

6.6. Validierung des Online-Algorithmus zur sequenziellen Bewegungsplanung

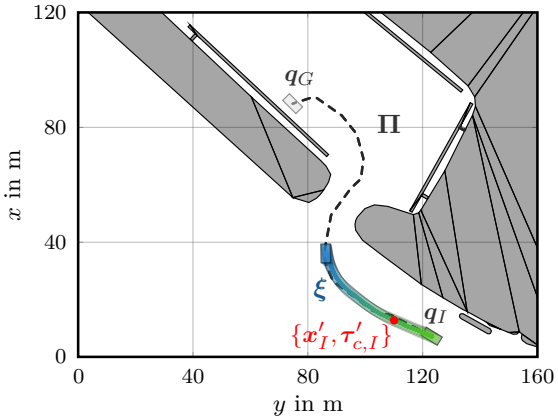
Der Online-Algorithmus zur sequenziellen Bewegungsplanung wurde in Abschnitt 5.5 vorgestellt. Er löst mehrere zeitlich aufeinanderfolgende Bewegungsplanungsprobleme unter Berücksichtigung der Rechenzeit und verbindet eine neu berechnete Trajektorie mit einer zuvor generierten Trajektorie, sodass keine Unstetigkeitsstellen in $\boldsymbol{x}$ und keine Sprünge in $\boldsymbol{\tau}_c$ entstehen. Für die Validierung wird das gleiche Szenario verwendet, welches in Abschnitt 6.5 beschrieben wurde. Die gewählten Tuningparameter sind identisch zu denen aus den Tabellen 6.2 und 6.3. Die im Rahmen dieses Simulationsexperimentes gewählten Rechenzeiten können der Tabelle 6.5 entnommen werden. Für das Zeitintervall t_ϵ wird aus Gründen der Darstellbarkeit der Ergebnisse ein Wert von 18 s verwendet. Die zeitliche Differenz zwischen zwei aufeinanderfolgenden Trajektorien entspricht demzufolge $t^p_{max} + t^m_{max} + t_\epsilon = 20\,\text{s}$.

In der Abbildung 6.18 sind die Ergebnisse der Pfad- und Bewegungsplanung zu unterschiedlichen Zeitpunkten des Online-Algorithmus gezeigt. Die Teilabbildung 6.18a illustriert das Ergebnis der Pfadplanung, welches sich auf das initiale Planungsproblem zum Zeitpunkt $t = 0$ bezieht. Die Start- und Zielkonfiguration sind mit $\boldsymbol{q}_I$ bzw. $\boldsymbol{q}_G$ gekennzeichnet und der resultierende Lösungspfad $\boldsymbol{\Pi}$ ist durch eine gestrichelte Linie dargestellt. Nach der Pfadplanung erfolgt die Bewegungsplanung und erzeugt eine Lösungstrajektorie $\boldsymbol{\xi}$, die in der Abbildung 6.18b als grün-blaue Kurve gezeigt ist. Aufgrund der beschränkten Pfadlänge von $L_{max} = 50$ endet die Trajektorie auf dem Pfad $\boldsymbol{\Pi}$ und erreicht nicht die Zielkonfiguration $\boldsymbol{q}_G$ der Pfadplanung. Die generierte Trajektorie beginnt zum Zeitpunkt $t = 0$ und erstreckt sich über eine Dauer von etwa 73 Sekunden. Der rot markierte Punkt kennzeichnet den Zeitpunkt $t = 20$, für den die sequenzielle Bewegungsplanung zum zweiten Mal ausgeführt wird, wobei der Anfangszustand $\boldsymbol{x}'_I$ sowie die Anfangsstellgröße $\boldsymbol{\tau}'_I$ dem Zustand $\boldsymbol{x}$ bzw. der Stellgröße $\boldsymbol{\tau}_c$ der Trajektorie $\boldsymbol{\xi}$ zu eben diesem Zeitpunkt entspricht. Die Teilabbildung 6.18c zeigt das Ergebnis der Pfadplanung. Die Pose von $\boldsymbol{x}'_I$ bildet hierbei die neue Startkonfiguration $\boldsymbol{q}'_I$ und die Zielkonfiguration bleibt unverändert. Der resultierende Lösungspfad $\boldsymbol{\Pi}'$ beginnt in $\boldsymbol{q}'_I$ und ist als eine gestrichelte Linie abgebildet. Zudem ist die zuvor generierte Trajektorie $\boldsymbol{\xi}$ durch eine orange Kurve dargestellt. In der Teilabbildung 6.18d ist das Ergebnis der Bewegungsplanung dargestellt. Die neu berechnete Trajektorie $\boldsymbol{\xi}'$ beginnt im Anfangszustand $\boldsymbol{x}'_I$ zum Zeitpunkt $t = 20$ und hat eine Dauer von 73.4 Sekunden. Es ist zu erkennen, dass der Endzustand der Trajektorie auf dem Lösungspfad der Pfadplanung liegt und dass der Abstand zur Zielkonfiguration im Vergleich zur Abbildung 6.18b reduziert wurde.

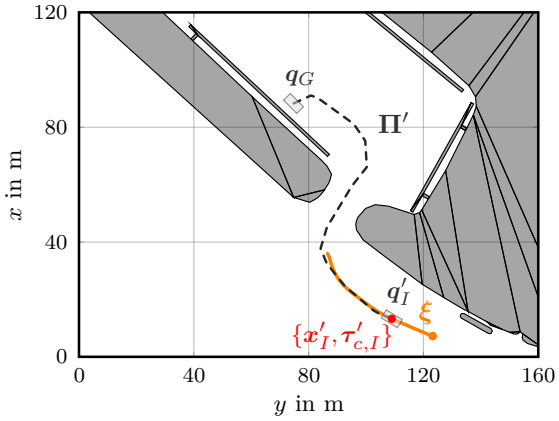
In der Abbildung 6.19 sind die zeitlichen Verläufe der Geschwindigkeit $\boldsymbol{\nu}$ sowie der Stellgröße $\boldsymbol{\tau}$ für die zwei Trajektorien $\boldsymbol{\xi}$ und $\boldsymbol{\xi}'$ dargestellt. Die blauen Kurven beziehen sich dabei auf die erste Trajektorie $\boldsymbol{\xi}$, die zum Zeitpunkt $t = 0$ beginnt, und die roten Kurven zeigen die Zustands- und Stellgrößenverläufe der zweiten Trajektorie $\boldsymbol{\xi}'$. Es ist zu erkennen, dass $\boldsymbol{\xi}'$ zum Zeitpunkt $t = 20$ beginnt und dass die zugehörigen Anfangswerte denen der ersten Trajektorie $\boldsymbol{\xi}$ entsprechen. Am Ende einer jeden Trajektorie gehen die Geschwindigkeiten und Stellgrößen gegen Null und das Zielgebiet der Bewegungsplanung nach (5.44) wird erreicht. Der zeitliche Verlauf von u' und X'_c der zweiten Trajektorie $\boldsymbol{\xi}'$ ist im Bereich von $t = 20$ bis $t = 40$ nahezu identisch zu dem der ersten Trajektorie $\boldsymbol{\xi}$. Bei der Quergeschwindigkeit und der Drehrate sind in diesem Zeitraum Unterschiede erkennbar. Der Grund dafür ist, dass für den Pfad $\boldsymbol{\Pi}'$ in der Umgebung von $\boldsymbol{q}'_I$



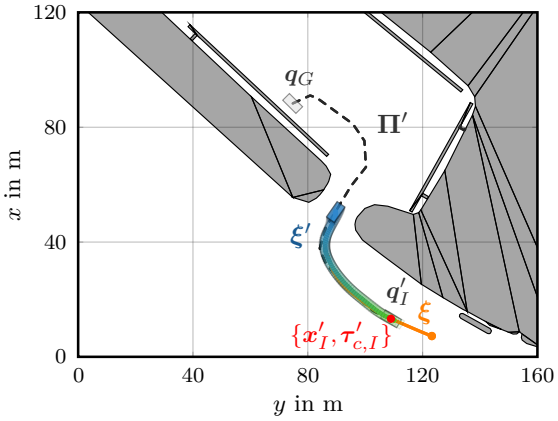
(a) Ergebnis der Pfadplanung für $t = 0$.



(b) Ergebnis der Bewegungsplanung für $t = 0$.

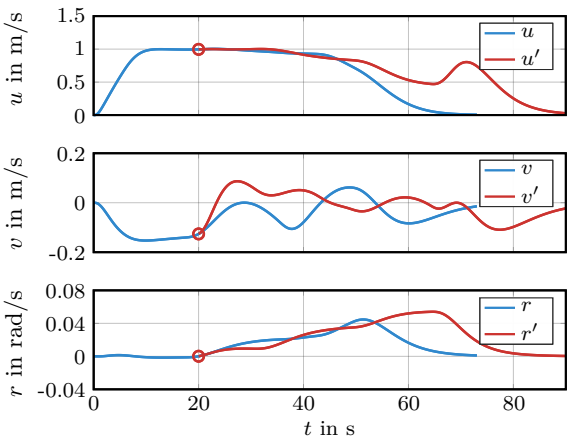


(c) Ergebnis der Pfadplanung für $t = 20$.

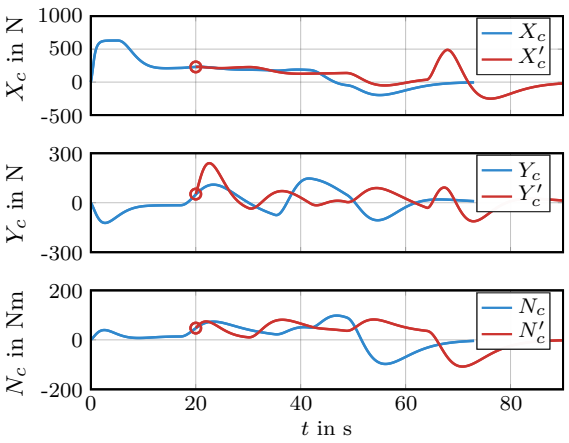


(d) Ergebnis der Bewegungsplanung für $t = 20$.

Abbildung 6.18.: Ergebnisse der Pfad- und Bewegungsplanung zu unterschiedlichen Zeitpunkten des Online-Algorithmus.

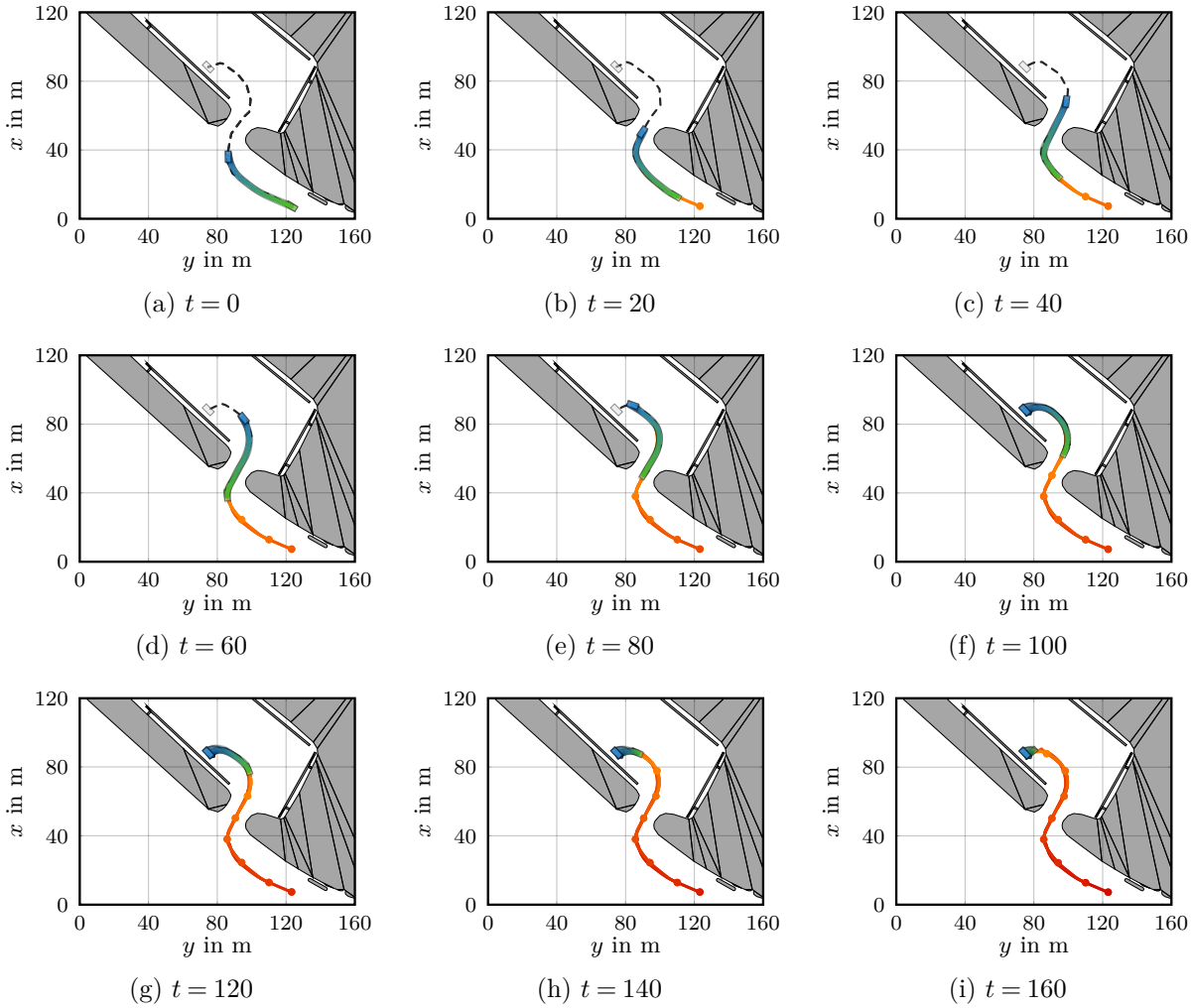


(a) Geschwindigkeit $\boldsymbol{\nu}$



(b) Stellgröße $\boldsymbol{\tau}_c$

Abbildung 6.19.: Zeitlicher Verlauf der Geschwindigkeiten und Stellgrößen von zwei aufeinanderfolgenden Trajektorien $\boldsymbol{\xi}$ (blau) und $\boldsymbol{\xi}'$ (rot).



(a) $t = 0$ (b) $t = 20$ (c) $t = 40$

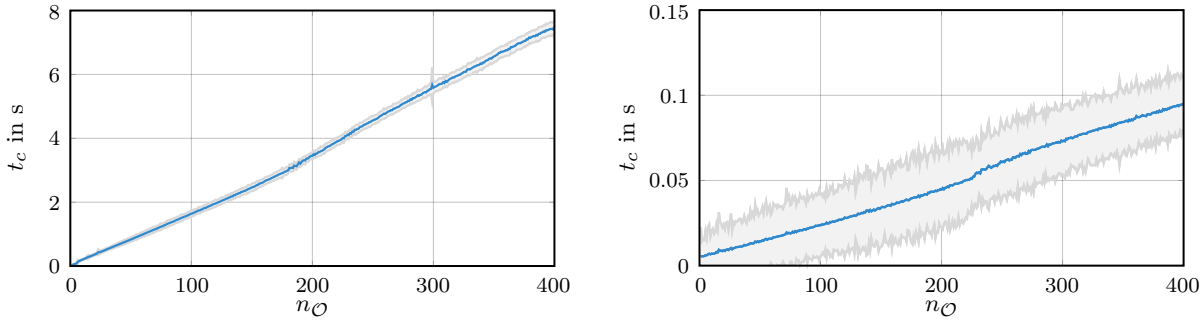
(d) $t = 60$ (e) $t = 80$ (f) $t = 100$

(g) $t = 120$ (h) $t = 140$ (i) $t = 160$

Abbildung 6.20.: Ergebnis der sequenziellen Bewegungsplanung zu neun unterschiedlichen Zeitpunkten des Online-Algorithmus.

eine besserer Lösungspfad als der zuvor berechnete Pfad $\mathbf{\Pi}$ gefunden wurde. Die Positionsdifferenz zwischen $\mathbf{\Pi}$ und $\mathbf{\Pi}'$ beträgt bis zu einem Meter und macht sich als initialer Querversatz bemerkbar, welcher vom Posenregler in eine entsprechende Querkraft übersetzt wird. Ab dem Zeitpunkt $t = 42$ beginnt für $\boldsymbol{\xi}$ der Abbremsvorgang, um das entsprechende Zielgebiet zu erreichen. Die zweite Trajektorie hat das zugehörige Zielgebiet zu diesem Zeitpunkt noch nicht erreicht. Für $\boldsymbol{\xi}'$ beginnt der Abbremsvorgang etwa ab dem Zeitpunkt $t = 64$.

Die Abbildung 6.20 zeigt das Ergebnis der sequenziellen Bewegungsplanung zu neun unterschiedlichen Zeitpunkten des Online-Algorithmus in einem Abstand von jeweils 20 Sekunden. In jeder Teilabbildung ist der geplante Pfad als gestrichelte Linie und die resultierende Lösung der Bewegungsplanung als grün-blaue Kurve dargestellt. Alle zuvor berechneten Trajektorien sind in einer Teilabbildung durch orange-rote Linien illustriert. Die Teilabbildungen 6.20a und 6.20b entsprechen dabei den Abbildungen 6.18b und 6.18d. Im Vergleich der einzelnen Teilgrafiken aus Abbildung 6.20 ist zu erkennen, dass die verbleibende Wegstrecke von der blau dargestellten finalen Pose einer Trajektorie zu der hellgrau gekennzeichneten Zielpose entlang des gestrichelten Pfades mit zunehmender Zeit minimiert wird. Wie in Abbildung 6.20f zu sehen ist, erreicht die Trajektorie zum Zeitpunkt $t = 100$ das Zielgebiet um $\boldsymbol{q}_G$. Ab diesem Zeitpunkt ist die Weglänge des Pfades kleiner als L_{max}. In den nachfolgenden Teilgrafiken 6.20g bis 6.20i ist zu erkennen, dass die räumliche Ausdehnung der Trajektorie kleiner wird. Alle weiteren Trajektorien, die generiert werden würden, entsprächen dem dynamischen Positionieren.



(a) $m_g = 1$ (b) $m_g = 10$

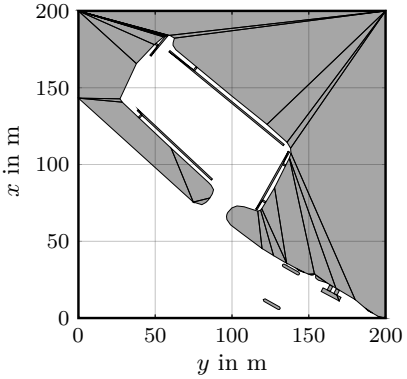
Abbildung 6.21.: Mittlere Rechenzeit t_c (blau) und 3-σ Standardabweichung (grau) für die Generierung einer LUT mit 16×10^6 Zellen in Abhängigkeit der Anzahl an Hindernissen $n_\mathcal{O}$ für verschiedene Werte von m_g.

6.7. Validierung der Lookup-Tabellenberechnung zur Approximation des Skalarfeldes $f_\varepsilon(x,y)$

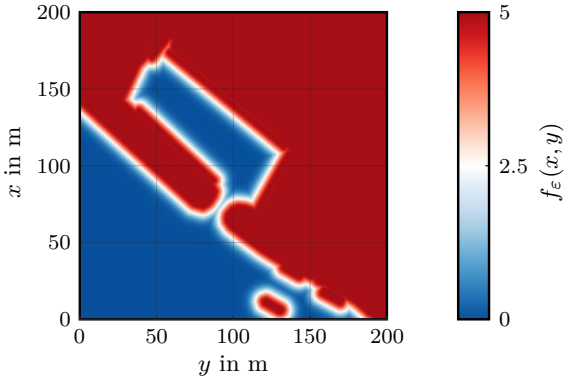
Wie in Abschnitt 5.3.2 beschrieben wurde, wird das Skalarfeld $f_\varepsilon(x,y)$ an zahlreichen diskreten Punkten berechnet und in einer LUT gespeichert. Da diese LUT von den Hindernissen $\mathcal{O}$ abhängt, muss sie für jedes Pfad- bzw. Bewegungsplanungsproblem erneut generiert werden. Der damit einhergehende Rechenaufwand führt dazu, dass weniger Rechenzeit für die Iterationen des RRT*-Algorithmus verbleibt. Die Anzahl an Hindernissen $n_\mathcal{O}$ hat einen signifikanten Einfluss auf die Rechenzeit, die für die Erstellung der LUT notwendig ist. Um die Rechenzeit zu reduzieren, lässt sich die Anzahl von Gitterpunkten, an denen $f_\varepsilon(x,y)$ bei gleicher Gitterauflösung ausgewertet wird, reduzieren. Die dazwischenliegenden Gitterpunkte werden durch eine bilineare Interpolation aufgefüllt. Der Abstand der Gitterpunkte, an denen $f_\varepsilon(x,y)$ berechnet wird, wird mit m_g bezeichnet, wobei $m_g = 1$ bedeutet, dass $f_\varepsilon(x,y)$ für jeden Gitterpunkt berechnet wird.

Um die zu erwartende Rechenzeit für die Erstellung einer LUT in Abhängigkeit von $n_\mathcal{O}$ und m_g bestimmen zu können, wird der folgende Versuch durchgeführt. Für eine bestimmte Anzahl von Hindernissen $n_\mathcal{O}$ wird 100-mal eine LUT mit einer Größe von 4000×4000 Zellen erzeugt und die Rechenzeit gemessen. Aus diesen 100 Messwerten wird der Mittelwert und die Standardabweichung für das entsprechende $n_\mathcal{O}$ bestimmt. Diese Messungen werden schließlich für $n_\mathcal{O} = 1$ bis $n_\mathcal{O} = 400$ wiederholt und der gesamte Versuch wird jeweils für $m_g = 1$ und $m_g = 10$ durchgeführt. Für die Hindernisse werden zufällige Dreiecke generiert. Das Ergebnis ist in der Abbildung 6.21 gezeigt. Hierbei stellt die blaue Kurve die mittlere Rechenzeit dar und der graue Bereich kennzeichnet die 3-σ Standardabweichung. Es ist zu erkennen, dass die Rechenzeit mit zunehmender Anzahl an Hindernissen steigt und das die Rechenzeit für $m_g = 10$ etwa um den Faktor 80 kleiner ist als für $m_g = 1$. Die Implementierung für das Erstellen der LUT basiert auf Multithreading, sodass die Berechnung von $f_\varepsilon(x,y)$ für unterschiedliche (x,y) gleichzeitig erfolgt. Demzufolge hängt die Rechenzeit auch von der Anzahl der verfügbaren CPU-Kerne ab.

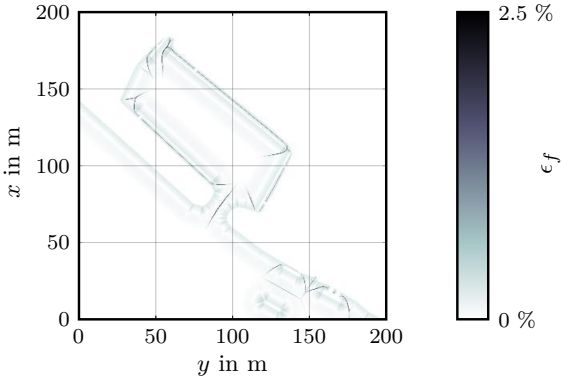
Als Resultat kann die Rechenzeit für Werte von $m_g > 1$ signifikant reduziert werden. Jedoch entstehen Approximationsfehler aufgrund der bilinearen Interpolation. Um diese Approximationsfehler für $m_g = 10$ untersuchen zu können, wird in einem weiteren Simulationsexperiment eine LUT für ein Beispielszenario aus SD100 generiert. Dieses enthält 51 konvexe Polygone und ist in der Abbildung 6.22a dargestellt. Die LUT wird zunächst für die Parameter $\alpha = 5$, $\beta = 0.02$, $g = 0.05$ und $m_g = 1$ erstellt. Das Ergebnis ist in der Abbildung 6.22b zu sehen, wobei blaue Bereiche Funktionswerte nahe Null darstellen und rote Flächen einem Funktionswert von $f_\varepsilon(x,y) = \alpha = 5$ entsprechen. Der Funktionswert dieser LUT an einer bestimmten Position wird im Folgenden mit $f_\varepsilon^{(1)}(x,y)$ bezeichnet, wobei der hochgestellte Index (1) kennzeichnet, dass diese LUT für $m_g = 1$ generiert wurde. Im Anschluss wird eine zweite LUT für $m_g = 10$ erzeugt,



(a) Beispielszenario mit 51 konvexen Polygonen.



(b) Skalarfeld $f_\varepsilon(x,y)$ für $\alpha = 5$, $\beta = 0.02$, $g = 0.05$ und $m_g = 1$.



(c) Relativer Approximationsfehler eines Skalarfeldes mit $m_g = 10$.

Abbildung 6.22.: Generierung des Skalarfeldes $f_\varepsilon(x,y)$ für ein gegebenes Beispielszenario.

dessen Funktionswert mit $f_\varepsilon^{(10)}(x,y)$ bezeichnet wird. Der relative Approximationsfehler von $f_\varepsilon^{(10)}(x,y)$ in Bezug auf $f_\varepsilon^{(1)}(x,y)$ wird mit

$$\epsilon_f = \left| \frac{f_\varepsilon^{(10)}(x,y) - f_\varepsilon^{(1)}(x,y)}{\alpha} \right| \tag{6.4}$$

berechnet und ist in der Abbildung 6.22c grafisch dargestellt. Der maximale Fehler liegt bei etwa 2.5%. Es ist zu erkennen, dass Approximationsfehler in der Nähe von Hindernissen entstehen, wo sich der Gradient des Skalarfeldes signifikant ändert. Zudem ist zu erkennen, dass sich an einigen Stellen wie beispielsweise dem Durchgangsbereich zum Hafenbecken große Approximationsfehler ergeben. In der Abbildung 6.23 ist die Schnittebene von $f_\varepsilon^{(1)}(x,y)$ und $f_\varepsilon^{(10)}(x,y)$ für $x = 100$ in einem Bereich von $y = 70$ bis $y = 80$ dargestellt. Die blaue Kurve stellt den Referenzwert dar und entspricht der generierten LUT für $m_g = 1$ und die rote Kurve zeigt die Approximation des Skalarfeldes mit $m_g = 10$. Bei einer Auflösung von $g = 0.05$ erfolgt die Berechnung von $f_\varepsilon^{(10)}(x,y)$ nach (5.16) alle 0.5 Meter, was ebenfalls anhand der roten Markierungen zu erkennen ist. Zwischen diesen Abtastpunkten werden die Funktionswerte der Approximation durch eine bilineare Interpolation erzeugt. Der resultierende Approximationsfehler ist zwischen den zwei Abtastpunkten $y = 73.5$ und $y = 74$ am größten und entspricht einem relativen Fehler von etwa 0.7%. Die abrupte Änderung des Gradienten des Skalarfeldes zwischen zwei Abtastpunkten ist der Grund für die streifenförmigen Approximationsfehler in Abbildung 6.22c. Angesichts der Tatsache, dass die Rechenzeit für das Erstellen der LUT mit $m_g = 10$ etwa um den Faktor 80 reduziert wurde, sind die relativen Approximationsfehler im Bereich bis 2.5% akzeptabel.

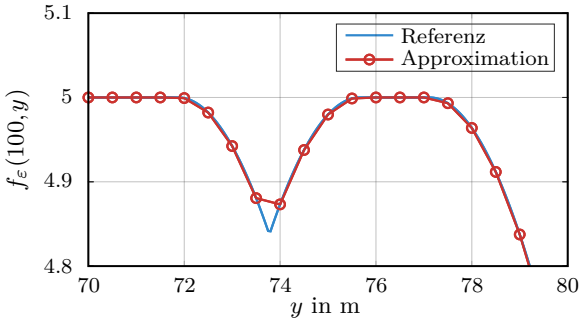


Abbildung 6.23.: Schnittebene des Skalarfeldes $f_\varepsilon(x, y)$ für $x = 100$.

6.8. Validierung der ExplorePath-Prozedur

Mithilfe der *ExplorePath*-Prozedur, die in Abschnitt 5.4.3 beschrieben wurde, wird innerhalb der Bewegungsplanung eine Trajektorie auf Basis eines gegebenen Pfades generiert. Die Sollpose für einen Posenregler wird anhand eines *Guidance*-Gesetzes berechnet, welches den Schnittpunkt eines Ellipsoids mit dem Pfad ermittelt. Die Radien des Ellipsoids werden in einem Vektor

$$\boldsymbol{r} = (r_{x,max},\ r_{y,max},\ r_\psi)^{\mathrm{T}} \tag{6.5}$$

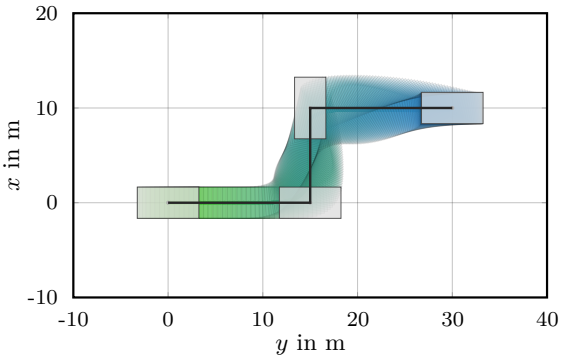
zusammengefasst. In diesem Abschnitt wird untersucht, welchen Einfluss $\boldsymbol{r}$ auf die generierte Trajektorie hat. Dafür wird zunächst ein Pfad mit vier Posen definiert.

$$\boldsymbol{\Pi} = \left\{ \begin{pmatrix} 0 \\ 0 \\ \pi/2 \end{pmatrix}, \begin{pmatrix} 0 \\ 15 \\ \pi/2 \end{pmatrix}, \begin{pmatrix} 10 \\ 15 \\ 0 \end{pmatrix}, \begin{pmatrix} 10 \\ 30 \\ \pi/2 \end{pmatrix} \right\}$$

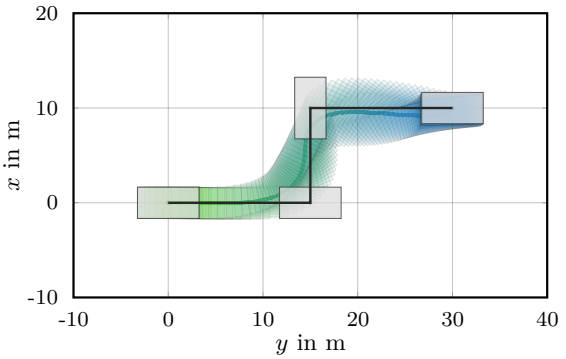
Im Anschluss wird die ExplorePath-Prozedur für drei verschiedene Radien

$$\boldsymbol{r}_1 = \begin{pmatrix} 5 \\ 3 \\ 0.3 \end{pmatrix},\ \boldsymbol{r}_2 = \begin{pmatrix} 10 \\ 6 \\ 0.6 \end{pmatrix},\ \boldsymbol{r}_3 = \begin{pmatrix} 15 \\ 6 \\ 0.9 \end{pmatrix}$$

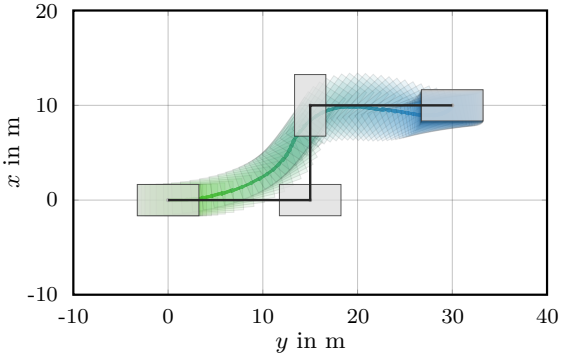
ausgeführt. Der Anfangszustand entspricht der ersten Pose des Pfades mit $\boldsymbol{\nu} = \mathbf{0}$ und $\boldsymbol{\tau} = \mathbf{0}$. Das Ergebnis des Simulationsexperiments ist in der Abbildung 6.24 dargestellt. Der gegebene Pfad $\boldsymbol{\Pi}$ ist mit einer schwarzen Linie gekennzeichnet, wobei in den Grafiken 6.24a bis 6.24c ebenfalls die zugehörigen geometrischen Fahrzeugformen abgebildet sind. In diesen Teilabbildungen ist die resultierende Trajektorie durch einen grün-blauen Verlauf dargestellt. Es ist zu erkennen, dass die Trajektorie für $\boldsymbol{r} = \boldsymbol{r}_1$ eine geringere Abweichung zum Pfad aufweist, als für $\boldsymbol{r} = \boldsymbol{r}_3$. Durch den kleineren Radius ergibt sich eine geringere Posendifferenz, was zu einer kleineren Stellgröße und somit zu einer geringeren Geschwindigkeit führt. Dies ist in Abbildung 6.24a am dichten Abstand der farbig gekennzeichneten geometrischen Fahrzeugformen entlang der Trajektorie zu erkennen, die in einem zeitlichen Abstand von einer Sekunde dargestellt sind. Im Vergleich der Teilabbildungen 6.24a bis 6.24c ist zu erkennen, dass die Dauer einer Trajektorie mit zunehmendem Radius $\boldsymbol{r}$ abnimmt. Die Abbildung 6.24d zeigt eine Zusammenfassung der Positionsverläufe der drei Trajektorien. Durch die Wahl von $\boldsymbol{r}$ wird neben der Geschwindigkeit



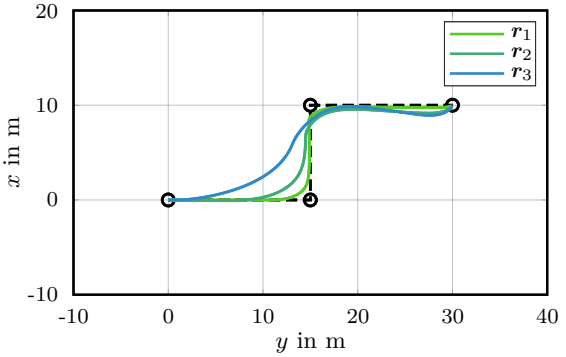
(a) Resultierende Trajektorie für $\boldsymbol{r} = \boldsymbol{r}_1$.



(b) Resultierende Trajektorie für $\boldsymbol{r} = \boldsymbol{r}_2$.



(c) Resultierende Trajektorie für $\boldsymbol{r} = \boldsymbol{r}_3$.



(d) Positionsverläufe der drei Trajektorien aus 6.24a, 6.24b und 6.24c.

Abbildung 6.24.: Ergebnis der *ExplorePath*-Prozedur in Abhängigkeit verschiedener Radien $\boldsymbol{r}$.

indirekt festgelegt, wie stark der gegebene Pfad geglättet wird.

7. Experimentelle Verfahrensapplikation

Das entwickelte Verfahren zur sequenziellen Bewegungsplanung autonomer Wasserfahrzeuge wird im Rahmen eines Feldexperiments für ein reales Schiff appliziert und erprobt. Zunächst wird der verwendete Versuchsträger vorgestellt und auf die Integration des Verfahrens in ein bestehendes Automationssystem eingegangen. Dieses Automationssystem setzt sich aus den Hauptmodulen für die Pfad- und Bewegungsplanung (***G**uidance*), der Zustandsschätzung und Umfelderkennung (***N**avigation*) sowie der Regelung und Antriebsallokation (***C**ontrol*) zusammen [116], weshalb im weiteren Verlauf dieser Arbeit von einem GNC-System gesprochen wird. Nach der Parametrisierung der sequenziellen Bewegungsplanung für den Versuchsträger wird die Versuchsumgebung und die Durchführung des Feldexperiments beschrieben. Anschließend werden die Ergebnisse des Feldexperiments analysiert und ausgewertet.

7.1. Vorstellung des Versuchsträgers

Als Versuchsträger wird das Vermessungs-, Wracksuch- und Forschungsschiff (VWFS) DENEB des Bundesamtes für Seeschifffahrt und Hydrographie (BSH) eingesetzt. Das Schiff, welches in Abbildung 7.1 zu sehen ist, hat eine Länge von 52 m und eine Breite von 11.4 m. Da es sich um ein Vermessungsschiff handelt, ist es ist mit entsprechend hochgenauer Sensorik ausgestattet. Für die durchgeführten Versuche wird ein inertiales Navigationssystem (INS) auf Basis eines faseroptischen Kreisels in Kombination mit mehreren GNSS-Empfängern mit Echtzeitkinematik (engl. *real-time kinematic*, RTK) verwendet, sodass eine Messgenauigkeit der Position im Zentimeterbereich erzielt wird. Die Messgenauigkeit des Heading-Winkels beträgt 0.01°. Alle notwendigen Bewegungsdaten stehen mit einer Abtastrate von 50 Hz zur Verfügung.

Das Schiff ist mit mehreren dieselelektrischen Antrieben ausgestattet. In der Abbildung 7.2 ist eine schematische Darstellung der Antriebskonfiguration gezeigt. Im Bug befindet sich ein Pump-Jet, der endlos um seine Hochachse gedreht und somit eine Schubkraft in jede beliebige Richtung der xy-Ebene erzeugen kann. Im Heck gibt es ein Heckstrahlruder, eine Hauptmaschine mit einem Festpropeller sowie eine Ruderanlage. Für das Feldexperiment werden alle Antriebsorgane verwendet. Aufgrund der Antriebskonfiguration ist das Schiff traversierfähig und kann

Abbildung 7.1.: Das Vermessungs-, Wracksuch- und Forschungsschiff DENEB.

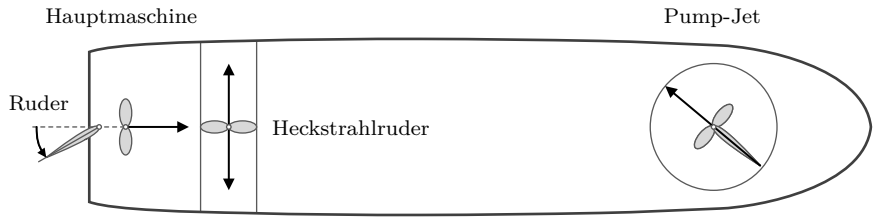


Abbildung 7.2.: Schematische Darstellung der Antriebskonfiguration der DENEB.

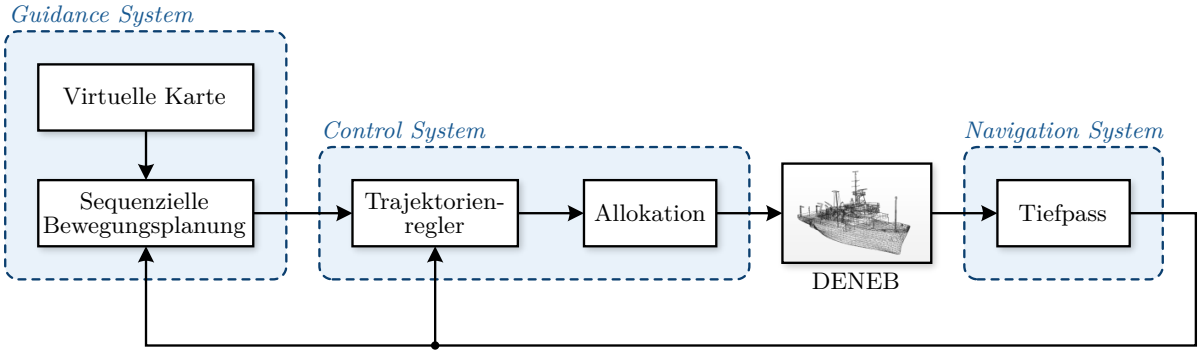


Abbildung 7.3.: Schematische Darstellung der GNC-Struktur für die experimentelle Applikation der sequenziellen Bewegungsplanung.

sich unabhängig in allen drei DoF bewegen. Die Stellwerte der einzelnen Antriebe werden über eine Hardwareschnittstelle mit einer Abtastrate von 2 Hz kommandiert.

7.2. Integration der sequenziellen Bewegungsplanung in ein bestehendes GNC-System

Für das Feldexperiment wird ein bestehendes GNC-System verwendet. Sowohl das *Navigation-* als auch das *Control-System* bleiben unverändert und im *Guidance-System* wird das in dieser Arbeit vorgestellte Verfahren zur sequenziellen Bewegungsplanung implementiert. Eine schematische Darstellung des applizierten GNC-Systems ist in der Abbildung 7.3 zu sehen. Auf das *Navigation-* und *Control-System* wird nicht im Detail eingegangen, da der Fokus dieser Arbeit auf dem *Guidance-System* liegt.

Das *Navigation-System* enthält im Wesentlichen einen Tiefpassfilter, mit dem die Messwerte des INS gefiltert werden. Die Zeitkonstante beträgt 0.8 s. Das *Control-System* besteht aus zwei Teilkomponenten: Trajektorienregler und Allokation. Mithilfe der Allokation werden die vom Trajektorienregler geforderten Kräfte und Momente durch eine quadratische Optimierung auf alle verfügbaren Antriebsorgane verteilt, wobei die Gesamtleistung minimiert wird. Im Trajektorienregler werden die Stellwerte der vom *Guidance-System* berechneten Trajektorie als Vorsteuerung verwendet. Zudem wird ein Störgrößenregler in Form einer linearen Zustandsrückführung mit integralem Verhalten und einem dynamischen *Anti-Windup* [117] eingesetzt, um Posenfehler zu kompensieren, die sich beispielsweise durch externe Störungen ergeben. Es sei erwähnt, dass ebenfalls modellprädiktive Regelungen (engl. *model predictive control* (MPC)) wie in [118] für die Trajektorienregelung geeignet sind, da der zukünftige Verlauf der Sollgrößen durch die geplante Trajektorie vorgegeben ist. Im *Guidance-System* berechnet die sequenzielle Bewegungsplanung auf Basis des aktuellen Bewegungszustands, einer gegebenen Zielpose sowie einer gegebenen vir-

tuellen Karte eine kollisionsfreie Trajektorie. Auf die virtuelle Karte wird in den nachfolgenden Abschnitten eingegangen. Nach Algorithmus 23 wird der aktuelle Bewegungszustand $\boldsymbol{x}$, welcher durch das *Navigation-System* berechnet wird, nur während des Starts oder beim Zurücksetzen der Bewegungsplanung verwendet, um den Anfangszustand $\boldsymbol{x}_I$ zu initialisieren. Demzufolge ist der Pfeil vom *Navigation-System* zum *Guidance-System* in Abbildung 7.3 nicht als eine dynamische Rückführung anzusehen, wie es beim Regelkreis mit dem Trajektorienregler der Fall ist.

Der rechentechnische Versuchsaufbau besteht aus zwei Industrie-PCs, die jeweils mit einer Intel Core i9-13900E CPU und 32 GB Arbeitsspeicher ausgestattet sind und auf denen die einzelnen Unterfunktionen des GNC-Systems verteilt werden. Auf dem ersten Rechner ist das *Navigation-System* und die Allokation sowie die Schnittstelle zum Prozess implementiert. Das *Guidance-System* und der Trajektorienregler laufen auf dem zweiten Industrie-PC. Im Gegensatz zur simulativen Verfahrensanalyse aus Kapitel 6, bei der die sequenzielle Bewegungsplanung auf allen 24 Kernen der CPU ausgeführt wurde, werden dem *Guidance-System* im Rahmen des Feldexperiments nur 16 CPU-Kerne zugewiesen, da die Trajektorienregelung und weitere Systemkomponenten ebenfalls Rechenressourcen benötigen.

7.3. Parametrisierung der sequenziellen Bewegungsplanung

Die Parametrisierung der sequenziellen Bewegungsplanung erfolgt analog zu der in Abschnitt 6.1 beschriebenen Vorgehensweise spezifisch für den Versuchsträger DENEB. Die geometrische Form des realen Schiffes wird durch ein Rechteck mit einer Länge von 54 m und einer Breite von 12 m beschrieben. Für die Approximation einer beliebigen Kurve durch eine endliche Menge von Liniensegmenten innerhalb des Kollisionstests der Bewegungsplanung wird $\Delta p_{max} = 0.1$ und $\Delta\psi_{max} = 5^\circ$ gesetzt. Diese Werte sind identisch zu denen der Simulationsexperimente. Wie in Appendix A.3.2 beschriebenen ist, wird die reale geometrische Fahrzeugform vergrößert, um einen sicheren Kollisionstest zu gewährleisten. Das resultierende Fahrzeugpolygon wird demzufolge als ein Rechteck mit der Länge von etwa 55.1 m und Breite von etwa 17.1 m repräsentiert.

Im *Control-System* des verwendeten GNC-Systems wird der kommandierte Kräfte-Momente-Vektor $\boldsymbol{\tau}_c$ beschränkt. Die maximale Längs- und Querkraft ist mit $X_{sat} = \pm 12 \times 10^3$ bzw. $Y_{sat} = \pm 10 \times 10^3$ gegeben und das maximale Moment beträgt $N_{sat} = \pm 150 \times 10^3$. Damit externe Störungen und Modellungenauigkeiten mit dem *Control-System* ausgeregelt werden können, wird für die Bewegungsplanung jedoch nur eine Untermenge dieses realisierbaren Kräfte-Momenten-Raums benutzt. Dazu werden 80% der maximalen Längskraft, 50% der maximalen Querkraft und 60% des maximalen Momentes als Stellgrößenbeschränkung für die Bewegungsplanung gewählt. Das Bewegungsmodell für den Versuchsträger entspricht den Gleichungen (5.1)-(5.3), wobei

$$\boldsymbol{f}(\boldsymbol{\nu}) = \underbrace{\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{17} \\ f_{21} & f_{22} & \cdots & f_{27} \\ f_{31} & f_{32} & \cdots & f_{37} \end{pmatrix}}_{\boldsymbol{F}} \underbrace{\begin{pmatrix} u & v & r & vr & u^3 & v^3 & r^3 \end{pmatrix}^{\mathrm{T}}}_{\boldsymbol{n}(\boldsymbol{\nu})} \tag{7.1}$$

ist. Tabelle 7.1 fasst alle Parameter des Bewegungsmodells zusammen. Basierend auf dem Bewegungsmodell erfolgt die Parametrisierung der *ExplorePath*-Prozedur auf die gleiche Weise, wie in Abschnitt 6.1.3 beschrieben wurde. Die Zeitkonstanten für das Eingangsfilter zur Vermeidung sprunghafter Stellgrößen werden auf $T_{f1} = T_{f2} = T_{f3} = 4$ gesetzt. Anschließend werden vier dreifach-Pole für die Dynamik des geschlossenen Posenregelkreises vorgegeben.

$$\left\{(-0.6)^3,\ (-0.3)^3,\ (-0.05)^3,\ (-0.04)^3\right\}$$

Mit dem *pole-placement*-Verfahren wird die Verstärkungsmatrix

$$\boldsymbol{K}_\eta = \begin{pmatrix} \boldsymbol{K}_{\eta,1} & \boldsymbol{K}_{\eta,2} & \boldsymbol{K}_{\eta,3} & \boldsymbol{K}_{\eta,4} \end{pmatrix} \tag{7.2}$$

Parameter	Wert
$\boldsymbol{F}$	$\begin{pmatrix} -0.0019 & 0 & 0 & 1.5328 & -0.0008 & 0 & 0 \\ 0 & -0.0121 & -0.1886 & 0 & 0 & -0.1036 & 0 \\ 0 & -0.0005 & -0.0197 & 0 & 0 & 0 & -71.37 \end{pmatrix}$
$\boldsymbol{B}$	$\begin{pmatrix} 1.9854 & 0 & 0 \\ 0 & 1.3247 & 0 \\ 0 & 0 & 0.005 \end{pmatrix} \times 10^{-6}$
$\{T_X,\ T_Y,\ T_N\}$	$\{2.0,\ 2.0,\ 2.0\}$
$\{X_{max},\ Y_{max},\ N_{max}\}$	$\{9600,\ 5000,\ 90000\}$
$\{X_{min},\ Y_{min},\ N_{min}\}$	$\{-9600,\ -5000,\ -90000\}$

Tabelle 7.1.: Modellparameter für den Versuchsträger DENEB.

Parameter	Wert
$\{T_{f1},\ T_{f2},\ T_{f3}\}$	$\{4.0,\ 4.0,\ 4.0\}$
$\boldsymbol{K}_{\eta,1}$	$\begin{pmatrix} 0.36 & 0 & 0 \\ 0 & 0.36 & 0 \\ 0 & 0 & 0.36 \end{pmatrix} \times 10^{-3}$
$\boldsymbol{K}_{\eta,2}$	$\begin{pmatrix} 18 & 0 & 0 \\ 0 & 18 & 0 \\ 0 & 0 & 18 \end{pmatrix} \times 10^{-3}$
$\boldsymbol{K}_{\eta,3}$	$\begin{pmatrix} 263 & 0 & 0 \\ 0 & 263 & 0 \\ 0 & 0 & 263 \end{pmatrix} \times 10^{-3}$
$\boldsymbol{K}_{\eta,4}$	$\begin{pmatrix} 990 & 0 & 0 \\ 0 & 990 & 0 \\ 0 & 0 & 990 \end{pmatrix} \times 10^{-3}$
$r_{x,max}$	50
$r_{y,max}$	9
r_{ψ}	$25°$
$r_{p,min}$	6
$\{\Delta x_{th},\ \Delta y_{th},\ \Delta\psi_{th}\}$	$\{2,\ 2,\ 5°\}$
$\{u_{th},\ v_{th},\ r_{th}\}$	$\{0.05,\ 0.05,\ 0.001\}$
$\{X_{th},\ Y_{th},\ N_{th}\}$	$\{300,\ 300,\ 1000\}$

Tabelle 7.2.: Tuningparameter für die *ExplorePath*-Prozedur.

auf der Grundlage des linearisierten Zustandsraummodells (5.66) berechnet. Die resultierenden Teilmatrizen sowie zusätzliche Tuningparameter der *ExplorePath*-Prozedur sind in der Tabelle 7.2 zu finden.

In Tabelle 7.3 sind alle weiteren Parameter zusammengefasst, die für das Feldexperiment verwendet werden. Für die Kostenfunktion werden die Gewichtungsfaktoren $w_v = 0$ und $w_\alpha = 1$ gewählt, um die Vorwärtsfahrt gegenüber der Rückwärtsfahrt zu bevorzugen. Die maximale Schrittweite innerhalb des RRT*-Algorithmus wird entsprechend der Simulationsergebnisse aus Abschnitt 6.4.3 mit $\lambda_{max} = \infty$ festgelegt. Für den Online-Algorithmus der sequenziellen Bewegungsplanung nach Algorithmus 23 werden die Rechenzeiten der Pfad- und Bewegungsplanung auf $t^p_{max} = 1.4$ bzw. $t^m_{max} = 3.3$ beschränkt. Jedoch werden während der *SolveInitialProblem*-Prozedur des Online-Algorithmus kleinere Rechenzeiten mit $t^p_{max} = 0.5$ und $t^m_{max} = 1.5$ verwendet, damit die erste Lösung des Planungsproblems bereits nach etwa zwei Sekunden zur Verfügung steht. Das Sampling-Gebiet, welches für die Pfadplanung verwendet wird, entspricht einem orientierten Quader und wird, wie in Abschnitt 5.3.3 beschrieben wurde, auf Basis der Start- und Zielkonfigurationen berechnet. Das Sampling erfolgt nach (5.28) mit $b_x = b_y = 100$.

Parameter	Wert	Beschreibung
T_s	0.5	Abtastzeit der Lösungstrajektorie.
n^p_{max}	200	Maximale Anzahl an Knoten in $\mathcal{T}$ während der Pfadplanung.
n^m_{max}	200	Maximale Anzahl an Knoten in $\mathcal{T}$ während der Bewegungsplanung.
$\{b_x, b_y\}$	$\{100,\ 100\}$	Tuningparameter für die Größe des Suchgebietes.
w_ψ	3	Wichtungsfaktor für ψ im Kostenterm c_ρ.
w_v	0	Wichtungsfaktor für Querbewegungen im Kostenterm c_v.
w_α	1	Wichtungsfaktor für Quer- und Rückwärtsbewegungen in c_v.
w_β	1	Stauchungsfaktor der Gewichtungsfunktion für die Rückwärtsfahrt.
α	5	Wichtungsfaktor für das Skalarfeld f_ε im Kostenterm c_μ.
β	0.0015	Exponentieller Abfall des Skalarfeldes f_ε.
g	0.5	Auflösung der LUT des Skalarfeldes.
m_g	10	Modulofaktor, der angibt, für welche LUT-Einträge f_ε berechnet wird.
λ_{max}	∞	Maximale Schrittweite innerhalb des RRT*-Algorithmus.
g_{max}	100	Periode für das *goal sampling*.
k_{max}	10^6	Anzahl der im Vorfeld generierten Pseudozufallszahlen.
D_{xy}	50	Dimension einer Sampling-Box in Positionskoordinaten.
D_ψ	1	Dimension einer Sampling-Box in Winkelkoordinaten.
L_{max}	200	Maximale Weglänge eines Teilpfades für die Bewegungsplanung.
$\{\boldsymbol{\kappa}^b_1, \boldsymbol{\kappa}^b_2\}$	$\left\{\binom{28}{0}, \binom{-26}{0}\right\}$	Körperfeste Punkte an denen der Kostenterm c_μ ausgewertet wird.
t^p_{max}	1.4	Maximale Rechenzeit für die Pfadplanung.
t^m_{max}	3.3	Maximale Rechenzeit für die Bewegungsplanung.
t_ϵ	0.2	Zusätzliches Zeitintervall für die Wahl neuer Initialwerte.

Tabelle 7.3.: Gewählte Tuningparameter für das Feldexperiment.

7.4. Versuchsumgebung und Durchführung des Feldexperimentes

Das Feldexperiment wurde im Marinehafen des Marinestützpunktes Hohe Düne in Rostock durchgeführt. Dieses Areal ist als Sperrgebiet gekennzeichnet und erlaubt somit das Durchführen von Versuchen ohne externe Verkehrsteilnehmer. Für das Versuchsszenario wird eine virtuelle Karte verwendet, die innerhalb der für das Schiff befahrbaren Wasserflächen liegt. Das ist notwendig, da präzise Kartendaten der realen Hafenstruktur sowie geometrische Informationen zu weiteren Hindernissen wie festgemachten Schiffen nicht zur Verfügung stehen. Mithilfe der virtuellen Karte wird die Gefahr einer Kollision des Schiffes mit realen Hindernissen vermieden. Die Abbildung 7.4 zeigt die Versuchsumgebung für das Feldexperiment. Die realen Land- und Wasserflächen sind beige bzw. blau gekennzeichnet und die virtuelle Karte ist mit dunklen Flächen dargestellt. Sie besteht aus 27 konvexen Polygonen und entspricht einem Datensatz aus SD100, der auf $400\,\mathrm{m} \times 400\,\mathrm{m}$ vergrößert wurde.

Eine schematische Darstellung der Versuchsdurchführung ist in der Abbildung 7.5 gezeigt. Vor dem Beginn eines Versuchs wird das Schiff manuell vom verantwortlichen Schiffsführer gesteuert. Er fährt das Schiff aus dem nord-östlichen Becken des Marinehafens heraus und fährt zur östliche Einfahrt der virtuellen Karte, was durch die grau-gestrichelte Linie gekennzeichnet ist. Sobald das Schiff diese Einfahrt erreicht, setzt der Schiffsführer alle Stellgrößen auf Null, damit eine Umschaltung auf das vorgestellte GNC-System durchgeführt werden kann. Nach dem Umschalten erhält das GNC-System den Zugriff auf die Antriebsorgane des Schiffes und der Versuch wird gestartet. Der Anfangszustand $\boldsymbol{x}_I$, welcher in der Abbildung 7.5 grün gekennzeichnet ist, entspricht dem aktuellen Bewegungszustand. Die Zielpose $\boldsymbol{q}_G$ ist blau dargestellt und entspricht einer vorgegebenen virtuellen Anlegeposition. Sowohl die Hindernisse, als auch die Zielpose bleiben während eines Versuchs unverändert.

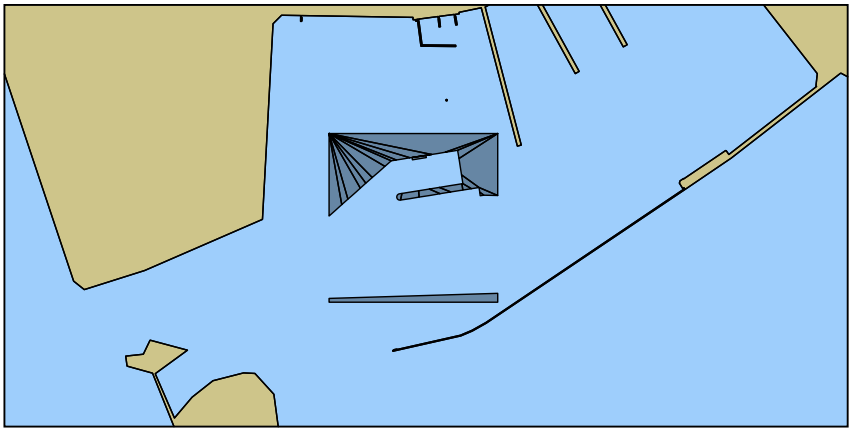


Abbildung 7.4.: Versuchsumgebung mit einer virtuellen Karte.

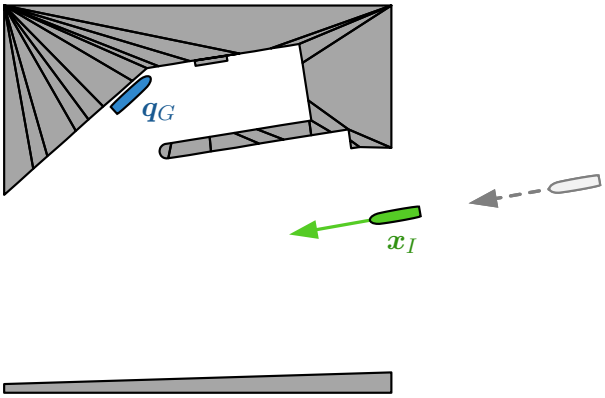
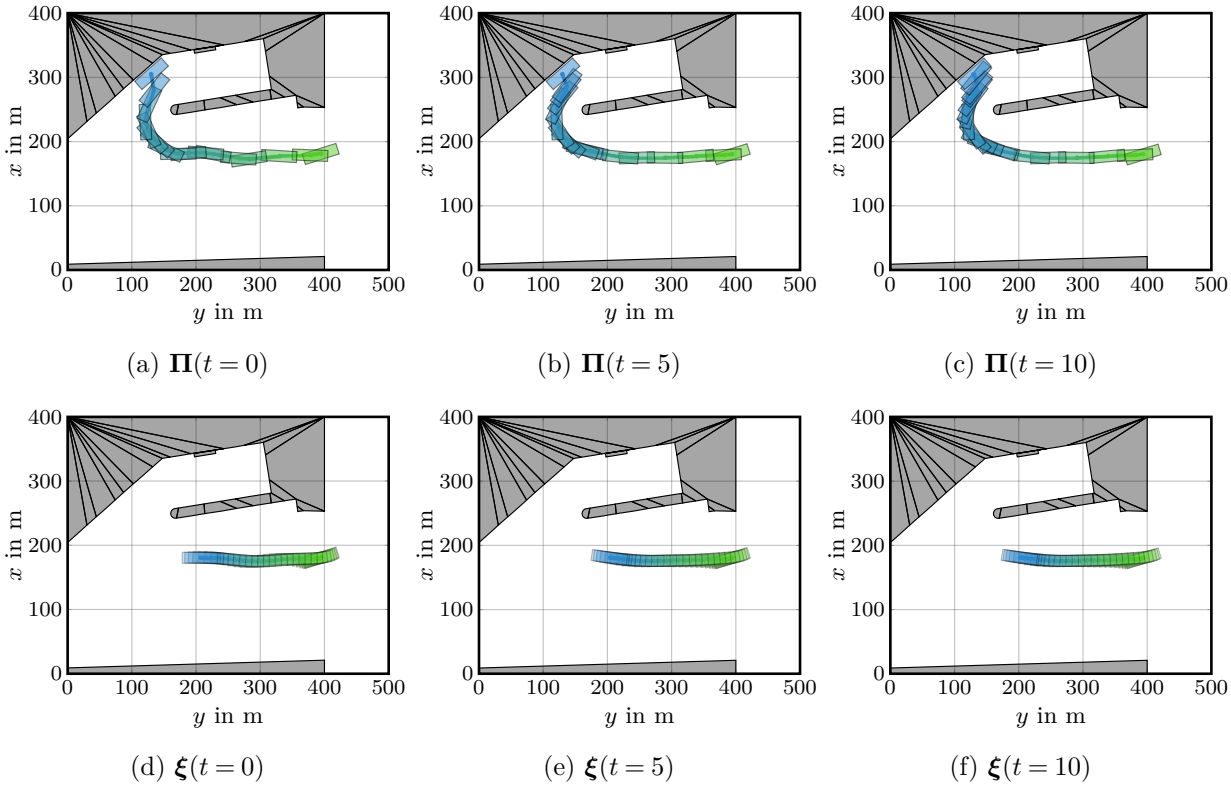


Abbildung 7.5.: Schematische Darstellung der Versuchsdurchführung zur Bewegungsplanung vom Anfangszustand $\boldsymbol{x}_I$ (grün) zur Zielpose $\boldsymbol{q}_G$ (blau).

7.5. Ergebnisse des Online-Algorithmus zur sequenziellen Bewegungsplanung

Dieser Abschnitt analysiert die Ergebnisse des Feldexperiments im Hinblick auf den Online-Algorithmus. Die Abbildung 7.6 zeigt den Lösungspfad $\boldsymbol{\Pi}$ sowie die zugehörige Lösungstrajektorie $\boldsymbol{\xi}$ zu mehreren aufeinanderfolgenden Zeitpunkten nach dem Starten des Versuchs. Die Lösungspfade sind in der oberen Zeile und die Trajektorien in der unteren Zeile abgebildet. Die Datenaufzeichnung der Trajektorie ist auf 500 Datenpunkte beschränkt, was einer zeitlichen Ausdehnung von 250 s entspricht. Für die grafische Darstellung einer Trajektorie in der xy-Ebene wird die geometrische Fahrzeugform in Abständen von zehn Sekunden gezeigt.

Im Nachfolgenden kennzeichnet der Zeitpunkt $t = 0$ die erste Lösung der sequenziellen Bewegungsplanung nach dem Starten des Versuchs. Der zugehörige Lösungspfad $\boldsymbol{\Pi}(t = 0)$ ist in der Abbildung 7.6a dargestellt. Er verbindet die grün markierte Anfangspose $\boldsymbol{q}_I = (181.18,\ 395.16,\ -1.88)^{\mathrm{T}}$ mit der blauen Zielpose $\boldsymbol{q}_G = (305,\ 130,\ 0.85)^{\mathrm{T}}$. Die Anfangsgeschwindigkeit beträgt $\boldsymbol{\nu}_I = (0.115,\ -0.0687,\ 0.0003)^{\mathrm{T}}$ und der initiale Kräfte-Momenten-Vektor sowie die initiale Stellgröße ist mit $\boldsymbol{\tau} = \boldsymbol{\tau}_c = \mathbf{0}$ gegeben. Es ist zu erkennen, dass der Pfad im Anfangsbereich signifikante Änderungen des Heading-Winkels sowie der Position in x-Richtung aufweist. Der Hauptgrund dafür ist die geringe Rechenzeit der Pfadplanung von 0.5 Sekunden während der ersten Iteration des Online-Algorithmus. Der Kostenwert des Pfades beträgt $c = 498.0216$. Die zugehörige Trajektorie ist in der Abbildung 7.6d dargestellt und zeigt ebenfalls Änderungen des Heading-Winkels. Sie wird dem *Control-System* übergeben und die Berechnung einer zukünftigen Trajektorie wird gestartet, wobei die Rechenzeit der Pfad- und Bewegungs-



(a) $\mathbf{\Pi}(t=0)$ (b) $\mathbf{\Pi}(t=5)$ (c) $\mathbf{\Pi}(t=10)$

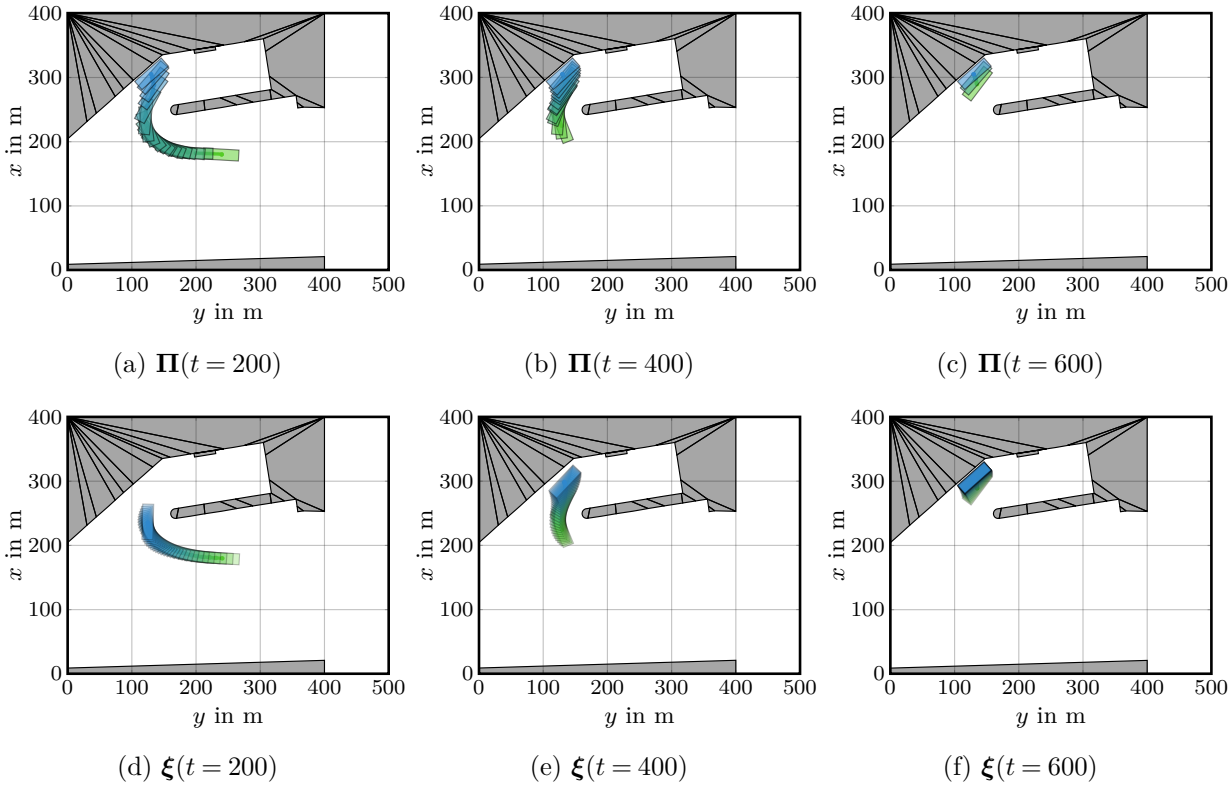
(d) $\boldsymbol{\xi}(t=0)$ (e) $\boldsymbol{\xi}(t=5)$ (f) $\boldsymbol{\xi}(t=10)$

Abbildung 7.6.: Lösungspfad $\mathbf{\Pi}$ (oben) und zugehörige Lösungstrajektorie $\boldsymbol{\xi}$ (unten) zu mehreren Zeitpunkten nach dem Starten der sequenziellen Bewegungsplanung.

planung entsprechend der Tabelle 7.3 mit $t^p_{max} = 1.4$ und $t^m_{max} = 3.3$ festgelegt ist.

Ab dem Zeitpunkt $t = 5$ steht dem *Control-System* die nächste Lösung des Online-Algorithmus zur Verfügung. Der berechnete Pfad sowie die zugehörige Trajektorie sind in den Teilgrafiken 7.6b und 7.6e abgebildet. Im Vergleich zur Lösung bei $t = 0$ ist eine signifikante Glättung des Pfades zu erkennen. Durch die vorgestellte Warmstart-Prozedur und der anschließenden Pfadplanung über einen Zeitraum von etwa 1.4 Sekunden wird ein neuer Lösungspfad mit einem geringeren Kostenwert von $c = 492.3775$ gefunden. Dabei ist zu erwähnen, dass die Anfangspose dieses zweiten Lösungspfades nicht identisch zu der des vorangegangenen Planungsproblems ist. Die Differenz beträgt etwa 0.5 m und 0.1°. In den Abbildungen 7.6c und 7.6f sind der Pfad bzw. die Trajektorie der dritten Lösung des Online-Algorithmus dargestellt, welche dem *Control-System* zum Zeitpunkt $t = 10$ zur Verfügung stehen. Der Kostenwert des Pfades beträgt $c = 490.8710$. Gegenüber den Abbildungen 7.6b und 7.6e ist keine signifikante Verbesserung erkennbar. Anhand der farbig gekennzeichneten Rechtecke ist zu sehen, dass der Heading-Winkel des Fahrzeuges zu Beginn eines Pfades bzw. einer Trajektorie angepasst wird, damit das Fahrzeug im Anschluss eine geradlinige Strecke in Vorwärtsfahrt zurücklegen kann und gleichzeitig der Zielpose näher kommt. Diese Eigenschaft resultiert aus den gewählten Gewichtungsfaktoren $w_\alpha = 1$ und $w_\beta = 1$. Die Abbildungen 7.6d-7.6f zeigen eine Glättung der Trajektorie mit zunehmender Anzahl an durchgeführten Iterationen.

In der Abbildung 7.7 sind die Lösungspfade und -trajektorien des Online-Algorithmus zu drei weiteren Zeitpunkten dargestellt. Zum Zeitpunkt $t = 200$ befindet sich die aktuelle Startpose der Pfadplanung bereits bei $\boldsymbol{q}_I = (179.93,\ 239.89,\ -1.51)^{\mathrm{T}}$. Durch das fortlaufende Sampling enthält der Pfad $\mathbf{\Pi}(t = 200)$ im Bereich des Manövrierens um die Hafenmauer eine höhere Dichte an Posen als es im Vergleich zu den vorherigen Zeitpunkten in der Abbildung 7.6 der Fall ist. Die Teilabbildungen 7.7b und 7.7e bzw. 7.7c und 7.7f zeigen jeweils den Pfad bzw. die Trajektorie zu den Zeitpunkten $t = 400$ und $t = 600$. Es ist zu erkennen, wie die Hafenmauer umfahren wird und dabei gleichzeitig ein Pfad berechnet wird, welcher einen möglichst großen



(a) $\mathbf{\Pi}(t = 200)$ (b) $\mathbf{\Pi}(t = 400)$ (c) $\mathbf{\Pi}(t = 600)$

(d) $\boldsymbol{\xi}(t = 200)$ (e) $\boldsymbol{\xi}(t = 400)$ (f) $\boldsymbol{\xi}(t = 600)$

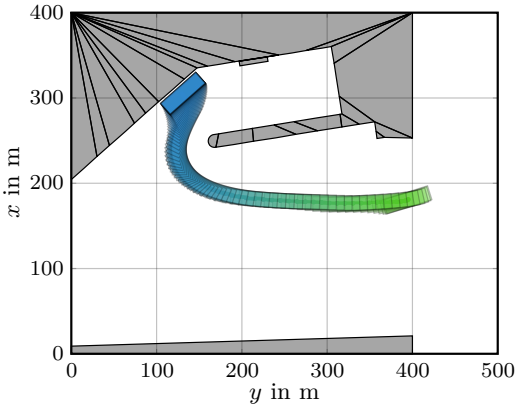
Abbildung 7.7.: Lösungspfad $\mathbf{\Pi}$ (oben) und zugehörige Lösungstrajektorie $\boldsymbol{\xi}$ (unten) zu verschiedenen Zeitpunkten während des Versuchs.

Abstand zu Hindernissen beibehält. Um die Zielkonfiguration zu erreichen, erfolgt etwa ab dem Zeitpunkt $t = 600$ ein Traversiermanöver nach Backbord. Dabei wird das vorgegebene Ziel durch eine kollisionsfreie Trajektorie erreicht, ohne dass es zum Überschwingen kommt.

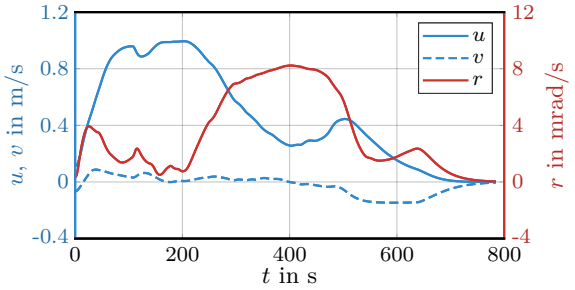
7.5.1. Analyse der geplanten Trajektorie

In Abbildung 7.8 ist der Verlauf der Zustands- und Stellgrößen der gesamten Trajektorie über einen Zeitraum von 784.5 s zu sehen. In der Teilabbildung 7.8a ist der Positionsverlauf vom grün markierten Anfangszustand zur blau gekennzeichneten Zielpose dargestellt. Er entspricht dem zeitlichen Verlauf der Sollpose für das *Control-System*. Die Teilabbildungen 7.8b-7.8d geben die Geschwindigkeit $\boldsymbol{\nu}$, den Kräfte-Momenten-Vektor $\boldsymbol{\tau}$ sowie die Stellgröße $\boldsymbol{\tau}_c$ wieder. Zu Beginn erfolgt eine Beschleunigung des Fahrzeuges auf eine maximale Längsgeschwindigkeit von etwa 1 m/s. Gleichzeitig wird der Heading-Winkel durch eine positive Drehrate angepasst. Ab dem Zeitpunkt $t \approx 200$ steigt die Drehrate an und erreicht bei $t \approx 400$ ihren Maximalwert. Diese Phase entspricht der Drehung über Steuerbord für das Umfahren der Hafenmauer. Zum Zeitpunkt $t = 400$ ist die Längsgeschwindigkeit reduziert, da der Abstand des Fahrzeuges zu Hindernissen kleiner als der Radius $r_{x,max} = 50$ des *Guidance*-Gesetzes der *ExplorePath*-Prozedur ist. Ab $t \approx 500$ wird eine Quergeschwindigkeit nach Backbord aufgebaut, die nach etwa 50 Sekunden einen Wert von $v = -0.14$ erreicht. Das Schiff traversiert an die virtuelle Pier. Bei $t = 640$ beginnt das Auffangen der Schiffsbewegung und die Geschwindigkeitskomponenten gehen zu null. Der Versuch wird zum Zeitpunkt $t = 784.5$ beendet.

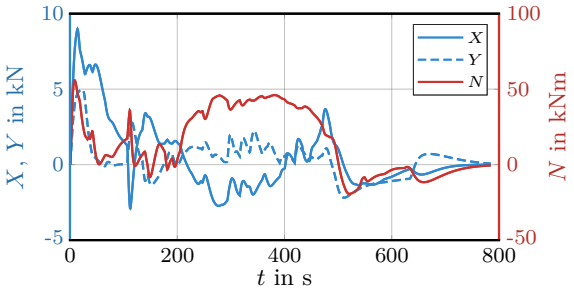
Im Vergleich der Abbildungen 7.8c und 7.8d ist kein signifikanter Unterschied zwischen dem Kräfte-Momenten-Vektor $\boldsymbol{\tau}$ und der Stellgröße $\boldsymbol{\tau}_c$ zu erkennen, da die Zeitkonstanten des Kraftmodells aus Gleichung (5.6) mit $T_X = T_Y = T_N = 2$ wesentlich kleiner als das abgebildete Zeitintervall sind. Im *Control-System* wird $\boldsymbol{\tau}_c$ als Vorsteuerung verwendet. Es sind deutliche Änderungen der Stellgröße über die gesamte Versuchszeit zu erkennen, was auf die bereits in Abschnitt



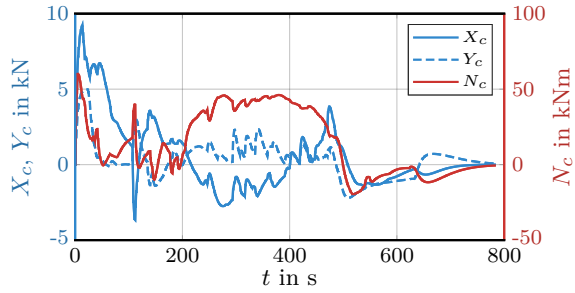
(a) Posenverlauf der Trajektorie.



(b) Geschwindicgkeit $\boldsymbol{\nu}$



(c) Kräfte-Momenten-Vektor $\boldsymbol{\tau}$

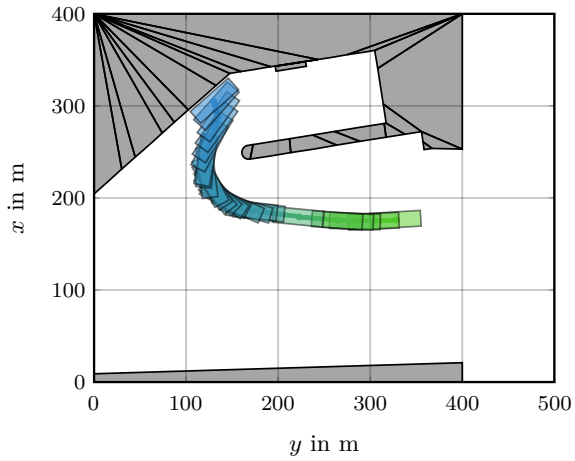


(d) Stellgröße $\boldsymbol{\tau}_c$

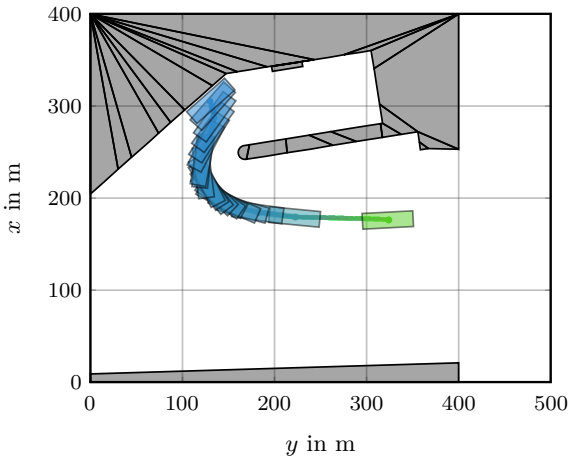
Abbildung 7.8.: Zeitlicher Verlauf der Zustands- und Stellgrößen der resultierenden Trajektorie.

6.2.2 beschriebenen Gründe zurückzuführen ist. Dabei sei angemerkt, dass die Stellrate der Antriebsorgane aufgrund des Eingangsfilters mit einer Zeitkonstante von 4 Sekunden geringer ist, als es während der manuellen Schiffsführung der Fall ist. Im Verlauf des Versuchs wurden die Änderungen der Stellgrößen vom Kapitän als ruhig empfunden, was durch die komprimierte Darstellung der Zeitachse innerhalb der Abbildungen nicht zu erkennen ist. Zudem gibt es aufgrund des Eingangsfilters keine sprunghaften Änderungen der Stellgröße.

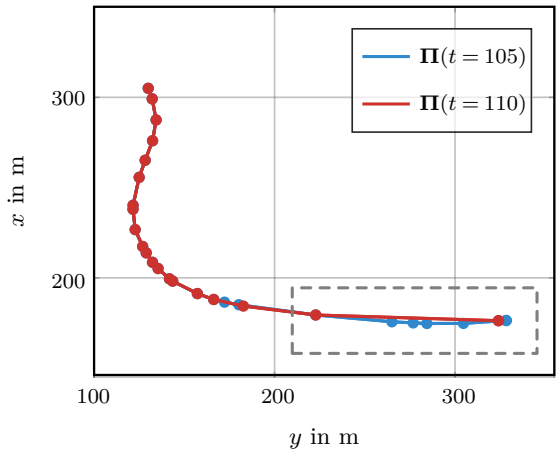
Jedoch zeigt sich ab etwa 100 Sekunden eine signifikante Änderung der Stellgröße. Ab diesem Zeitpunkt ist anhand der Längsgeschwindigkeit in der Abbildung 7.8b ein kurzzeitiger Abbremsvorgang zu erkennen, bevor das Fahrzeug erneut auf etwa 1 m/s beschleunigt wird. Der Grund dafür ist eine signifikante Änderung eines neu berechneten Lösungspfades in der Umgebung der Startpose. Dieser Fall wird mithilfe der Abbildung 7.9 verdeutlicht, in welcher der Lösungspfad zu zwei aufeinanderfolgenden Zeitpunkten $t = 105$ und $t = 110$ dargestellt ist. In der Teilabbildung 7.9c sind die Positionen der zwei Lösungspfade durch Punkte dargestellt, die über Linien miteinander verbunden sind. Im Bereich der Startpose, welcher durch eine grau gestrichelte Box visualisiert ist, sind Positionsdifferenzen der zwei Lösungspfade erkennbar. Der maximale Abstand beträgt etwa 2.7 m und macht sich als ein Querversatz bemerkbar. Auf diesen Posenfehler reagiert der Posenregler der *ExplorePath*-Prozedur mit einer Querkraft nach Steuerbord und einer gleichzeitigen Reduktion der Längskraft. Die Abbildung 7.9d zeigt eine schematische Darstellung dieses Zusammenhangs und illustriert die sprunghafte Änderung der Sollpose $\boldsymbol{\eta}_c$ aufgrund eines sich ändernden Pfades. Hierbei repräsentieren die blau und rot gestrichelten Linien die Lösungspfade $\boldsymbol{\Pi}(t = 105)$ bzw. $\boldsymbol{\Pi}(t = 110)$. Die Sollpose $\boldsymbol{\eta}_c(t = 105)$ ergibt sich entsprechend des *Guidance*-Gesetzes, welches in Abschnitt 5.4.3.4 beschrieben wurde, durch den Schnittpunkt von $\boldsymbol{\Pi}(t = 105)$ mit einem Ellipsoid. Mit dem nachfolgenden Lösungspfad $\boldsymbol{\Pi}(t = 110)$ wird eine zugehörige Sollpose $\boldsymbol{\eta}_c(t = 110)$ berechnet, die eine sprunghafte Änderung zu $\boldsymbol{\eta}_c(t = 105)$ darstellt. In körperfesten Koordinaten entsteht ein Längs- und Querversatz. Bei Fahrt mit einer Längsgeschwindigkeit von $u \approx 1\,\mathrm{m/s}$ wird als Folge die Längskraft reduziert, da der longitudinale



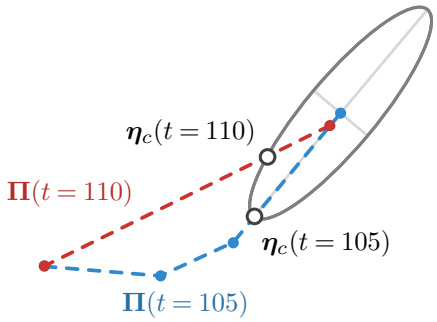
(a) $\mathbf{\Pi}(t = 105)$



(b) $\mathbf{\Pi}(t = 110)$



(c) Positionsverlauf der zwei aufeinanderfolgenden Lösungspfade $\mathbf{\Pi}(t = 105)$ und $\mathbf{\Pi}(t = 110)$ in der xy-Ebene.



(d) Schematische Darstellung der sprunghaften Änderung der Sollpose innerhalb der *ExplorePath*-Prozedur.

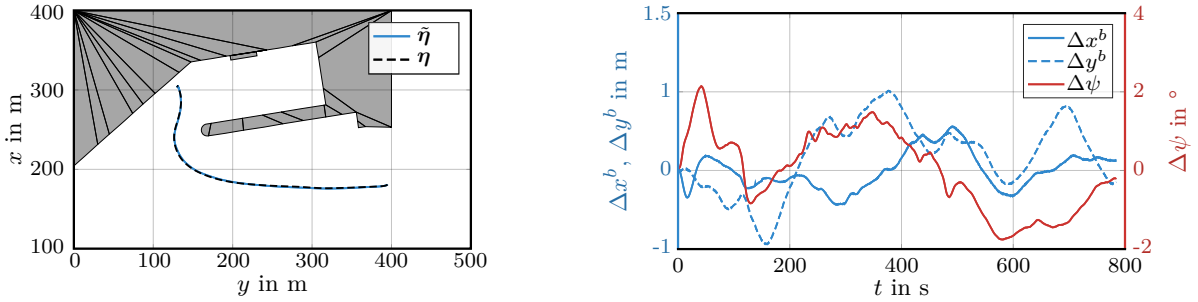
Abbildung 7.9.: Änderung von zwei aufeinanderfolgenden Lösungspfaden.

Positionsabstand sprunghaft kleiner wird.

Der beschriebene Effekt macht sich bei jeder Iteration des Online-Algorithmus bemerkbar, sobald ein neuer Lösungspfad mit einem geringeren Kostenwert zur Verfügung steht. Die rippelförmigen Änderungen der Stellgröße im Bereich von 300 bis 400 Sekunden aus der Abbildung 7.8d sind ebenfalls auf diesen Effekt zurückzuführen. Um dieses Problem zu lösen, ist es denkbar, dass die Planung eines neuen Pfades erst ab dem Schnittpunkt $\boldsymbol{\eta}_c$ des vorangegangenen Pfades mit dem Ellipsoid des *Guidance*-Gesetzes beginnt, um sprunghafte Änderungen der Sollpose zu vermeiden. Diese Strategie wird in der vorliegenden Arbeit nicht untersucht.

7.5.2. Analyse der ausgeführten Trajektorie

Wie in Abschnitt 7.2 erwähnt wurde, werden die geplanten Trajektorien von der sequenziellen Bewegungsplanung an den Trajektorienregler übergeben. Dieser verwendet die Stellgröße $\boldsymbol{\tau}_c$ als Vorsteuerung. Über eine Zustandsrückführung wird zudem der Regelfehler zwischen der geplanten Pose $\boldsymbol{\eta}$ und dem Istwert $\tilde{\boldsymbol{\eta}}$ minimiert. Der Entwurf des Reglers basiert auf demselben Bewegungsmodell, welches innerhalb der Bewegungsplanung verwendet wird. Im Folgenden werden die aktuellen Zustandsgrößen, welche vom *Navigation-System* berechnet werden, mit einer



(a) Positionsverlauf der geplanten (schwarz) und ausgeführten (blau) Trajektorie.

(b) Zeitlicher Verlauf der Posendifferenz in körperfesten Koordinaten.

Abbildung 7.10.: Posendifferenz zwischen der geplanten und ausgeführten Trajektorie.

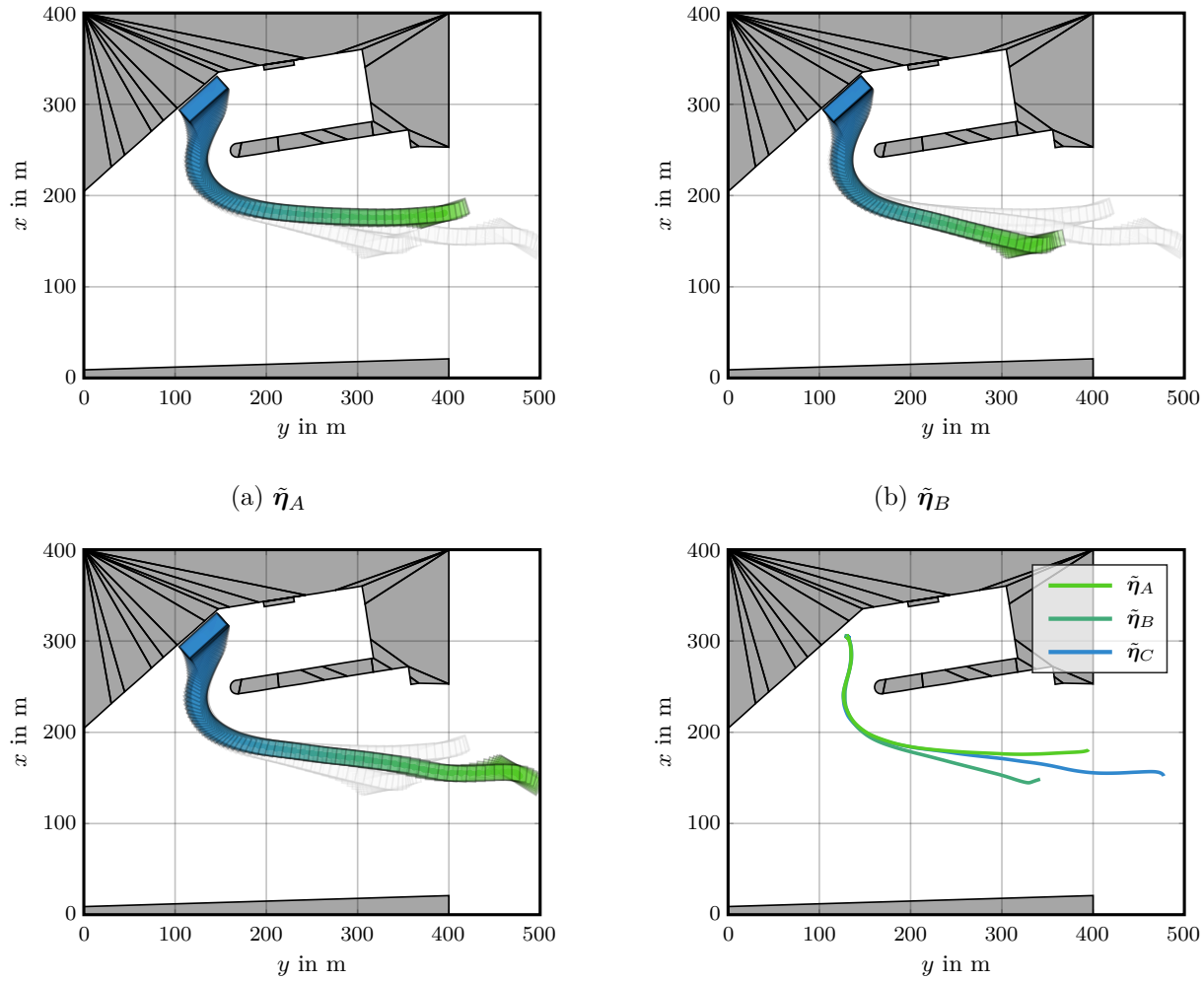
Tilde gekennzeichnet. Die Abbildung 7.10 zeigt den Positionsverlauf sowie die Posendifferenz

$$\begin{pmatrix} \Delta x^b \\ \Delta y^b \\ \Delta\psi \end{pmatrix} = \boldsymbol{T}_b^{n,\mathrm{T}}(\tilde{\psi}) \begin{pmatrix} \tilde{x} - x \\ \tilde{y} - y \\ \sigma(\tilde{\psi} - \psi) \end{pmatrix} \tag{7.3}$$

in körperfesten Koordinaten zwischen der geplanten und der ausgeführten Trajektorie. In der Abbildung 7.10a ist die geplante Trajektorie durch eine schwarz gestrichelte Linie dargestellt. Die blau gekennzeichnete Linie entspricht der ausgeführten Trajektorie und zeigt keine signifikanten Positionsabweichungen zur geplanten Trajektorie. Der zeitliche Verlauf der körperfesten Posendifferenz dieser zwei Trajektorien ist in der Teilgrafik 7.10b abgebildet. Es ist zu erkennen, dass die Differenz zum Zeitpunkt $t = 0$ bei null liegt, da die Bewegungsplanung im aktuellen Bewegungszustand des Fahrzeuges startet. Während des Versuchs variieren die Positionsfehler im Bereich von $\pm 1\,\mathrm{m}$ und der Winkelfehler bleibt betragsmäßig unter etwa 2°. Der Grund für die Fehler sind vor allem externe Eingangsstörungen, die auf den Prozess wirken. Zum Zeitpunkt der Versuchsdurchführung kam der Wind mit 4 bis 6 m/s aus nordwestlicher Richtung. Eine Strömung war im Hafenbecken nicht vorhanden. Die DENEB hat achtern zahlreiche Aufbauten wie Schornstein, Krane und Beiboote, welche dem Wind eine Angriffsfläche bieten. Zu Beginn des Versuchs drückt der Wind beispielsweise das Heck des Fahrzeuges nach Backbord, sodass ein positiver Winkelfehler und zugleich ein negativer lateraler Positionsfehler entstehen. Während des Traversierens ab etwa 600 Sekunden greift der Wind an der Backbordseite des Schiffes an und es entsteht ein positiver, lateraler Positionsfehler, welcher vom Trajektorienregler minimiert wird.

7.5.3. Reproduzierbarkeit der sequenziellen Bewegungsplanung

Um die Reproduzierbarkeit der sequenziellen Bewegungsplanung zu untersuchen, wurde das Anlegeszenario des Feldexperimentes insgesamt dreimal durchgeführt. Jeder Versuch war hierbei durch unterschiedliche Anfangszustände charakterisiert, wohingegen die Hindernisse sowie die Zielpose unverändert bleiben. Des Weiteren bleiben alle Tuningparameter konstant. Die Versuche A und B wurden an einem Tag mit ähnlichen Umweltbedingungen durchgeführt. Der Wind wehte mit 4 bis 6 m/s aus nordwestlicher Richtung. Der Versuch C wurde hingegen an einem anderen Tag durchgeführt, an dem der Wind mit 6 bis 8 m/s aus nördlicher Richtung kam. In der Abbildung 7.11 sind die Posenverläufe der ausgeführten Trajektorien gezeigt. Die Teilabbildung 7.11a zeigt den Posenverlauf $\tilde{\boldsymbol{\eta}}_A$ des ersten Versuchs, welcher identisch zu dem aus Abschnitt 7.5.2 ist. Zudem sind die Posenverläufe der anderen Versuche hellgrau gekennzeichnet. Das Ergebnis des zweiten Versuchs ist in der Teilabbildung 7.11b hervorgehoben. Der Heading-Winkel ist vergleichbar zu dem des ersten Versuchs, während die Startposition verschieden ist. Es ist eine Drehung des Schiffes zu Beginn des Versuchs, gefolgt von einer geradlinigen Fahrt zu erkennen.



(a) $\tilde{\eta}_A$

(b) $\tilde{\eta}_B$

(c) $\tilde{\eta}_C$

(d) Positionsverläufe der drei Trajektorien.

Abbildung 7.11.: Posenverlauf ausgeführter Trajektorien bei unterschiedlichen Anfangszuständen.

Der weitere Posenverlauf ab der Position $x = 200$ ist vergleichbar zu dem des ersten Versuchs. In der Teilabbildung 7.11c ist die ausgeführte Trajektorie des dritten Versuchs zu sehen, welcher durch einen anderen initialen Heading-Winkel gegenüber den Versuchen A und B charakterisiert ist. Nach der Drehung über Backbord fährt das Schiff in Vorausrichtung zur virtuellen Anlegepose, wobei das Manövrieren in der Nähe der Hafenmauer sowie das finale Traversieren einen vergleichbaren Posenverlauf in Bezug auf die Versuche A und B aufweist. Die Teilabbildung 7.11d gibt die Positionsverläufe der drei durchgeführten Versuche wieder. In jedem Versuch wurden erfolgreich kollisionsfreie Trajektorien generiert und durch ein bestehendes *Control-System* ausgeführt. Es ist zu erkennen, dass die gemessenen Positionswerte ab $x > 200$ während des Manövrierens um die Hafenmauer sowie dem Anlegen trotz unterschiedlicher Anfangsbedingungen und wechselnden Gegebenheiten von Umwelteinflüssen einen nahezu identischen Verlauf zeigen. Die Positionsdifferenz zwischen den Versuchen liegt in diesem Bereich bei unter 2.5 m und die Winkeldifferenz ist kleiner als 3°.

8. Zusammenfassung und Ausblick

Diese Arbeit stellt ein neuartiges Verfahren zur Bewegungsplanung traversierfähiger, autonomer Wasserfahrzeuge vor, bei dem das Manövrieren in der Nähe unbewegter Hindernisse von zentraler Bedeutung ist. Für die Wahl einer geeigneten Methodik werden Anforderungen an die Bewegungsplanung formuliert und entsprechend ihrer Relevanz priorisiert. Die exakte geometrische Darstellung von Hindernissen und Fahrzeugform sowie die Berücksichtigung des dynamischen Bewegungsverhaltens sind dabei maßgebend für die Generierung kollisionsfreier Trajektorien. Eine umfassende Analyse zum Stand der Technik hat ergeben, dass bestehende Verfahren die in der vorliegenden Arbeit definierten Anforderungen nur partiell erfüllen, sodass eine Modifikation existierender Verfahren notwendig ist.

Der vorgestellte Lösungsansatz zerlegt das Planungsproblem in zwei Teilprobleme: Pfadplanung und Bewegungsplanung. Sie basieren auf dem RRT*-Algorithmus und werden sequenziell gelöst, indem zunächst ein optimaler Pfad berechnet und anschließend in dessen Umgebung eine kollisionsfreie Trajektorie geplant wird, welche auf einem dynamischen Bewegungsmodell des Fahrzeuges beruht. Durch den sampling-basierten Planungsalgorithmus wird ein definierter Suchraum effizient erkundet und Randbedingungen wie das Vermeiden von Kollisionen lassen sich mithilfe von *black-box*-Funktion realisieren und müssen nicht explizit modelliert werden. Sofern eine Lösung existiert, konvergiert sie mit zunehmender Anzahl an Iterationen gegen eine optimale Lösung. Das Optimum entspricht dem Minimum einer Kostenfunktion, in der die Länge des Pfades gegenüber der Längsbewegung entlang dieses Pfades und dem Abstand zu Hindernissen gewichtet wird. Für Letzteres wird ein Skalarfeld berechnet, welches für den effizienten Einsatz innerhalb zahlreicher Iterationen in einer Lookup-Tabelle hinterlegt wird. Um Quer- und Rückwärtsbewegungen zu bestrafen, werden zwei Kostenterme entwickelt, die je nach Fahrzeugtyp parametrisiert werden können. Mithilfe der vorgestellten *WarmStart*-Prozedur erfolgt die Initialisierung der Pfadplanung auf Basis eines zuvor berechneten Pfades, wodurch eine signifikante Reduktion der Konvergenzzeit erreicht wird.

Die Methodik der Bewegungsplanung baut auf der der Pfadplanung auf, wobei nur in der Umgebung eines gefundenen Lösungspfades gesampelt wird. Für jeden Zweig des Suchbaums werden Trajektorien mithilfe der numerischen Simulation eines geschlossenen Regelkreises generiert und auf Kollisionsfreiheit der geometrischen Fahrzeugform mit den Hindernissen getestet. Dadurch wird das dynamische Bewegungsverhalten in eine *black-box*-Funktion des RRT*-Algorithmus eingebettet. Bei der Generierung einer Trajektorie wird das Verfahren der exakten Linearisierung verwendet, um nichtlineare Terme im Geschwindigkeitsmodell zu kompensieren. Mithilfe des entwickelten Online-Algorithmus werden in Echtzeit fortlaufend Trajektorien berechnet und Änderungen der Umgebung zur Laufzeit mit einbezogen. Dabei wird die Zeit, in der die aktuell gültige Trajektorie ausgeführt wird, für die Planung einer zukünftigen Trajektorie verwendet, welche derart an die bestehende Trajektorie angefügt wird, sodass ein stetiger Verlauf der Zustandsgrößen entsteht und sprunghafte Änderungen der Stellgröße vermieden werden.

Das vorgestellte Verfahren ist im Rahmen von Simulationen analysiert und anhand von Testszenarien validiert worden. Die Ergebnisse der simulativen Verfahrensanalyse zeigen, dass die vorgeschlagene Methodik das Problem der Bewegungsplanung mit den gegebenen Anforderungen löst. Zudem wird auf das Tuning der Kostenfunktion eingegangen und der Einfluss ausgewählter Parameter auf die Performance wird analysiert. Die experimentellen Ergebnisse bestätigen die Applizierbarkeit des Verfahrens am Beispiel eines Versuchsträgers. Die Generierung von Trajektorien erfolgt in Echtzeit und erlaubt unter Verwendung eines Trajektorienreglers präzises Manövrieren mit einem Positionsfehler im Dezimeterbereich.

Das in dieser Arbeit vorgestellte Verfahren zur Bewegungsplanung autonomer Wasserfahrzeuge bietet eine Grundlage für weiterführende Entwicklungen in dieser Thematik. Eine offene Fragestellung ist, ob innerhalb des Online-Algorithmus eine alternative Startpose für die sequenzielle Bewegungsplanung der nächsten Iteration verwendet werden kann, um einen glatteren Stellgrößenverlauf zu erzielen, wie in 7.5.1 diskutiert wurde. Da die Wahl von Tuningparametern einen signifikanten Einfluss auf die Performance und den resultierenden Verlauf der Trajektorie hat, bleibt zudem zu untersuchen, inwiefern sich für unterschiedliche Fahrzeuggrößen möglicherweise heuristische Einstellregeln ableiten lassen. Ein weiterer Punkt ist die Berücksichtigung von externen Störgrößen bereits während der Planung, sofern diese bekannt sind. Denkbar ist die Verwendung einer Störgrößenaufschaltung innerhalb der *ExplorePath*-Prozedur. Für ortsabhängige Störungen wie beispielsweise der Strömungsänderung an einer Flussmündung ließen sich zusätzliche Terme der Kostenfunktion hinzufügen. Analog zur Gewichtung des Abstands zu Hindernissen könnte für diese Zwecke ebenfalls eine Lookup-Tabelle verwendet werden. Das vorgestellte Verfahren berücksichtigt ausschließlich unbewegte Hindernisse. Um eine Bewegungsplanung bei sich bewegenden Hindernissen umzusetzen, müssen zunächst die Anforderungen und gegebenenfalls die Methodik als Grundlage der Planung angepasst werden.

Das Thema der Bewegungsplanung autonomer Fahrzeuge ist Bestandteil aktueller, internationaler Forschung und es sind kontinuierliche Entwicklungen auf diesem Gebiet zu beobachten. Die Problemstellung der Bewegungsplanung speziell für Wasserfahrzeuge geht mit Herausforderungen einher, die sich vom Bereich der Luft- und Raumfahrt und von Landfahrzeugen unterscheiden. Diese Arbeit adressiert offene Fragestellungen und leistet ein Beitrag auf diesem Forschungsgebiet. Zugleich bietet sie Anknüpfungspunkte für zukünftige Forschungen und Entwicklungen.

Anhang

A. Anhang zur sequenziellen Bewegungsplanung

A.1. Vergleich der verwendeten Modellstruktur mit gängigen Modellen aus der Literatur

Das Geschwindigkeitsmodell wird in dieser Arbeit mit der nichtlinearen Differenzialgleichung

$$\dot{\boldsymbol{\nu}} = \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \tag{A.1}$$

beschrieben, wobei $\boldsymbol{\nu} = (u, v, r)^{\mathrm{T}}$ den Geschwindigkeitsvektor darstellt. In der Literatur wird die Bewegungsgleichung in der Form

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{A.2}$$

angegeben, wobei $\boldsymbol{M}$ als Trägheitsmatrix, $\boldsymbol{C}(\boldsymbol{\nu})$ als Coriolismatrix und $\boldsymbol{D}(\boldsymbol{\nu})$ als Dämpfungsmatrix bezeichnet wird [102]. Die Trägheitsmatrix sowie die Coriolismatrix enthalten neben der eigentlichen Masse des Fahrzeuges hydrodynamische Massen. Da das Fahrzeug und das Fluid, indem es sich bewegt, nicht denselben Raum zur gleichen Zeit belegen können, muss bei der Bewegung des Fahrzeuges ein Teil des umgebenen Fluids bewegt werden. Dieser Effekt macht sich als zusätzliche Trägheit bemerkbar und wird durch eine zusätzliche oder hydrodynamische Masse (engl. *added mass*) abgebildet.

Die beiden nichtlinearen Differenzialgleichungen (A.1) und (A.2) sind mathematisch identisch, sofern in $\boldsymbol{f}(\boldsymbol{\nu})$ die gleichen geschwindigkeitsabhängigen Terme wie in (A.2) enthalten sind. Die Gleichung (A.2) lässt sich zu

$$\dot{\boldsymbol{\nu}} = \underbrace{-\boldsymbol{M}^{-1}\left(\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right)}_{\boldsymbol{f}(\boldsymbol{\nu})} + \underbrace{\boldsymbol{M}^{-1}}_{\boldsymbol{B}}\boldsymbol{\tau} \tag{A.3}$$

umformen. Prinzipiell können die Matrizen $\boldsymbol{M}$, $\boldsymbol{C}(\boldsymbol{\nu})$ und $\boldsymbol{D}(\boldsymbol{\nu})$ voll besetzt sein. Für reduzierte Bewegungsmodelle zum Manövrieren wird in [102] jedoch die Längsbewegung von der Quer- und Drehbewegung entkoppelt. Damit ergeben sich für die Trägheits- und Coriolismatrix

$$\boldsymbol{M} = \begin{pmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & m x_g - Y_{\dot{r}} \\ 0 & m x_g - Y_{\dot{r}} & I_z - N_{\dot{r}} \end{pmatrix}, \tag{A.4}$$

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{pmatrix} 0 & 0 & -m(x_g r + v) + Y_{\dot{v}} v + Y_{\dot{r}} r \\ 0 & 0 & m u - X_{\dot{u}} \\ m(x_g r + v) - Y_{\dot{v}} v - Y_{\dot{r}} r & -m u + X_{\dot{u}} & 0 \end{pmatrix}. \tag{A.5}$$

Hierbei sind m und I_z die Masse bzw. das Trägheitsmoment, x_g ist die Position des Schwerpunktes entlang der x^b-Achse und $X_{\dot{u}}$, $Y_{\dot{v}}$, $Y_{\dot{r}}$, $N_{\dot{r}}$ sind die zusätzlichen Massen. Für die Dämpfung werden oft lineare und quadratische Terme verwendet. Die Dämpfungsmatrix ist dann beispielsweise mit

$$\boldsymbol{D}(\boldsymbol{\nu}) = \begin{pmatrix} -X_u - X_{|u|u}|u| & 0 & 0 \\ 0 & -Y_v - Y_{|v|v}|v| - Y_{|r|v}|r| & -Y_r - Y_{|v|r}|v| - Y_{|r|r}|r| \\ 0 & -N_v - N_{|v|v}|v| - N_{|r|v}|r| & -N_r - N_{|v|r}|v| - N_{|r|r}|r| \end{pmatrix} \tag{A.6}$$

gegeben, wobei X_u, Y_v, Y_r, N_v, N_r die zugehörigen Parameter für lineare Terme und $X_{|u|u}$, $Y_{|v|v}$, $Y_{|v|r}$, $Y_{|r|v}$, $Y_{|r|r}$, $N_{|v|v}$, $N_{|v|r}$, $N_{|r|v}$, $N_{|r|r}$ die entsprechenden Parameter für quadratische Terme kennzeichnen. Werden nun die Matrizen (A.4), (A.5) und (A.6) in (A.3) eingesetzt, ergeben sich die folgenden Proportionalitäten.

$$\dot{\boldsymbol{\nu}} \sim \{u,\ v,\ r,\ vr,\ ur,\ uv,\ r^2,\ |u|u,\ |v|v,\ |r|r,\ |v|r,\ |r|v\}$$

Der Ausdruck für $-\boldsymbol{M}^{-1}\left(\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu}+\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right)$ in (A.3) kann in einer verallgemeinerten Form als

$$\boldsymbol{f}(\boldsymbol{\nu}) = \boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu}) \tag{A.7}$$

dargestellt werden, wobei $\boldsymbol{F} \in \mathbb{R}^{3\times 12}$ eine Koeffizienten-Matrix ist, welche sich aus Modellparametern zusammensetzt, und

$$\boldsymbol{n}(\boldsymbol{\nu}) = (u,\ v,\ r,\ vr,\ ur,\ uv,\ r^2,\ |u|u,\ |v|v,\ |r|r,\ |v|r,\ |r|v)^{\mathrm{T}}$$

unterschiedliche Terme aus verschiedenen Kombinationen der Geschwindigkeitskomponenten enthält. Prinzipiell lassen sich auf diese Weise auch weitere nichtlineare Terme berücksichtigen. In [102] werden Bewegungsmodelle vorgestellt, die noch höhere Ordnungen verwenden. Dazu kann $\boldsymbol{n}(\boldsymbol{\nu})$ mit den folgenden Termen erweitert werden.

$$\boldsymbol{n}(\boldsymbol{\nu}) = (\ldots,\ u^2,\ v^2,\ u^3,\ v^3,\ r^3,\ uvr,\ r^2v,\ v^2r)^{\mathrm{T}}$$

In dieser Arbeit wird sich auf den allgemeinen Fall (A.7) beschränkt. Der Entwickler hat somit die Freiheit, sich auf die Terme beschränken zu können, mit denen der Prozess ausreichend gut modelliert werden kann.

A.2. Berechnung der Kostenterme $c_{v,1}$ und $c_{v,2}$

Die Kostenfunktion für das Bevorzugen der Längsgeschwindigkeit entlang einer Geraden von $\boldsymbol{q}_0 = (x_0, y_0, \psi_0)^{\mathrm{T}}$ nach $\boldsymbol{q}_1 = (x_1, y_1, \psi_1)^{\mathrm{T}}$ wurde in Abschnitt 5.3.2 vorgestellt und ist definiert als das Integral über eine Gewichtungsfunktion der Winkeldifferenz zwischen dem Winkel der Geraden und dem Heading-Winkel.

$$c_v(\boldsymbol{q}_0, \boldsymbol{q}_1) = \begin{cases} 0 & \text{falls } L = 0 \\ \underbrace{\int_{s=0}^{L} g_1\left(\Delta\psi(s)\right) ds}_{c_{v,1}} + \underbrace{\int_{s=0}^{L} g_2\left(\Delta\psi(s)\right) ds}_{c_{v,2}} & \text{sonst} \end{cases} \tag{A.8}$$

Hierbei sind

$$g_1(\Delta\psi) = w_v \sin^2\left(\Delta\psi\right) \tag{A.9}$$

$$g_2(\Delta\psi) = w_\alpha \frac{(w_\beta \Delta\psi)^2}{1+(w_\beta \Delta\psi)^2} \tag{A.10}$$

die Gewichtungsfunktionen in Abhängigkeit der Winkeldifferenz

$$\Delta\psi(s) = \sigma(\psi(s) - \psi_\varepsilon) \tag{A.11}$$

mit

$$\psi(s) = \psi_0 + \frac{s}{L}\sigma(\psi_1 - \psi_0) \quad \text{mit} \quad s \in [0, L] \tag{A.12}$$

$$\psi_\varepsilon = \operatorname{atan2}\left(y_1 - y_0,\ x_1 - x_0\right)\ . \tag{A.13}$$

Die Länge der Geraden wird mit

$$L = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{A.14}$$

berechnet. Die bestimmten Integrale $c_{v,1}$ und $c_{v,2}$ aus (A.8) lassen sich analytisch lösen, wobei zur korrekten Behandlung von Winkeldifferenzen mithilfe der $\sigma()$-Funktion nach (5.9) zwischen mehreren Fällen unterschieden werden muss. Die Stammfunktion des Integrals $c_{v,1}$ ist

$$F_1 = w_v \left(\frac{s}{2} + \frac{L \sin\left(2\psi_A - \frac{2s(\psi_A - \psi_B)}{L}\right)}{4(\psi_A - \psi_B)} + C \right) , \tag{A.15}$$

wobei C eine Konstante ist. Hierbei sind

$$\psi_A = \sigma(\psi_0 - \psi_\varepsilon)$$
$$\psi_B = \sigma(\psi_1 - \psi_\varepsilon)$$

die relativen Heading-Winkel von $\boldsymbol{q}_0$ bzw. $\boldsymbol{q}_1$ bezüglich des Geradenwinkels ψ_ε. Wenn $\psi_A = \psi_B$ ist, dann hängt $\Delta\psi$ nicht mehr von der Integrationsvariable s ab und das erste Integral aus (A.8) vereinfacht sich zu

$$\int_{s=0}^{L} g_1(\Delta\psi(s))\, ds = g_1(\Delta\psi) \int_{s=0}^{L} ds = w_v L \sin^2(\Delta\psi) , \tag{A.16}$$

wobei für diesen Fall $\Delta\psi = \psi_A = \psi_B$ gilt. Durch die Auswertung der Stammfunktion (A.15) im Intervall $[0, L]$ sowie dem Ergebnis aus (A.16) resultiert

$$c_{v,1}(\boldsymbol{q}_0, \boldsymbol{q}_1) = \begin{cases} w_v L \sin^2(\psi_A) & \text{falls } \psi_A = \psi_B \\ w_v L \dfrac{2(\psi_B - \psi_A) + \sin(2\psi_A) - \sin(2\psi_B)}{4(\psi_B - \psi_A)} & \text{sonst.} \end{cases} \tag{A.17}$$

Die Stammfunktion des zweiten Integrals aus (A.8) ist

$$F_2 = w_\alpha \left(s + \frac{L \arctan\left(\frac{w_\beta(L\psi_A - s\psi_A + s\psi_B)}{L}\right)}{w_\beta(\psi_A - \psi_B)} + C \right) . \tag{A.18}$$

Für den Fall $\psi_A = \psi_B$ vereinfacht sich das Integral zu

$$\int_{s=0}^{L} g_2(\Delta\psi(s))\, ds = g_2(\Delta\psi) \int_{s=0}^{L} ds = w_\alpha L \frac{(w_\beta \Delta\psi)^2}{1 + (w_\beta \Delta\psi)^2} . \tag{A.19}$$

Im Gegensatz zur Gewichtungsfunktion g_1 ist g_2 nicht periodisch bezüglich der Winkeldifferenz $\Delta\psi$. Infolgedessen muss die Stammfunktion (A.18) für zwei Teilintervalle berechnet werden, falls $\Delta\psi(s)$ die Grenze $\pm\pi$ überschreitet. Der Ausdruck für $c_{v,2}$ wird mit

$$c_{v,2}(\boldsymbol{q}_0, \boldsymbol{q}_1) = \begin{cases} w_\alpha L \dfrac{(w_\beta \psi_A)^2}{1 + (w_\beta \psi_A)^2} & \text{falls } \delta = 0 \\ F_2|_{s=0}^{s=L^+} + F_2|_{s=L^+}^{s=L} & \text{falls } \psi_A + \delta > \pi \\ F_2|_{s=0}^{s=L^-} + F_2|_{s=L^-}^{s=L} & \text{falls } \psi_A + \delta < -\pi \\ w_\alpha L \left(1 + \dfrac{\arctan(w_\beta \psi_A) - \arctan(w_\beta \psi_B)}{w_\beta(\psi_B - \psi_A)}\right) & \text{sonst} \end{cases} \tag{A.20}$$

berechnet, wobei

$$\delta = \sigma(\psi_B - \psi_A) = \sigma(\psi_1 - \psi_0) \tag{A.21}$$

die Winkeldifferenz zwischen den Posen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$ ist und

$$L^+ = L\frac{\pi - \psi_A}{\delta} \tag{A.22}$$

$$L^- = L\frac{-\pi - \psi_A}{\delta} \tag{A.23}$$

die Integrationsgrenzen kennzeichnen, an denen die Stelle $\Delta\psi(s) = \pm\pi$ überschritten wird. Das Auswerten der Stammfunktion F_2 an den angegebenen Grenzen liefert

$$F_2|_{s=0}^{s=L^+} = w_\alpha L^+ \left(1 + \frac{\arctan(w_\beta\psi_A) - \arctan(w_\beta\pi)}{w_\beta(\pi - \psi_A)}\right) \tag{A.24}$$

$$F_2|_{s=L^+}^{s=L} = w_\alpha \left(L - L^+\right) \left(1 + \frac{\arctan(-w_\beta\pi) - \arctan(w_\beta\psi_B)}{w_\beta(\psi_B + \pi)}\right) \tag{A.25}$$

$$F_2|_{s=0}^{s=L^-} = w_\alpha L^- \left(1 + \frac{\arctan(w_\beta\psi_A) - \arctan(-w_\beta\pi)}{w_\beta(-\pi - \psi_A)}\right) \tag{A.26}$$

$$F_2|_{s=L^-}^{s=L} = w_\alpha (L - L^-) \left(1 + \frac{\arctan(w_\beta\pi) - \arctan(w_\beta\psi_B)}{w_\beta(\psi_B - \pi)}\right) . \tag{A.27}$$

A.3. Wahl der geometrischen Fahrzeugform für einen sicheren Kollisionstest

Für den Kollisionstest des Fahrzeugpolygons entlang einer Geraden oder Kurve innerhalb der Pfad- bzw. Bewegungsplanung muss die Vereinigung aller Fahrzeugpolygone $\mathcal{V}(\boldsymbol{q})$ für alle Konfigurationen $\boldsymbol{q}$ entlang dieser Geraden bzw. Kurve betrachtet werden. Die resultierende geometrische Form

$$\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1) = \bigcup_{\boldsymbol{q}\in\varepsilon} \mathcal{V}(\boldsymbol{q}) \tag{A.28}$$

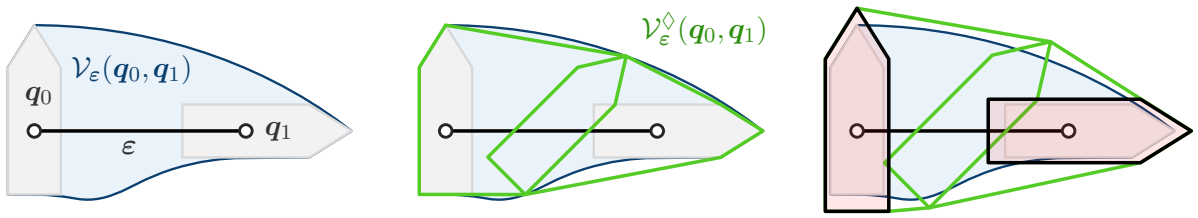
besteht demzufolge aus der Vereinigung unendlich vieler Polygone entlang einer beliebigen Kurve ε von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$. In der Abbildung A.1a ist diese geometrische Form anhand eines Beispiels als blaue Fläche dargestellt. In den vorgestellten Verfahren der Kollisionsprüfung wird $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch eine endliche Anzahl konvexer Polygone approximiert, die sich jeweils aus der konvexen Hülle von $\mathcal{V}$ an zwei Konfigurationen $\boldsymbol{q}_i \in \varepsilon$ und $\boldsymbol{q}_j \in \varepsilon$ ergeben, deren Positions- bzw. Winkeldifferenz unter den vorgegebenen Schwellwerten Δp_{max} und $\Delta\psi_{max}$ liegen. Die Approximation wird mit

$$\mathcal{V}_\varepsilon^\diamondsuit(\boldsymbol{q}_0, \boldsymbol{q}_1) = \bigcup_N \text{convhull}\left(\mathcal{V}(\boldsymbol{q}_i), \mathcal{V}(\boldsymbol{q}_j)\right) \tag{A.29}$$

bezeichnet, wobei N die Anzahl an Liniensegmenten kennzeichnet, in die ε zerlegt wird. Die Teilabbildung A.1b zeigt die Approximation von $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch zwei konvexe Polygone (grün) an einem Beispiel für $\Delta\psi_{max} = 45°$. Es ist zu erkennen, dass mit $\mathcal{V}_\varepsilon^\diamondsuit(\boldsymbol{q}_0, \boldsymbol{q}_1)$ in einigen Bereichen eine Kollision ausgeschlossen werden würde, obwohl sich $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ mit Hindernissen überschneiden könnte. Um diesem Problem zu begegnen, muss das Fahrzeugpolygon, welches für sämtliche Kollisionsüberprüfungen innerhalb der Planung verwendet wird, so gewählt werden, dass $\mathcal{V}_\varepsilon(\boldsymbol{q}_0, \boldsymbol{q}_1)$ stets eine Untermenge von $\mathcal{V}_\varepsilon^\diamondsuit(\boldsymbol{q}_0, \boldsymbol{q}_1)$ ist. Im Allgemeinen muss das Fahrzeugpolygon größer als die reale geometrische Fahrzeugform gewählt werden. Die Teilabbildung A.1c zeigt die Approximation $\mathcal{V}_\varepsilon^\diamondsuit(\boldsymbol{q}_0, \boldsymbol{q}_1)$ auf Basis eines Fahrzeugpolygons, welches um den Faktor 1.2 gegenüber der realen geometrischen Fahrzeugform vergrößert wurde.

A.3.1. Skalierung des Fahrzeugpolygons

Die Vergrößerung des Fahrzeugpolygons in Form einer Skalierung lässt sich für einen Spezialfall berechnen. Es wird angenommen, dass sich der Ursprung des *b-frame* innerhalb der realen



(a) Exakte geometrische Form $\mathcal{V}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ am Beispiel einer Geraden $\boldsymbol{\varepsilon}$ von $\boldsymbol{q}_0$ zu $\boldsymbol{q}_1$.

(b) Approximation von $\mathcal{V}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch zwei konvexe Polygone für $\Delta\psi_{max} = 45°$.

(c) Vergrößerung des Fahrzeugpolygons um den Faktor 1.2 für einen sicheren Kollisionstest.

Abbildung A.1.: Grafische Darstellung der Approximation von $\mathcal{V}_{\boldsymbol{\varepsilon}}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ durch zwei konvexe Polygone (grün) für $\Delta\psi_{max} = 45°$ und unterschiedlich große Fahrzeugpolygone.

geometrischen Fahrzeugform befindet und das diese Form einem einzigen konvexen Polygon entspricht. Für den Fall, dass das Fahrzeug auf der Stelle dreht, lässt sich ein Skalierungsfaktor für ein gegebenes $\Delta\psi_{max}$ berechnen, mit dem alle Vertices des konvexen Polygons multipliziert werden müssen, um ein größeres konvexes Polygon zu erhalten, welches einen sicheren Kollisionstest garantiert.

Der Vertex $\boldsymbol{p}$ des Fahrzeugpolygons mit dem größten Abstand zum Ursprung des *b-frame* liegt auf einem Kreis mit Radius R. Durch die Rotation des Fahrzeugpolygons mit $\Delta\psi_{max}$ ergibt sich der Vertex $\boldsymbol{p}'$. Die Vertices $\boldsymbol{p}$ und $\boldsymbol{p}'$ bilden eine Sekante, sodass die Fläche des resultierenden Dreiecks aus dem Koordinatenursprung sowie $\boldsymbol{p}$ und $\boldsymbol{p}'$ kleiner als die Fläche des zugehörigen Kreissegments ist. Um die gesamte Fläche des Kreissegmentes während des Kollisionstests abdecken zu können, wird jeder Vertex mit einem Faktor s skaliert, so dass die Strecke von $s \cdot \boldsymbol{p}$ zu $s \cdot \boldsymbol{p}'$ eine Tangente des Kreises bildet. Für das blau markierte rechtwinklige Dreieck aus Abbildung A.2 gilt der trigonometrische Zusammenhang

$$\cos\left(\frac{\Delta\psi_{max}}{2}\right) = \frac{R}{\|s \cdot \boldsymbol{p}\|_2} = \frac{R}{sR} \,. \tag{A.30}$$

Hieraus ergibt sich der Skalierungsfaktor

$$s = \frac{1}{\cos\left(\frac{\Delta\psi_{max}}{2}\right)} \,. \tag{A.31}$$

A.3.2. Vorschlag für die Wahl des Fahrzeugpolygons

Sei $\mathcal{V}_r$ das gegebene Fahrzeugpolygon des realen ASVs und $\mathcal{V}_{r,\varepsilon}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ die exakte geometrische Form von $\mathcal{V}_r$ entlang einer beliebigen Kurve $\boldsymbol{\varepsilon}$ zwischen zwei Konfigurationen $\boldsymbol{q}_0$ und $\boldsymbol{q}_1$, dann ist ein Fahrzeugpolygon $\mathcal{V}$ gesucht, sodass für die Approximation $\mathcal{V}_{\boldsymbol{\varepsilon}}^{\diamondsuit}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ entlang jeder beliebigen Kurve die Bedingung

$$\mathcal{V}_{r,\varepsilon}(\boldsymbol{q}_0, \boldsymbol{q}_1) \subseteq \mathcal{V}_{\boldsymbol{\varepsilon}}^{\diamondsuit}(\boldsymbol{q}_0, \boldsymbol{q}_1) \tag{A.32}$$

gilt. Da diese Bedingung auch für den trivialen Fall $\boldsymbol{q}_0 = \boldsymbol{q}_1 = \boldsymbol{0}$ erfüllt sein muss, folgt

$$\mathcal{V}_r \subseteq \mathcal{V} \,. \tag{A.33}$$

Das gesuchte Fahrzeugpolygon muss demnach eine Obermenge der realen geometrischen Fahrzeugform bilden. Prinzipiell lässt sich $\mathcal{V}$ beliebig wählen und im Anschluss mithilfe von Monte-Carlo-Simulationen überprüfen, ob die Bedingung (A.32) für das gewählte $\mathcal{V}$ erfüllt ist. Für eine numerische Untersuchung muss in diesem Zusammenhang $\mathcal{V}_{r,\varepsilon}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ ebenfalls durch eine

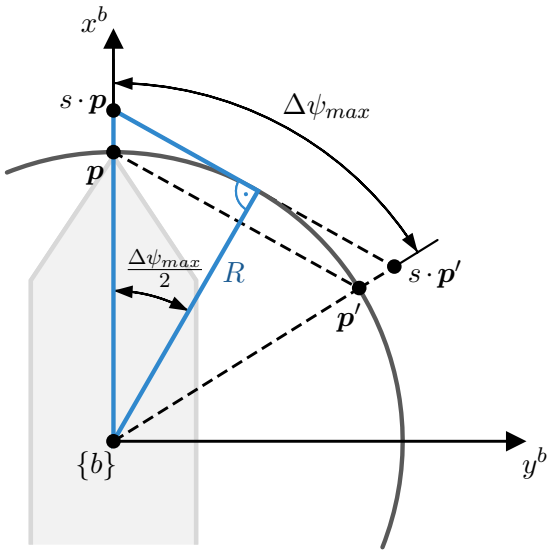


Abbildung A.2.: Schematische Darstellung zur Berechnung des Skalierungsfaktors s.

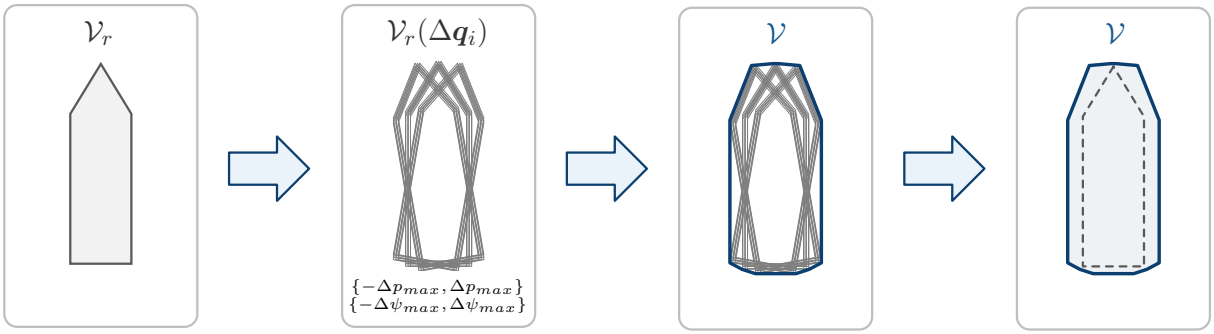


Abbildung A.3.: Schematische Darstellung der Berechnung eines vergrößerten Fahrzeugpolygons auf Basis der realen geometrischen Fahrzeugform.

endlichen Menge konvexer Polygone approximiert werden. Die zugehörigen Schwellwerte Δp_{max} und $\Delta\psi_{max}$ für die Berechnung von $\mathcal{V}_{r,\varepsilon}^{\diamondsuit}(\boldsymbol{q}_0, \boldsymbol{q}_1)$ können entsprechend klein gewählt werden.

Im Folgenden wird eine Vorgehensweise für die Wahl des Fahrzeugpolygons $\mathcal{V}$ auf der Grundlage von $\mathcal{V}_r$ sowie Δp_{max} und $\Delta\psi_{max}$ vorgeschlagen. Der maximale Positions- bzw. Winkelfehler zwischen einer beliebigen Kurve und den Liniensegmenten, welche diese Kurve approximieren, ist durch die Schwellwerte Δp_{max} und $\Delta\psi_{max}$ festgelegt. Diese Fehler werden der realen geometrischen Fahrzeugform hinzugefügt. Eine schematische Darstellung für die Berechnung eines vergrößerten Fahrzeugpolygons ist in Abbildung A.3 gezeigt. Die reale geometrische Fahrzeugform ist durch ein konvexes Polygon $\mathcal{V}_r$ gegeben. Im ersten Schritt wird $\mathcal{V}_r$ mit mehreren Konfigurationen $\Delta\boldsymbol{q}_i$ transformiert, die sich aus allen Kombinationen der Schwellwerte Δp_{max} und $\Delta\psi_{max}$ ergeben, sodass der gesamte Fehlerbereich abgedeckt wird. Da Winkelfehler im Wertebereich $[-\Delta\psi_{max}, \Delta\psi_{max}]$ durch eine endliche Anzahl von Polygonen approximiert werden, muss $\mathcal{V}_r$ mit einem Faktor nach (A.31) skaliert werden, wie in A.3.1 beschrieben wurde. Aus allen transformierten Polygonen wird eine konvexe Hülle berechnet, welche schließlich das vergrößerte Fahrzeugpolygon $\mathcal{V}$ ergibt. Je größer die Schwellwerte Δp_{max} und $\Delta\psi_{max}$ gewählt werden, desto größer wird das resultierende Fahrzeugpolygon $\mathcal{V}$.

A.4. Exakte Linearisierung anhand eines Beispielmodells

In diesem Anhang werden die zeitlichen Ableitungen des Ausgangsvektors des Geschwindigkeitsmodells anhand eines konkreten Beispielmodells berechnet. Das nichtlineare Modell ist mit

$$\begin{aligned}\dot{\boldsymbol{\nu}} &= \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \\ \dot{\boldsymbol{\tau}} &= \boldsymbol{A}_\tau \boldsymbol{\tau} + \boldsymbol{B}_\tau \boldsymbol{\tau}_c \\ \dot{\boldsymbol{\tau}}_c &= \boldsymbol{A}_c \boldsymbol{\tau} + \boldsymbol{B}_c \boldsymbol{u}_\tau\end{aligned} \tag{A.34}$$

gegeben. In diesem Beispiel wird

$$\boldsymbol{f}(\boldsymbol{\nu}) = \underbrace{\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1,10} \\ f_{21} & f_{22} & \cdots & f_{2,10} \\ f_{31} & f_{32} & \cdots & f_{3,10} \end{pmatrix}}_{\boldsymbol{F}} \underbrace{\begin{pmatrix} u & v & r & vr & ur & uv & r^2 & u^3 & v^3 & r^3 \end{pmatrix}^{\mathrm{T}}}_{\boldsymbol{n}(\boldsymbol{\nu})} \tag{A.35}$$

angenommen. In Komponentenschreibweise ist die Vektorfunktion demzufolge mit

$$\boldsymbol{f}(\boldsymbol{\nu}) = \begin{pmatrix} f_u \\ f_v \\ f_r \end{pmatrix} = \begin{pmatrix} f_{11}u + f_{12}v + f_{13}r + f_{14}vr + f_{15}ur + f_{16}uv + f_{17}r^2 + f_{18}u^3 + f_{19}v^3 + f_{1,10}r^3 \\ f_{21}u + f_{22}v + f_{23}r + f_{24}vr + f_{25}ur + f_{26}uv + f_{27}r^2 + f_{28}u^3 + f_{29}v^3 + f_{2,10}r^3 \\ f_{31}u + f_{32}v + f_{33}r + f_{34}vr + f_{35}ur + f_{36}uv + f_{37}r^2 + f_{38}u^3 + f_{39}v^3 + f_{3,10}r^3 \end{pmatrix} \tag{A.36}$$

gegeben. Die Eingangsmatrix

$$\boldsymbol{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \tag{A.37}$$

ist regulär und in diesem Beispiel voll besetzt. Entsprechend der Gleichungen (5.55)-(5.59) erfolgt die Berechnung der zeitlichen Ableitungen des Ausgangsvektors $\boldsymbol{y}_\nu = \boldsymbol{\nu}$ mit

$$\dot{\boldsymbol{y}}_\nu = \boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \tag{A.38}$$

$$\ddot{\boldsymbol{y}}_\nu = \frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}} \Big(\boldsymbol{f}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \Big) + \boldsymbol{B} \left(\boldsymbol{A}_\tau \boldsymbol{\tau} + \boldsymbol{B}_\tau \boldsymbol{\tau}_c \right) \tag{A.39}$$

$$\dddot{\boldsymbol{y}}_\nu = \mathcal{A}(\boldsymbol{x}_\nu) + \mathcal{B}\boldsymbol{u}_\tau \ . \tag{A.40}$$

Für das konkrete Beispielmodell (A.35) ergibt sich demzufolge

$$\dot{\boldsymbol{y}}_\nu = \boldsymbol{F} \cdot \boldsymbol{n}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \tag{A.41}$$

für die erste zeitliche Ableitung. In (A.39) wird die partielle Ableitung von (A.35) bezüglich $\boldsymbol{\nu}$ benötigt. Die Berechnung einer partiellen Ableitung einer Vektorfunktion nach einem Vektor entspricht der Jacobi-Matrix, welche mit

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}} = \begin{pmatrix} \frac{\partial f_u}{\partial u} & \frac{\partial f_u}{\partial v} & \frac{\partial f_u}{\partial r} \\ \frac{\partial f_v}{\partial u} & \frac{\partial f_v}{\partial v} & \frac{\partial f_v}{\partial r} \\ \frac{\partial f_r}{\partial u} & \frac{\partial f_r}{\partial v} & \frac{\partial f_r}{\partial r} \end{pmatrix} \tag{A.42}$$

berechnet wird. Daraus folgt für die zweite zeitliche Ableitung des Ausgangsvektors

$$\ddot{\boldsymbol{y}}_\nu = \boldsymbol{J} \left(\boldsymbol{F} \cdot \boldsymbol{n}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau} \right) + \boldsymbol{B} \left(\boldsymbol{A}_\tau \boldsymbol{\tau} + \boldsymbol{B}_\tau \boldsymbol{\tau}_c \right) \tag{A.43}$$

mit

$$\boldsymbol{J} = \begin{pmatrix} f_{11} + f_{15}r + f_{16}v + 3f_{18}u^2 & f_{12} + f_{14}r + f_{16}u + 3f_{19}v^2 & f_{13} + f_{14}v + f_{15}u + 2f_{17}r + 3f_{1,10}r^2 \\ f_{21} + f_{25}r + f_{26}v + 3f_{28}u^2 & f_{22} + f_{24}r + f_{26}u + 3f_{29}v^2 & f_{23} + f_{24}v + f_{25}u + 2f_{27}r + 3f_{2,10}r^2 \\ f_{31} + f_{35}r + f_{36}v + 3f_{38}u^2 & f_{32} + f_{34}r + f_{36}u + 3f_{39}v^2 & f_{33} + f_{34}v + f_{35}u + 2f_{37}r + 3f_{3,10}r^2 \end{pmatrix} . \tag{A.44}$$

Der Term $\mathcal{A}(\boldsymbol{x}_\nu)$ in (A.40) wird entsprechend der Gleichung (5.58) berechnet. Für das gegebene Beispielmodell (A.35) sowie dem bereits bekannten Ausdruck der Jacobi-Matrix aus (A.44) ergibt sich

$$\begin{aligned}\mathcal{A}(\boldsymbol{x}_\nu) = &\left[\frac{\partial}{\partial \boldsymbol{\nu}}\left(\boldsymbol{J}\left(\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau}\right)\right)\right](\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau})+\ldots\\ &\ldots(\boldsymbol{J}\boldsymbol{B}+\boldsymbol{B}\boldsymbol{A}_\tau)(\boldsymbol{A}_\tau\boldsymbol{\tau}+\boldsymbol{B}_\tau\boldsymbol{\tau}_c)+\boldsymbol{B}\boldsymbol{B}_\tau\boldsymbol{A}_c\boldsymbol{\tau}_c\ .\end{aligned} \tag{A.45}$$

Der Term in eckigen Klammern entspricht einer weiteren partiellen Ableitung eines Vektors nach $\boldsymbol{\nu}$. Die resultierende Matrix dieses Klammerausdrucks wird mit

$$\boldsymbol{G} = \begin{pmatrix} g_{11} & g_{12} & g_{13}\\ g_{21} & g_{22} & g_{23}\\ g_{31} & g_{32} & g_{33}\end{pmatrix} = \frac{\partial}{\partial \boldsymbol{\nu}}\left(\boldsymbol{J}\left(\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau}\right)\right) \tag{A.46}$$

bezeichnet. Dabei ist zu beachten, dass $\boldsymbol{J}$ von $\boldsymbol{\nu}$ abhängt. Mithilfe der Produktregel lässt sich dieser Ausdruck in

$$\frac{\partial}{\partial \boldsymbol{\nu}}\left(\boldsymbol{J}\left(\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau}\right)\right) = \underbrace{\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{\nu}}\otimes\left(\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau}\right)}_{\boldsymbol{T}} + \underbrace{\boldsymbol{J}\frac{\partial}{\partial \boldsymbol{\nu}}\left(\boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau}\right)}_{\boldsymbol{J}^2} \tag{A.47}$$

zerlegen. Hierbei entspricht die partielle Ableitung einer Matrix $\boldsymbol{J}$ nach einem Vektor $\boldsymbol{\nu}$ einem Tensor. Der $\otimes$-Operator kennzeichnet das Tensorprodukt dieses Tensors mit einem Vektor. Das Resultat dieser Operation ist eine Matrix $\boldsymbol{T}\in\mathbb{R}^{3\times 3}$. Der zweite Summand entspricht dem Quadrat der Jacobi-Matrix. Als Alternative zur Berechnung des Tensorproduktes kann (A.46) auch komponentenweise mithilfe der Vorschrift (A.42) berechnet werden. Die erste Komponente von $\boldsymbol{G}$ ergibt dann

$$\begin{aligned} g_{11} = {} & 6f_{18}u(f_{11}u+f_{12}v+f_{13}r+f_{14}rv+f_{15}ru+f_{16}uv+f_{17}r^2+f_{18}u^3+f_{19}v^3+f_{1,10}r^3+Xb_{11}+Yb_{12}+Nb_{13})\\ & +f_{16}(f_{21}u+f_{22}v+f_{23}r+f_{24}rv+f_{25}ru+f_{26}uv+f_{27}r^2+f_{28}u^3+f_{29}v^3+f_{2,10}r^3+Xb_{21}+Yb_{22}+Nb_{23})\\ & +f_{15}(f_{31}u+f_{32}v+f_{33}r+f_{34}rv+f_{35}ru+f_{36}uv+f_{37}r^2+f_{38}u^3+f_{39}v^3+f_{3,10}r^3+Xb_{31}+Yb_{32}+Nb_{33})\\ & +(f_{12}+f_{14}r+f_{16}u+3f_{19}v^2)(f_{21}+f_{25}r+f_{26}v+3f_{28}u^2)\\ & +(f_{11}+f_{15}r+f_{16}v+3f_{18}u^2)^2\\ & +(f_{31}+f_{35}r+f_{36}v+3f_{38}u^2)(f_{13}+f_{14}v+f_{15}u+2f_{17}r+3f_{1,10}r^2)\ . \end{aligned} \tag{A.48}$$

Die Terme aller Matrixeinträge g_{ij} weisen die gleiche Struktur auf, wobei die letzten drei Zeilen aus (A.48) dem Summand $\boldsymbol{J}^2$ aus (A.47) entsprechen. Für die Bestimmung der Matrixeinträge t_{ij} wird zunächst der Beschleunigungsvektor

$$\boldsymbol{a} = \begin{pmatrix} a_X\\ a_Y\\ a_N\end{pmatrix} = \boldsymbol{F}\cdot\boldsymbol{n}(\boldsymbol{\nu})+\boldsymbol{B}\boldsymbol{\tau} \tag{A.49}$$

definiert. Gleichung (A.48) vereinfacht sich dadurch zu

$$g_{11} = \underbrace{6f_{18}u\cdot a_X + f_{16}\cdot a_Y + f_{15}\cdot a_N}_{t_{11}} + j_{12}\cdot j_{21} + j_{11}^2 + j_{31}\cdot j_{13}\ , \tag{A.50}$$

wobei j_{mn} die Komponenten der Jacobi-Matrix beschreiben. Wird diese Vorgehensweise für alle g_{ij} ausgeführt, dann ergeben sich die folgenden Komponenten der Matrix $\boldsymbol{T}$.

$$\begin{aligned}
t_{11} &= \begin{pmatrix} 6f_{18}u & f_{16} & f_{15} \end{pmatrix} \boldsymbol{a} \\
t_{12} &= \begin{pmatrix} f_{16} & 6f_{19}v & f_{14} \end{pmatrix} \boldsymbol{a} \\
t_{13} &= \begin{pmatrix} f_{15} & f_{14} & (2f_{17} + 6f_{1,10}r) \end{pmatrix} \boldsymbol{a} \\
t_{21} &= \begin{pmatrix} 6f_{28}u & f_{26} & f_{25} \end{pmatrix} \boldsymbol{a} \\
t_{22} &= \begin{pmatrix} f_{26} & 6f_{29}v & f_{24} \end{pmatrix} \boldsymbol{a} \\
t_{23} &= \begin{pmatrix} f_{25} & f_{24} & (2f_{27} + 6f_{2,10}r) \end{pmatrix} \boldsymbol{a} \\
t_{31} &= \begin{pmatrix} 6f_{38}u & f_{36} & f_{35} \end{pmatrix} \boldsymbol{a} \\
t_{32} &= \begin{pmatrix} f_{36} & 6f_{39}v & f_{34} \end{pmatrix} \boldsymbol{a} \\
t_{33} &= \begin{pmatrix} f_{35} & f_{34} & (2f_{37} + 6f_{3,10}r) \end{pmatrix} \boldsymbol{a}
\end{aligned} \tag{A.51}$$

Im Allgemeinen gilt

$$\begin{aligned}
t_{11} &= \begin{pmatrix} \frac{\partial j_{11}}{\partial u} & \frac{\partial j_{12}}{\partial u} & \frac{\partial j_{13}}{\partial u} \end{pmatrix} \boldsymbol{a} \\
t_{12} &= \begin{pmatrix} \frac{\partial j_{11}}{\partial v} & \frac{\partial j_{12}}{\partial v} & \frac{\partial j_{13}}{\partial v} \end{pmatrix} \boldsymbol{a} \\
t_{13} &= \begin{pmatrix} \frac{\partial j_{11}}{\partial r} & \frac{\partial j_{12}}{\partial r} & \frac{\partial j_{13}}{\partial r} \end{pmatrix} \boldsymbol{a} \\
t_{21} &= \begin{pmatrix} \frac{\partial j_{21}}{\partial u} & \frac{\partial j_{22}}{\partial u} & \frac{\partial j_{23}}{\partial u} \end{pmatrix} \boldsymbol{a} \\
t_{22} &= \begin{pmatrix} \frac{\partial j_{21}}{\partial v} & \frac{\partial j_{22}}{\partial v} & \frac{\partial j_{23}}{\partial v} \end{pmatrix} \boldsymbol{a} \\
t_{23} &= \begin{pmatrix} \frac{\partial j_{21}}{\partial r} & \frac{\partial j_{22}}{\partial r} & \frac{\partial j_{23}}{\partial r} \end{pmatrix} \boldsymbol{a} \\
t_{31} &= \begin{pmatrix} \frac{\partial j_{31}}{\partial u} & \frac{\partial j_{32}}{\partial u} & \frac{\partial j_{33}}{\partial u} \end{pmatrix} \boldsymbol{a} \\
t_{32} &= \begin{pmatrix} \frac{\partial j_{31}}{\partial v} & \frac{\partial j_{32}}{\partial v} & \frac{\partial j_{33}}{\partial v} \end{pmatrix} \boldsymbol{a} \\
t_{33} &= \begin{pmatrix} \frac{\partial j_{31}}{\partial r} & \frac{\partial j_{32}}{\partial r} & \frac{\partial j_{33}}{\partial r} \end{pmatrix} \boldsymbol{a} \ .
\end{aligned} \tag{A.52}$$

Die dritte zeitliche Ableitung des Ausgangsvektors wird schließlich mit

$$\dddot{\boldsymbol{y}}_\nu = \underbrace{\left(\boldsymbol{T} + \boldsymbol{J}^2\right)(\boldsymbol{F} \cdot \boldsymbol{n}(\boldsymbol{\nu}) + \boldsymbol{B}\boldsymbol{\tau}) + (\boldsymbol{J}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}_\tau)(\boldsymbol{A}_\tau\boldsymbol{\tau} + \boldsymbol{B}_\tau\boldsymbol{\tau}_c) + \boldsymbol{B}\boldsymbol{B}_\tau\boldsymbol{A}_c\boldsymbol{\tau}_c}_{\mathcal{A}(\boldsymbol{x}_\nu)} + \underbrace{\boldsymbol{B}\boldsymbol{B}_\tau\boldsymbol{B}_c\boldsymbol{u}_\tau}_{\mathcal{B}\boldsymbol{u}_\tau} \tag{A.53}$$

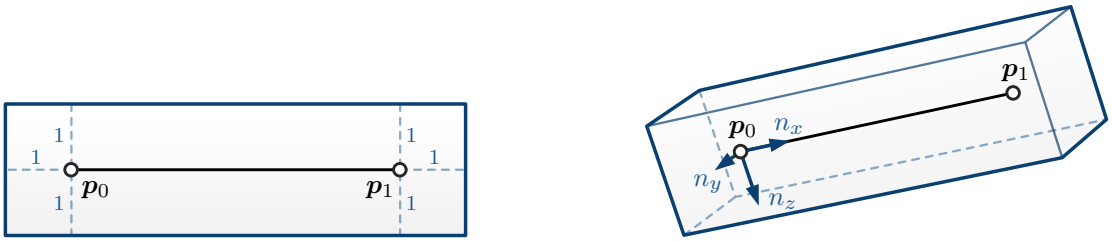
berechnet.

A.5. Sampling innerhalb einer Box um ein Pfadsegment

Für die Bewegungsplanung werden Samples in der Umgebung eines gegebenen Pfades generiert. Dafür wird um jedes Pfadsegment eine dreidimensionale Box gelegt, dessen Größe mithilfe der zwei Tuningparameter $D_{xy} > 0$ und $D_\psi > 0$ festgelegt wird. Zufällige Samples werden mit der Halton-Sequenz erzeugt. Durch die Transformation

$$\boldsymbol{q}_{rand} = \boldsymbol{\sigma}\left(\boldsymbol{A}_{box}\boldsymbol{s} + \boldsymbol{b}_{box}\right) \tag{A.54}$$

können gleichverteilte Samples $\boldsymbol{s} \in (0,1)^3$ der Halton-Sequenz auf den Konfigurationsraum $\mathcal{C} = SE(2)$ abgebildet werden. Im Folgenden werden $\boldsymbol{A}_{box}$ und $\boldsymbol{b}_{box}$ hergeleitet.



(a) Resultierende Größe der Box durch die Erweiterung um 1 in jeder Richtung.

(b) Ausrichtung der Box basierend auf dem Liniensegment von $\boldsymbol{p}_0$ zu $\boldsymbol{p}_1$.

Abbildung A.4.: Schematische Darstellung der Box im euklidischen Raum.

Ein Pfadsegment sei durch zwei Posen $\boldsymbol{q}_0 \in \mathcal{C}$ und $\boldsymbol{q}_1 \in \mathcal{C}$ gegeben. Diese Posen werden zunächst in einen euklidischen Raum überführt, dessen Ursprung in $\boldsymbol{q}_0$ liegt und dessen Raumdimensionen auf die Parameter D_{xy} und D_ψ normiert sind. Aus dieser Transformation ergeben sich

$$\boldsymbol{p}_0 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \tag{A.55}$$

$$\boldsymbol{p}_1 = \begin{pmatrix} 1/D_{xy} & 0 & 0 \\ 0 & 1/D_{xy} & 0 \\ 0 & 0 & 1/D_\psi \end{pmatrix} \boldsymbol{\sigma}\left(\boldsymbol{q}_1 - \boldsymbol{q}_0\right) \ . \tag{A.56}$$

Die zwei Punkte $\boldsymbol{p}_0$ und $\boldsymbol{p}_1$ kennzeichnen das Liniensegment in $\mathbb{R}^3$. Basierend auf diesem Liniensegment wird eine Box definiert, dessen Ausrichtung mit dem Vektor $\boldsymbol{p} = \boldsymbol{p}_1 - \boldsymbol{p}_0$ übereinstimmt und dessen Größe sich aus der Erweiterung des Liniensegmentes um 1 in jeder Richtung ergibt, wie in Abbildung A.4 illustriert. In dem euklidischen Raum ist das Volumen der Box demzufolge mit

$$V = 2 \cdot 2 \cdot (2 + \|\boldsymbol{p}\|_2) \tag{A.57}$$

gegeben. Das Volumen im Konfigurationsraum unter Berücksichtigung der Gewichtung w_ψ des Heading-Winkels wird mit

$$V_{box} = D_{xy}^2 D_\psi w_\psi V \tag{A.58}$$

berechnet. Der Einheitsvektor

$$\boldsymbol{n}_x = \begin{cases} \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|_2} & \text{für } \|\boldsymbol{p}\|_2 > 0 \\ \boldsymbol{e}_x & \text{sonst} \end{cases} \tag{A.59}$$

entspricht der x-Achse des Box-Koordinatensystems und gibt die Richtung der Box in $\mathbb{R}^3$ vor. Falls die Länge Null ist, wird der Einheitsvektor $\boldsymbol{e}_x = (1,0,0)^{\mathrm{T}}$ für $\boldsymbol{n}_x$ gewählt. Die y-Achse des Box-Koordinatensystems wird so definiert, dass diese in der x-y-Ebene des euklidischen Raums liegt. Der zugehörige Einheitsvektor wird mit

$$\boldsymbol{n}_y = \begin{cases} \frac{\boldsymbol{e}_z \times \boldsymbol{n}_x}{\|\boldsymbol{e}_z \times \boldsymbol{n}_x\|_2} & \text{für } \|\boldsymbol{e}_z \times \boldsymbol{n}_x\|_2 > 0 \\ \boldsymbol{e}_x & \text{sonst} \end{cases} \tag{A.60}$$

berechnet, wobei $\boldsymbol{e}_z = (0,0,1)^{\mathrm{T}}$ der Einheitsvektor in z-Richtung ist, welcher dem Normaleneinheitsvektor der x-y-Ebene entspricht. Die z-Achse des Box-Koordinatensystems zeigt in Richtung des Normalenvektors

$$\boldsymbol{n}_z = \boldsymbol{n}_x \times \boldsymbol{n}_y \ . \tag{A.61}$$

Aus den Normaleneinheitsvektoren wird die Rotationsmatrix

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{n}_x & \boldsymbol{n}_y & \boldsymbol{n}_z \end{pmatrix} \tag{A.62}$$

berechnet, mit deren Hilfe Größen aus dem Box-Koordinatensystem in das erdfeste Koordinatensystem transformiert werden können. Ein Sample $\boldsymbol{s} \in (0,1)^3$ wird mit

$$\boldsymbol{p}_{rand} = \boldsymbol{R} \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} + \boldsymbol{R} \begin{pmatrix} 2 + \|\boldsymbol{p}\|_2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \boldsymbol{s} \tag{A.63}$$

in das euklidische Koordinatensystem transformiert. Hierbei entspricht der erste Summand einer Ecke der Box in erdfesten Koordinaten. Im zweiten Summand wird der Sample $\boldsymbol{s}$ mit der Größe der Box skaliert und anschließend mithilfe der Rotationsmatrix $\boldsymbol{R}$ in das erdfeste Koordinatensystem transformiert. Der dreidimensionale Vektor $\boldsymbol{p}_{rand}$ kennzeichnet den Sample im euklidischen Raum und wird durch die Umkehrung von (A.56) mit

$$\boldsymbol{q}_{rand} = \boldsymbol{\sigma} \left(\begin{pmatrix} D_{xy} & 0 & 0 \\ 0 & D_{xy} & 0 \\ 0 & 0 & D_{\psi} \end{pmatrix} \boldsymbol{p}_{rand} + \boldsymbol{q}_0 \right) \tag{A.64}$$

in den Konfigurationsraum $\mathcal{C}$ transformiert. Aus (A.63) und (A.64) resultieren schließlich

$$\boldsymbol{A}_{box} = \begin{pmatrix} D_{xy} & 0 & 0 \\ 0 & D_{xy} & 0 \\ 0 & 0 & D_{\psi} \end{pmatrix} \boldsymbol{R} \begin{pmatrix} 2 + \|\boldsymbol{p}\|_2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \tag{A.65}$$

$$\boldsymbol{b}_{box} = \begin{pmatrix} D_{xy} & 0 & 0 \\ 0 & D_{xy} & 0 \\ 0 & 0 & D_{\psi} \end{pmatrix} \boldsymbol{R} \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} + \boldsymbol{q}_0 \ . \tag{A.66}$$

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[1] United Nations, "UNCTAD - Review of Maritime Transport 2019," 2019, [Online]. Available: https://unctad.org/system/files/official-document/rmt2019_en.pdf, visited on 2022-11-22.

[2] B. v. M. Karel Vanroye, "Der Fachkräftemangel im See- und Binnenschiffsverkehr," 2009, [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/etudes/join/2009/419096/IPOL-TRAN_ET(2009)419096_DE.pdf, visited on 2022-11-22.

[3] L. Mora, X. Wu, and A. Panori, "Mind the Gap: Developments in Autonomous Driving Research and the Sustainability Challenge," *Journal of Cleaner Production*, vol. 275, 09 2020. doi: 10.1016/j.jclepro.2020.124087

[4] Y. Wu, X. Li, J. Gao, and X. Yang, "Research on Automatic Vertical Parking Path-Planning Algorithms for Narrow Parking Spaces," *Electronics*, vol. 12, no. 20, 2023. doi: 10.3390/electronics12204203

[5] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-Time Motion Planning With Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009. doi: 10.1109/TCST.2008.2012116

[6] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *I. J. Robotic Res.*, vol. 29, pp. 485–501, 04 2010. doi: 10.1177/0278364909359210

[7] J. Kurien, P. Nayak, and B. Williams, "Model-based Autonomy for Robust Mars Operations," 04 2001.

[8] R. Washington, K. Golden, J. Bresina, D. Smith, C. Anderson, and T. Smith, "Autonomous rovers for Mars exploration," vol. 1, 02 1999. doi: 10.1109/AERO.1999.794236. ISBN 0-7803-5425-7 pp. 237 – 251 vol.1.

[9] P. Tompkins, A. Stentz, and D. Wettergreen, "Global path planning for Mars rover exploration," in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, vol. 2, 2004. doi: 10.1109/AERO.2004.1367681 pp. 801–815 Vol.2.

[10] Y. Shi, C. Shen, H. Fang, and H. Li, "Advanced Control in Marine Mechatronic Systems: A Survey," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 3, pp. 1121–1131, 2017. doi: 10.1109/TMECH.2017.2660528

[11] J. Adamy, *Nichtlineare Regelungen*. Springer, 2009. ISBN 978-3-642-00793

[12] A. Isidori, *Nonlinear Control Systems*, 3rd ed. Springer, 2004. ISBN 3-540-19916-0

[13] D. Titterton, J. Weston, J. Weston, I. of Electrical Engineers, A. I. of Aeronautics, and Astronautics, *Strapdown Inertial Navigation Technology*, ser. IEE Radar Series. Institution of Engineering and Technology, 2004. ISBN 9780863413582

[14] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. ISBN 9781107156302

[15] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[16] P. Tittmann, *Graphentheorie: Eine anwendungsorientierte Einführung*. Carl Hanser Verlag GmbH Co KG, 2019. ISBN 9783446460522 3., aktualisierte Auflage.

[17] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, 1979. doi: 10.1109/SFCS.1979.10 pp. 421–427.

[18] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010. doi: 10.1109/CDC.2010.5717430 pp. 7681–7687.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *IEEE Transactions on Systems Science and Cybernetics SSC4*, vol. 2, 1968, pp. 100–107.

[20] A. Stentz, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments," *Proc IEEE Int Conf on Robotics and Automation*, vol. 10, 02 2003.

[21] F. Lingelbach, "Path planning using probabilistic cell decomposition," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, 2004. doi: 10.1109/ROBOT.2004.1307193 pp. 467–472 Vol.1.

[22] N. Sleumer and N. Tschichold-Gürmann, "Exact cell decomposition of arrangements used for path planning in robotics," ETH Zurich, Tech. Rep., 1999, technical Report / ETH Zurich, Department of Computer Science 329.

[23] R. Gonzalez, M. Kloetzer, and C. Mahulea, "Comparative study of trajectories resulted from cell decomposition path planning approaches," in *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, 2017. doi: 10.1109/ICSTCC.2017.8107010 pp. 49–54.

[24] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, 2007. doi: 10.1109/ISVD.2007.43 pp. 38–47.

[25] H. Kaluđer, M. Brezak, and I. Petrović, "A visibility graph based method for path planning in dynamic environments," in *2011 Proceedings of the 34th International Convention MIPRO*, 2011, pp. 717–721.

[26] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. doi: 10.1109/70.508439

[27] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[28] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. doi: 10.1109/70.508439

[29] L. Chen, Y. Shan, W. Tian, B. Li, and D. Cao, "A Fast and Efficient Double-Tree RRT*-Like Sampling-Based Planner Applying on Mobile Robotic Systems," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 6, pp. 2568–2578, 2018. doi: 10.1109/TMECH.2018.2821767

[30] I. Noreen, A. Khan, and Z. Habib, "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, 11 2016. doi: 10.14569/IJACSA.2016.071114

[31] X. Gu, M. Han, W. Zhang, G. Xue, G. Zhang, and Y. Han, "Intelligent Vehicle Path Planning Based on Improved Artificial Potential Field Algorithm," in *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*, 2019. doi: 10.1109/HPBDIS.2019.8735451 pp. 104–109.

[32] P. Lin, W. Y. Choi, and C. C. Chung, "Local Path Planning Using Artificial Potential Field for Waypoint Tracking with Collision Avoidance," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020. doi: 10.1109/ITSC45102.2020.9294717 pp. 1–7.

[33] M. Likhachev and D. Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009. doi: 10.1177/0278364909340445

[34] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. Wiley, 2012, 3. edition.

[35] J. Prussing, "Primer Vector Theory and Applications," *Spacecraft Trajectory Optimization*, pp. 16–36, 08 2010. doi: 10.1017/CBO9780511778025.003

[36] F. Topputo and C. Zhang, "Survey of Direct Transcription for Low-Thrust Space Trajectory Optimization with Applications," *Abstract and Applied Analysis*, vol. 2014, pp. 1–15, 06 2014. doi: 10.1155/2014/851720

[37] J. T. Betts, "A Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998. doi: 10.2514/2.4231

[38] A. Rao, "A Survey of Numerical Methods for Optimal Control," *Advances in the Astronautical Sciences*, vol. 135, 01 2010.

[39] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2010, 2. edition.

[40] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006, 2. edition.

[41] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using RRT," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017. doi: 10.1109/SMC.2017.8122814 pp. 1429–1434.

[42] F. Afrasiabi and N. Haspel, "Efficient Exploration of Protein Conformational Pathways using RRT* and MC," 09 2020. doi: 10.1145/3388440.3414705 pp. 1–6.

[43] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," 06 2010. doi: 10.15607/RSS.2010.VI.034

[44] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014. doi: 10.1109/IROS.2014.6942976 pp. 2997–3004.

[45] M. Jordan and A. Perez, "Optimal Bidirectional Rapidly-Exploring Random Trees," 08 2013.

[46] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *2013 IEEE International Conference on Mechatronics and Automation*, 2013. doi: 10.1109/ICMA.2013.6617944 pp. 354–359.

[47] W. Wu, X. Xie, M. Wei, N. Liu, X. Chen, P. Yan, O. Mechali, and L. Xu, *Planetary Rover Path Planning Based on Improved A* Algorithm*, 08 2019, pp. 341–353. ISBN 978-3-030-27537-2

[48] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, "Global path planning on board the mars exploration rovers," in *2007 IEEE Aerospace Conference*, 2007. doi: 10.1109/AERO.2007.352683 pp. 1–11.

[49] G. Matheron, "Designing the path planning algorithm for Mars2020," Jet Propulsion Laboratory, California Institute of Technology, Tech. Rep., 2016.

[50] M. Pivtoraiko, T. Howard, I. N. Nesnas, and A. Kelly, "Field Experiments in Rover Navigation via Model-Based Trajectory Generation and Nonholonomic Motion Planning in State Lattices," in *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS'08)*, 2008.

[51] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011. doi: 10.1109/ICRA.2011.5980409 pp. 2520–2525.

[52] C. Richter, A. Bry, and N. Roy, *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, 04 2016, pp. 649–666. ISBN 978-3-319-28870-3

[53] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 07 2019. doi: 10.1109/LRA.2019.2927938

[54] R. Pepy, A. Lambert, and H. Mounier, "Path Planning using a Dynamic Vehicle Model," in *2006 2nd International Conference on Information & Communication Technologies*, vol. 1, 2006. doi: 10.1109/ICTTA.2006.1684472 pp. 781–786.

[55] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American Control Conference*, 2013. doi: 10.1109/ACC.2013.6579835 pp. 188–193.

[56] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion Planning in Complex Environments Using Closed-loop Prediction," 08 2008. doi: 10.2514/6.2008-7166

[57] R. Lattarulo and J. Perez, "Fast Real-Time Trajectory Planning Method with 3rd-Order Curve Optimization for Automated Vehicles," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020. doi: 10.1109/ITSC45102.2020.9294658 pp. 1–6.

[58] Y. Guo, H. Xu, D. Yao, and L. Li, "A Real-Time Optimization-Based Trajectory Planning Method in Dynamic and Uncertain Environments," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020. doi: 10.1109/ITSC45102.2020.9294321 pp. 1–6.

[59] R. Zaccone, M. Martelli, and M. Figari, "A COLREG-Compliant Ship Collision Avoidance Algorithm," in *2019 18th European Control Conference (ECC)*, 2019. doi: 10.23919/ECC.2019.8796207 pp. 2530–2535.

[60] A. Lazarowska, "A Trajectory Base Method for Ship's Safe Path Planning," *Procedia Computer Science*, vol. 96, pp. 1022–1031, 12 2016. doi: 10.1016/j.procs.2016.08.118

[61] I. B. Hagen, D. K. M. Kufoalor, E. F. Brekke, and T. A. Johansen, "MPC-based Collision Avoidance Strategy for Existing Marine Vessel Guidance Systems," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018. doi: 10.1109/ICRA.2018.8463182 pp. 7618–7623.

[62] J.-y. Zhuang, Y.-m. Su, Y.-l. Liao, and H.-b. Sun, "Motion Planning of USV Based on Marine Rules," *Procedia Engineering*, vol. 15, p. 269–276, 12 2011. doi: 10.1016/j.proeng.2011.08.053

[63] J. Leng, J. Liu, and H. Xu, "Online path planning based on MILP for unmanned surface vehicles," in *2013 OCEANS - San Diego*, 2013. doi: 10.23919/OCEANS.2013.6741114 pp. 1–7.

[64] G. Xia, Z. Han, B. Zhao, and X. Wang, "Local Path Planning for Unmanned Surface Vehicle Collision Avoidance Based on Modified Quantum Particle Swarm Optimization," *Complexity*, vol. 2020, pp. 1–15, 04 2020. doi: 10.1155/2020/3095426

[65] R. Dubey and S. J. Louis, "VORRT-COLREGs: A Hybrid Velocity Obstacles and RRT Based COLREGs-Compliant Path Planner for Autonomous Surface Vessels," in *OCEANS 2021: San Diego – Porto*, 2021. doi: 10.23919/OCEANS44145.2021.9706020 pp. 1–8.

[66] H.-T. L. Chiang and L. Tapia, "COLREG-RRT: An RRT-Based COLREGS-Compliant Motion Planner for Surface Vehicle Navigation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2024–2031, 2018. doi: 10.1109/LRA.2018.2801881

[67] B. Shah, P. Švec, I. Bertaska, A. Sinisterra, W. Klinger, K. von Ellenrieder, M. Dhanak, and S. Gupta, "Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic," *Autonomous Robots*, vol. 40, 10 2016. doi: 10.1007/s10514-015-9529-x

[68] J. Zhang, F. Zhang, Z. Liu, and Y. Li, "Efficient Path Planning Method of USV for Intelligent Target Search," *Journal of Geovisualization and Spatial Analysis*, vol. 3, 12 2019. doi: 10.1007/s41651-019-0035-0

[69] Y. Singh, S. Sharma, R. Sutton, and D. Hatton, "Optimal Path Planning of an Unmanned Surface Vehicle in a Real- Time Marine Environment using a Dijkstra Algorithm," 06 2017. doi: 10.1201/9781315099132-70 pp. 399–402.

[70] Z. Zhang, D. Wu, J. Gu, and F. Li, "A Path-Planning Strategy for Unmanned Surface Vehicles Based on an Adaptive Hybrid Dynamic Stepsize and Target Attractive Force-RRT Algorithm," *Journal of Marine Science and Engineering*, vol. 7, p. 132, 05 2019. doi: 10.3390/jmse7050132

[71] M. Martelli and R. Zaccone, "A random sampling based algorithm for ship path planning with obstacles," 10 2018. doi: 10.24868/issn.2631-8741.2018.018

[72] W. Wang, L. Mateos, S. Park, P. Leoni, B. Gheneti, F. Duarte, C. Ratti, and D. Rus, "Design, Modeling, and Nonlinear Model Predictive Tracking Control of a Novel Autonomous Surface Vehicle," 05 2018. doi: 10.1109/ICRA.2018.8460632

[73] X. Liu, J. Ren, and H. Wang, "Optimal Energy Trajectory Planning and Control for Automatic Ship Berthing," in *2020 39th Chinese Control Conference (CCC)*, 2020. doi: 10.23919/CCC50068.2020.9188805 pp. 1420–1425.

[74] M. Kosch, A. Elkhashap, P. Koschorrek, R. Zweigel, and D. Abel, "Hardware-in-the-Loop Trajectory Tracking and Collision Avoidance of Automated Inland Vessels Using Model Predictive Control," in *2021 European Control Conference (ECC)*, 2021. doi: 10.23919/ECC54610.2021.9655183 pp. 2251–2256.

[75] B.-O. H. Eriksen, G. Bitar, M. Breivik, and A. M. Lekkas, "Hybrid Collision Avoidance for ASVs Compliant with COLREGs Rules 8 and 13-17," in *Frontiers in Robotics and AI*, vol. 7, no. 11, 2020. doi: 10.3389/frobt.2020.00011

[76] G. Bitar, A. B. Martinsen, A. M. Lekkas, and M. Breivik, "Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 14 488–14 494, 2020. doi: 10.1016/j.ifacol.2020.12.1451 21st IFAC World Congress.

[77] E. H. Thyri and M. Breivik, "Collision avoidance for ASVs through trajectory planning: MPC with COLREGs-compliant nonlinear constraints," *MIC Journal*, vol. 43, no. 2, pp. 55–77, 2022. doi: 10.4173/mic.2022.2.2

[78] B. Luders, M. Kothari, and J. How, "Chance Constrained RRT for Probabilistic Robustness to Environmental Uncertainty," *AIAA Guidance, Navigation, and Control Conference*, 08 2010. doi: 10.2514/6.2010-8160

[79] B. Chandler and M. A. Goodrich, "Online RRT and online FMT: Rapid replanning with dynamic cost," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. doi: 10.1109/IROS.2017.8206535 pp. 6313–6318.

[80] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT*: a real-time path planning algorithm based on RRT*," 11 2015. doi: 10.1145/2822013.2822036 pp. 113–118.

[81] D. Webb and J. van den Berg, "Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints," 05 2012.

[82] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005. doi: 10.1109/TRO.2005.852260

[83] ——, "Real-time motion planning for agile autonomous vehicles," in *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, vol. 1, 2001. doi: 10.1109/ACC.2001.945511 pp. 43–49 vol.1.

[84] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT*," in *2011 IEEE International Conference on Robotics and Automation*, 2011. doi: 10.1109/ICRA.2011.5980479 pp. 1478–1483.

[85] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017. doi: 10.1109/ICRA.2017.7989581 pp. 4991–4996.

[86] O. Adiyatov and H. A. Varol, "A novel RRT-based algorithm for motion planning in Dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2017. doi: 10.1109/ICMA.2017.8016024 pp. 1416–1421.

[87] L. Bascetta, I. Arrieta, and M. Prandini, "Flat-RRT*: A sampling-based optimal trajectory planner for differentially flat vehicles with constrained dynamics," *IFAC-PapersOnLine*, vol. 50, pp. 6965–6970, 07 2017. doi: 10.1016/j.ifacol.2017.08.1337

[88] L. I. Reyes Castro, P. Chaudhari, J. Tůmová, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *52nd IEEE Conference on Decision and Control*, 2013. doi: 10.1109/CDC.2013.6760374 pp. 3217–3224.

[89] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: The driftless case," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015. doi: 10.1109/ICRA.2015.7139514 pp. 2368–2375.

[90] C. Yin, B. Xu, X. Chen, Z. Qin, Y. Bian, and N. Sun, "Nonlinear Model Predictive Control for Path Tracking Using Discrete Previewed Points," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020. doi: 10.1109/ITSC45102.2020.9294173 pp. 1–6.

[91] M. Ragaglia, M. Prandini, and L. Bascetta, "Poli-RRT: Optimal RRT-based planning for constrained and feedback linearisable vehicle dynamics," in *2015 European Control Conference (ECC)*, 2015. doi: 10.1109/ECC.2015.7330917 pp. 2521–2526.

[92] T. Brüdigam, F. Luzio, L. Pallottino, D. Wollherr, and M. Leibold, "Grid-Based Stochastic Model Predictive Control for Trajectory Planning in Uncertain Environments," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020. doi: 10.1109/ITSC45102.2020.9294388 pp. 1020–1027.

[93] N. Evestedt, O. Ljungqvist, and D. Axehill, "Motion planning for a reversing general 2-trailer configuration using Closed-Loop RRT," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. doi: 10.1109/IROS.2016.7759544 pp. 3690–3697.

[94] N. Wang, Y. Gao, Z. Zheng, H. Zhao, and J. Yin, "A Hybrid Path-Planning Scheme for an Unmanned Surface Vehicle," in *2018 Eighth International Conference on Information Science and Technology (ICIST)*, 2018. doi: 10.1109/ICIST.2018.8426161 pp. 231–236.

[95] P. Chen, Y. Huang, E. Papadimitriou, J. Mou, and P. van Gelder, "Global path planning for autonomous ship: A hybrid approach of Fast Marching Square and velocity obstacles methods," *Ocean Engineering*, vol. 214, p. 107793, 2020. doi: 10.1016/j.oceaneng.2020.107793

[96] M. Greytak and F. Hover, "Robust Motion Planning for Marine Vehicle Navigation," 01 2008.

[97] H. Niu, Y. Lu, A. Savvaris, and A. Tsourdos, "Efficient Path Planning Algorithms for Unmanned Surface Vehicle," *IFAC-PapersOnLine*, vol. 49, pp. 121–126, 12 2016. doi: 10.1016/j.ifacol.2016.10.331

[98] R. Damerius, T. Hahn, and T. Jeinsch, "Towards Optimal Motion Planning of Surface Vehicles in Confined Waters," 09 2021. doi: 10.1016/j.ifacol.2021.10.112

[99] R. Damerius and T. Jeinsch, "Real-Time Path Planning for Fully Actuated Autonomous Surface Vehicles," in *2022 30th Mediterranean Conference on Control and Automation (MED)*, 2022. doi: 10.1109/MED54222.2022.9837178 pp. 508–513.

[100] R. Damerius, J. R. Marx, and T. Jeinsch, "Fast Trajectory Generation on a Path using Feedback Linearization," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 10 990–10 995, 2023. doi: 10.1016/j.ifacol.2023.10.796 22nd IFAC World Congress.

[101] H. Jond, V. Nabiyev, and R. Benveniste, "Trajectory Planning Using High Order Polynomials under Acceleration Constraint," *Journal of Optimization in Industrial Engineering*, vol. 10, pp. 1–6, 12 2016. doi: 10.22094/joie.2016.255

[102] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, 2011. ISBN 9781119994138

[103] Wasserstraßen- und Schiffahrtsverwaltung des Bundes, "Inland-ENC der WSV," 2022, [Online]. Available: https://www.elwis.de/DE/dynamisch/IENC/, visited on 2022-12-23.

[104] J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *Proceedings of EuroGraphics*, vol. 87, 08 1987.

[105] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, pp. 84–90, 1960. doi: 10.1007/BF01386213

[106] M. Kolář and S. F. O'Shea, "Fast, portable, and reliable algorithm for the calculation of Halton numbers," *Computers & Mathematics with Applications*, vol. 25, no. 7, pp. 3–13, 1993. doi: 10.1016/0898-1221(93)90307-H

[107] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, p. 3–30, jan 1998. doi: 10.1145/272991.272995

[108] S. Ganesan, S. K. Natarajan, and A. Thondiyath, "G-RRT*: Goal-Oriented Sampling-Based RRT* Path Planning Algorithm for Mobile Robot Navigation with Improved Convergence Rate," in *Advances in Robotics - 5th International Conference of The Robotics Society*, ser. AIR2021. New York, NY, USA: Association for Computing Machinery, 2022. doi: 10.1145/3478586.3478588. ISBN 9781450389716. [Online]. Available: https://doi.org/10.1145/3478586.3478588

[109] Z. Tang and H. Ma, "An overview of path planning algorithms," *IOP Conference Series: Earth and Environmental Science*, vol. 804, no. 2, p. 022024, jul 2021. doi: 10.1088/1755-1315/804/2/022024

[110] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull Algorithm for Convex Hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, p. 469–483, dec 1996. doi: 10.1145/235815.235821

[111] H. Freeman and R. Shapira, "Determining the minimum-area encasing rectangle for an arbitrary closed curve," *Commun. ACM*, vol. 18, no. 7, p. 409–413, jul 1975. doi: 10.1145/360881.360919

[112] J. Kautsky and N. NICHOLS, "Robust Pole Assignment in Linear State Feedback," *Int. J. Control, 41*, pp. 1129–1155, 05 1985. doi: 10.1080/0020718508961188

[113] D. H. Douglas and T. K. Peucker, *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*. John Wiley & Sons, Ltd, 2011, ch. 2, pp. 15–28. ISBN 9780470669488

[114] S. Drake, "Converting GPS coordinates [phi, lambda, h] to navigation coordinates (ENU)," *DSTO*, vol. DSTO-TN, 04 2002.

[115] R. Damerius, T. Hahn, I. Karez, A. Schubert, B. Kolewe, and T. Jeinsch, "Guidance, Navigation and Control of Couplable Unmanned Surface Vehicles," in *OCEANS 2024 - Singapore*, 2024. doi: 10.1109/OCEANS51537.2024.10682217 pp. 1–8.

[116] A. Schubert, R. Damerius, C. Rethfeldt, M. Kurowski, T. Jeinsch, and M. Gluch, "Concepts and System Requirements for Automatic Ship Operations," 06 2023. doi: 10.1109/OCEANSLimerick52467.2023.10244661 pp. 1–8.

[117] M. Kanamori and M. Tomizuka, “Dynamic Anti-Integrator-Windup Controller Design for Linear Systems With Actuator Saturation,” *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR*, vol. 129, 01 2007. doi: 10.1115/1.2397146

[118] J. Marx, R. Damerius, and T. Jeinsch, “Linearized Model Predictive Control with Offset-freeness for Trajectory Tracking on Inland Vessels,” 06 2023. doi: 10.1109/MED59994.2023.10185819 pp. 692–697.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit selbstständig und unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt wurde. Es wurden keine anderen Quellen außer den angegebenen verwendet und inhaltlich und wörtlich entnommene Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in der gleichen oder ähnlichen Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Rostock, 21. Januar 2025 Robert Damerius