Marlies Mischinger, BSc

Bahnplanung für automatisierte Fahrzeuge unter Berücksichtigung des Fahrkomforts

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Masterstudium Information and Computer Engineering

eingereicht an der

Technischen Universität Graz

Betreuer

Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Horn

Institut für Automatisierungs- und Regelungstechnik (IRT)

Graz, Mai 2016

Kurzfassung

Das Ziel dieser Arbeit ist es, einen Algorithmus zur Trajektorienplanung für automatisierte Fahrzeuge auszuwählen und zu implementieren. Der Algorithmus muss in der Lage sein gewisse Anforderungen zu erfüllen. Zu diesen gehören die Kollisionsvermeidung und die Berücksichtigung des Fahrkomforts bereits während der Planung. Um Kollisionen zu vermeiden muss der Algorithmus Wege um Hindernisse herum planen, wobei ein definierbarer Sicherheitsabstand zu nicht befahrbaren Bereichen und Hindernissen eingehalten werden muss. Des Weiteren soll der Algorithmus auch in dynamischen Umgebungen einsetzbar sein. Dazu muss der Planer auf Änderungen in der Umwelt reagieren und Pfade gegebenfalls neu planen, um mögliche Kollisionen auszuschließen. Auch im Falle eines Ausweichmanövers muss der Fahrkomfort der resultierenden Trajektorien gewährleistet sein. Ein wichtiges Kriterium zur Beurteilung der Fahrbarkeit der Trajektorie ist die Querbeschleunigung. Aus diesm Grund werden bei der Planung sogenannte Bewegungsprimitive verwendet. Bewegungsprimitive beschreiben die kinematische Bewegung des Fahrzeuges und ermöglichen die Planung von Trajektorien unter Berücksichtigung der Querbeschleunigung. Als Input erhält der Planer eine schwarz-weiß Karte, auf welcher die Trajektorienplanung stattfindet. Dabei stellen weiße Bereiche befahrbare Zonen für den Algorithmus dar. Schwarze Flächen repräsentieren nicht befahrbare Bereiche. Etwaige Hindernisse oder Objekte werden ebenfalls schwarz dargestellt. Der Algorithmus muss von einer gegebenen Startposition zu einer gegebenen Zielposition einen Pfad planen, unter Berücksichtigung der bereits genannten Anforderungen. Anschließend werden die resultierenden Pfade evaluiert. Da das lineare Einspurmodell in der Praxis ein gängiges Fahrzeugmodell zur Beurteilung der Querdynamik ist, wird es auch in dieser Arbeit verwendet. Um der Trajektorie zu folgen wird ein Regler mit einer flachheitsbasierte Vorsteuerung für das lineare Einspurmodell entworfen. Aussagen hinsichtlich Komfort und Fahrbarkeit der Trajektorie können anhand der resultierenden Querbeschleunigungen getroffen werden.

Abstract

This project aims to select an algorithm to design and implement trajectories for automated vehicles and to meet a number of requirements. These requirements are prevention of collisions and having regard to comfort of driving already during planning. To avoid collisions the algorithm has to plan pathways around obstacles by observing a definable safety distance to non driveable areas. Furthermore the algorithm shall be applicable in dynamic surroundings. Therefore the planner has to react on changes of its environment and possibly plan new pathways to avoid collisions. Even in case of an evasive action the resulting trajectories have to ensure driving comfort. One important criterion to evaluate driveability of an trajectory are the lateral dynamics. Therefore, so called motion primitives are being used during a planning process. Motion primitives describe the kinematic motion of a vehicle and allow planning trajectories by regarding lateral dynamics. As an input the planner receives a black and white map on which the trajecories will be planned. On this map white areas represent driveable pathways and black areas represent non driveable areas. Obstacles or other objects are represented in black too. The algorithm has to plan a driveable pathway beginning from a given startpoint to an endpoint on the map. Afterwards the resulting pathways are evaluated. As the linear single-track model is a common model to evaluate lateral dynamics for vehicles it is being used in this project. Thus, to evaluate trajectories a flatness based control for the single-track model is being implemented. Analysing these trajectories make it possible to make statements about comfort on driveability of the trajectory by regarding lateral dynamics.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Danksagung

Die vorliegende Arbeit ist während meines Masterstudiums *Information and Computer Engineering* an der TU Graz und in Kooperation mit dem VIRTUAL VEHICLE Research Center entstanden.
An dieser Stelle möchte ich mich bei all jenen bedanken, die mich im Rahmen dieser Masterarbeit unterstützt und begleitet haben.

Zunächst möchte ich meinem Arbeitskollegen Dipl.-Ing. Peter Wimmer danken, der mir diese Diplomarbeit ermöglichte. Ebenso möchte ich mich bei Dipl.-Ing. Martin Rudigier für die Betreuung während der Masterarbeit bedanken. Vielen Dank euch beiden für die zahlreichen Diskussionen, eure fachliche Unterstützung und euer Vertrauen.

Ebenfalls bedanken möchte ich mich bei meinem Betreuer Herrn Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Horn, Leiter des Instituts für Regelungs- und Automatisierungstechnik, für die fachliche Betreuung und Unterstützung während der gesamten Masterarbeit bedanken.

Mein besonderer Dank gilt meiner Mutter und meiner Schwester, die immer für mich da sind und an mich glauben.
Großer Dank gebührt auch meinem Freund Christof, der in den letzten Monaten auf viel gemeinsame Zeit verzichten musste, mich stets unterstützt und ermutigt hat.

Marlies Mischinger
Graz, Mai 2016

Symbole

$U(q)$	$[V]$	...	Potentialfunktion
$U_{att}(q)$	$[V]$	...	Potentialfunktion mit anziehenden Potentialen
$U_{rep}(q)$	$[V]$	...	Potentialfunktion mit abstoßenden Potentialen
$\Delta\vec{U}(q)$	[-]	...	Gradient einer Potentialfunktion
$\begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix}$	$[-]$	...	partielle Ableitung
$\vec{F}(q)$	$[N]$	...	Kraftfeld
$\vec{F}_{att}(q)$	$[N]$	...	attraktive Kräfte
$\vec{F}_{rep}(q)$	$[N]$	...	repulsive Kräfte
x, y	$[m]$	...	Koordinaten
x_M, y_M	$[m]$	...	Mittelpunkt eines Kreises
x_s, y_s	$[m]$	...	Koordinaten eines Startpunktes
x_g, y_g	$[m]$	...	Koordinaten eines Endpunktes
α_s	$[rad]$	...	Steigung im Startpunkt
α_g	$[rad]$	...	Steigung im Endpunkt
r	$[m]$	...	Kreisradius
k	$[rad]$	...	Steigung
d	$[m]$	...	Entfernung einer Geraden auf y-Achse vom Nullpunkt
b	$[m]$	...	Länge eines Kreisbogens
$B_k^n(t)$	$[-]$	...	Bernsteinpolynom
$\frac{df(x_s)}{dx}$	$[-]$	...	erste Ableitung
λ	$[m]$	...	Länge einer Geraden (ad Bezierkurve)
D_1, D_2	$[-]$	...	Skalierungsfaktoren (ad Bezierkurve)
u	$[s]$	...	Zeit (ad Bezierkurve)
$G(s)$	$[-]$	...	Übertragungsfunktion
a_n	$[-]$	...	Koeffizienten des Nennerpolynoms von G(s)
b_n	$[-]$	...	Koeffizienten des Zählerpolynoms von G(s)
y_f	$[-]$	...	Flacher Ausgang
u	$[-]$	...	Steuerung
$rang$	$[-]$	...	Rang einerMatrix
z	$[-]$	...	Flachheitskoordinaten
z_d	$[-]$	...	Polynom in Flachheitskoordinaten
c_{fi}	$[-]$	...	Koeffizienten des flachen Ausgangs
v_x	$[m/s]$	...	Geschwindigkeit in x-Richtung
v_y	$[m/s]$	...	Geschwindigkeit in y-Richtung
v_z	$[m/s]$	...	Geschwindigkeit in z-Richtung
a_x	$[m/s^2]$	...	Beschleunigung in x-Richtung
a_y	$[m/s^2]$	...	Beschleunigung in y-Richtung
a_z	$[m/s^2]$	...	Beschleunigung in z-Richtung
F_x	$[N]$	...	Kraft in x-Richtung allgemein

F_y	$[N]$	...	Kraft in y-Richtung allgemein
F_z	$[N]$	...	Kraft in z-Richtung allgemein
$c_{\alpha v}$	$[N/-]$	...	Schräglaufsteifigkeit Vorderachse
$c_{\alpha h}$	$[N/-]$	...	Schräglaufsteifigkeit Hinterachse
l	$[m]$	...	Radstand
l_h	$[m]$	...	Abstand des Fahrzeugschwerpunktes von der Hinterachse
l_v	$[m]$	...	Abstand des Fahrzeugschwerpunktes von der Vorderachse
m	$[kg]$	...	Masse Fahrzeug
g	9.81 $[m/s^2]$	...	Erdbeschleunigungskonstante
R	$[m]$	...	Fahrbahnradius
α	$[rad]$	...	Schräglaufwinkel
β	$[rad]$	...	Schwimmwinkel
δ	$[rad]$	...	Radlenkwinkel
ψ	$[rad]$	...	Gierwinkel
$\dot{\psi}, \omega$	$[rad/s]$	...	Gierrate
I_z, θ	$[kgm^2]$	...	Massenträgheitsmoment des Fahrzeuges um die Hochachse

Vektoren und Matrizen

$\mathbf{A}$	...	Systemmatrix
$\mathbf{M}_S$	...	Steuerbarkeitsmatrix
$\mathbf{M}_B$	...	Beobachtbarkeitsmatrix
$\mathbf{T}$	...	Rotationsmatrix
$\mathbf{x}$	...	Zustandsgrößenvektor
$\mathbf{b}$	...	Eingangsvektor
$\mathbf{c^T}$	...	Ausgangsvektor
$\lambda^{\mathbf{T}}$	...	Vektor mit Koeffizienten des flachen Ausgangs

Bemerkungen zur Schreibweise

In dieser Arbeit kennzeichnet

a	eine skalare Größe,
$\mathbf{a}$	einen $n \times 1$-Vektor,
$\mathbf{A}$	eine $m \times n$-Matrix,

wobei m und n beliebige natürliche Zahlen sind.

Inhaltsverzeichnis

1. Einleitung

Der Trend rund um das automatisierte Fahren hat in den letzten Jahren immer mehr an Bedeutung gewonnen und dies wird sich auch in naher Zukunft nicht ändern. Es gibt viele Faktoren die für automatisiertes Fahren sprechen. Dies zeigen die bereits auf dem Markt verfügbaren Fahrerassistenzsysteme. Diese Systeme unterstützen den Fahrer in schwierigen Situationen indem sie warnen, informieren und gegebenfalls sogar regelnd eingreifen. Der Fahrer kann aber immer noch selbst entscheiden, ob er während der Fahrt Unterstützung durch das System möchte oder nicht. Fahrerassistenzsysteme erhöhen schlichtweg die Qualität der Mobilität. So kann beispielsweise die Anzahl der Verkehrsunfälle reduziert und eine erhöhte Sicherheit auf den Straßen garantiert werden. Ein erhöhter Verkehrsfluss und ein geringerer Kraftstoffverbrauch sind nur zwei weitere Gründe warum ein Fokus auf diesem Forschungsschwerpunkt weiterhin bestehen bleibt.
Abhängig vom Automatisierungsgrad wurden von der SAE International (Society of Automotive Engineers) fünf Stufen der Automatisierung definiert [1]. In Level 0 gibt es keine automatisierten Fahrfunktionen, die den Fahrer unterstützen. Der Fahrer selbst hat die absolute Kontrolle über das Fahrzeug. Level 1 definiert bereits assistierendes Fahren. Der Fahrer wird bei der Längs- oder Querführung, beispielsweise durch einen Bremsassistent oder durch einen Tempomat unterstützt. Wird vom *teilautomatisierten* Fahren gesprochen, ist damit Level 2 gemeint. Ab Level 2 übernehmen Fahrerassistenzsysteme für eine gewisse Zeit sowohl Längs- als auch Querführung. Vom Fahrer wird jedoch erwartet das System dauerhaft zu überwachen. In diese Kategorie gehören beispielsweise Parkassistenten, ACC oder Spurhalteassistenten. Level 3 ist die *bedingte Automatisierung*. Ein Automationssystem übernimmt alle dynamischen Fahraspekte. Das System muss nicht mehr dauerhaft vom Fahrer überwacht werden. Wird der Fahrer jedoch aufgefordert die Fahraufgabe wieder zu übernehmen, muss dieser dazu bereit sein. Zur Übernahme der Kontrolle durch den Fahrer wird eine gewisse Zeitreserve berücksichtigt. In Level 4, *hochautomatisiertes Fahren*, kann der Fahrer die Fahrzeugführung an das System übergeben. Ein unmittelbares Eingreifen des Fahrers ist nicht mehr erforderlich. Ab Level 5 spricht man von *autonomen* Fahrzeugen. Das System bewältigt während der ganzen Fahrt alle Situationen automatisch, das heißt alle Aufgaben die sonst von einem Fahrer ausgeübt werden würden.
Die ersten Forschungsprojekte zum automatisierten Fahren liegen viel weiter zurück als man annehmen könnte[2]. Die erste Erfindung wurde 1945 von Ralph Teetor entwickelt, er erfand den Geschwindigkeits-Tempomat, der seit 1958 in Fahrzeugen verwendet wird. 1987 wurde das europaweite EUREKA Prometheus Projekt ins Leben gerufen. (Prometheus: „PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety“). In diesem Projekt entwickelte Daimler Benz, in Kooperation mit der Bundeswehr Universität München, aus dem Mercedes Modell 500 SEL die robotergesteuerten Fahrzeuge VaMP und VITA-3. 1994 bewältigten die Roboterautos des EUREKA Prometheus Projekts bereits folgende autonome Funktionen:

- Spur halten
- im Konvoi fahren

- automatisches Tracking anderer Fahrzeuge
- automatisches Spurwechseln
- autonomes Überholen

Es ist kaum zu glauben, aber diese Roboterautos fuhren damals bereits 1.000 Kilometer auf einer mehrspurigen Autobahn, in gewöhnlichem Verkehr mit einer Geschwindigkeit von bis zu 130 km/h.
Der wahrscheinlich bedeutendste Meilenstein in den letzten 20 Jahren geht von den DARPA Challenges (Defense Advanced Research Projects Agency) aus ([3], [4], [5]). 2004 fand der erste Wettkampf statt. Gefordert war es, ein Fahrzeug so auszurüsten, dass es mit einer Karte und GPS autonom und vollkommen eigenständig fährt. Das Fahrzeug musste in der Lage sein, Hindernisse und Routen zu erkennen. Erlaubte Hilfstmittel waren Radar- und Lidarsensoren sowie Kameras. Das Ziel war es, eine knapp 230 km lange Strecke durch die Mojave Wüste in Kalifornien zu bewältigen, das Preisgeld betrug eine Million USDollar. Von 107 Teams erreichte aber keines der Fahrzeuge die Ziellinie.
Bereits ein Jahr später fand der zweite Wettkampf statt, die *Grand Challenge*. Das Preisgeld wurde verdoppelt, die Bedinungen blieben aber dieselben. Dieses mal gab es gleich fünf Teilnehmer, deren Fahrzeuge es schafften, ins Ziel zu gelangen. Das Gewinnerteam war das Stanford Racing Team mit ihrem Roboterauto 'Stanley' das in knapp 7 h das Ziel erreicht hatte.
2007 findet schließlich die *Urban Challenge* mit einer Änderung der Teststrecke statt. Es wurde eine Stadt in der George Air Force Base, Kalifornien, errichtet, in der eine Wegstrecke von 97 km in unter 6 h zurückgelegt werden sollte. Für diese Challenge hatten sich 89 Teams aus aller Welt beteiligt, davon schafften es 11 Teams ins Finale, aber nur 5 Roboterautos bewältigten die Strecke. Der Gewinner [4], Boss Chevrolet Tahoe, schaffte die Strecke in 4 Stunden und 10 Minuten. Boss wurde von Forschern und Studenten der Carnegie Mellon University, General Motors, Caterpillar, Continental and Intel entwickelt. Boss war mit 17 unterschiedlichen Sensoren ausgestattet, deren Kombination eine 360 Grad Rundumsicht des Fahrzeuges bildeten. Durch das Zusammenspiel aus GPS, Radar- und Lasersensoren sowie Kameras war eine Selbstlokalisierung und Umwelterfassung möglich. Die Systemarchitektur von Boss ist in Abbildung 1 dargestellt. Die Unterteilung in verschiedene Ebenen, hat den Vorteil, dass die einzelnen Ebenen unabhängig voneinander ihre Aufgaben erledigen können.

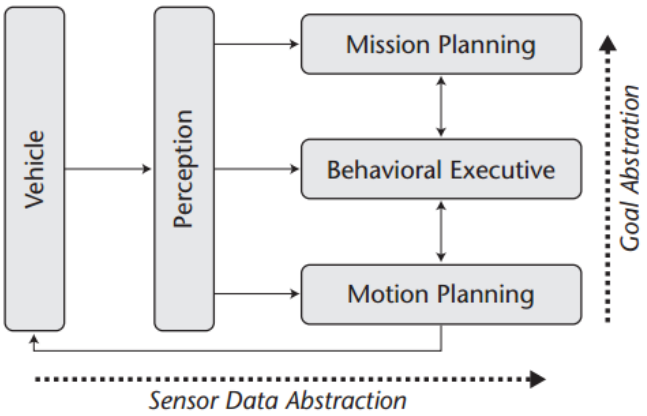


Abbildung 1: Systemarchitektur von Boss [4]

Zu beachten ist der Unterschied zwischen den beiden Ebenen *Mission Planning* und *Motion Planning*. Die Trennung dieser beiden Ebenen führt zu einer neuen Begriffsdefinition, nämlich der *globalen* Pfadplanung und der *lokalen* Pfadplanung ([6], [7]). Die globale Pfadplanung ist gleichzusetzen mit der Funktion eines Navigationssystems. Diese Systeme benötigen eine globale Karte der Umwelt mit allen statischen Hindernissen. Das Ziel der globalen Navigation ist es, ausgehend von der aktuellen Position des Fahrzeuges, eine hindernisfreie Route zu einem gewünschten Ziel auf der globalen Karte zu ermitteln. Im Gegensatz dazu bezieht sich die lokale Navigation auf die nahe Umgebung des Fahrzeuges. Auch die lokale Karte repräsentiert das nahe Umfeld des Fahrzeuges. Diese Karte wird ständig durch Sensorinformationen aktualisiert, wodurch es möglich ist einen Pfad anhand des Straßenverlaufs und der aktuellen Verkehrssituation zu planen. Die lokalen Pfade sind Bahnen, im Gegensatz zu den globalen Pfaden, die von einem Fahrzeug verfolgt werden können. Daher wird die lokale Planung auch als Fahrzeugführungsebene bezeichnet. Die Systemarchtitektur von Boss (1) beinhaltet noch zwei weitere Ebenen. Die *Behavioral Executive*-Ebene ist der sogenannte Verhaltensmanager. Um sich immer entsprechend an die Verkehrsituationen anpassen zu können, benötigt diese Ebene auch Informationen aus der Wahrnehmungsebene *Perception*. In der *Perception*-Ebene werden die Daten von allen Sensoren, welche die Umwelt erfassen, verarbeitet.
Ein wesentlicher Teil dieser Masterarbeit beschäftigt sich mit der lokalen Pfadplanung. Im Fokus stehen dabei die Kollisionsvermeidung und der Fahrkomfort der Trajektorie.

1.1. Zielsetzung

Im ersten Schritt dieser Masterarbeit sollen die Ergebnisse einer ausführlichen Literaturrecherche zum Thema Trajektorienplanung für autonome Fahrzeuge zusammengefasst werden. Es soll ein kurzer Überblick über verwendete Trajektorienplaner, deren Einsatz sowie Eigenschaften verfasst werden. Im nächsten Schritt ist ein Trajektorienplaner auszuwählen und zu implementieren. Die Basis für die Planung ist eine Karte mit *befahrbaren* und *nicht befahrbaren* Bereichen. Befahrbare Raster sind weiß dargestellt, nicht befahrbare Raster schwarz. Der Planer erhält als Input eine Start- und eine Zielposition. Ausgehend von der Startposition bewegt sich der Algorithmus immer nur von einer befahrbaren Zelle zur nächsten befahrbaren Zelle, bis er am Ziel angelangt ist. Der enstehende Pfad setzt sich also aus vielen kurzen Segmenten zusammen. Die einzelnen Pfadsegmente werden mit Hilfe von Bewegungsprimitiven konstruiert. Der Vorteil in der Verwendung von Bewegungsprimitiven liegt darin, die kinematische Bewegung des Fahrzeuges direkt in die Planung miteinfließen zu lassen. Die Idee ist es, die einzelnen Pfadsegmente so zu planen, dass die Querbeschleunigungen entlang dieser Segmente unter einer definierten Schranke liegen. Dadurch müssen auch die Querbeschleunigungen des resultierenden Pfades unter dieser definierten Schranke liegen.
Die Ergebnisse des Bahnplaners werden anschließend analysiert. Dazu ist es notwendig eine Trajektorienfolgeregelung für ein Fahrzeugmodell zu entwerfen. Zur Trajektorienfolge wird eine flachheitsbasierte Steuerung entworfen. Als Fahrzeugmodell wird das lineare Einspurmodell verwendet, da dadurch bereits gute Aussagen hinsichtlich Querdynamik getroffen werden können.

1.2. Aufbau der Arbeit

Der Aufbau der Arbeit teilt sich in zwei Teile. Der erste Teil beinhaltet die Literaturrecherche zum Thema Trajektorienplanung für autonome Fahrzeuge und anschließender Wahl eines Planers. Die Funktionsweise dieses Planers wird in weiterer Folge ausführlich erklärt und einige Details zur Implementierung erläutert.
Im zweiten Teil wird die flachheitsbasierte Steuerung für lineare Systeme vorgestellt. Anschließend wird die Umsetzung der flachheitsbasierten Vorsteuerung für das lineare Einspurmodell gezeigt.
Zum Schluß werden noch Ergebnisse des Trajektorienplaners und deren Evaluierung mit der Vorsteuerung diskutiert.

Das Kapitel 2, *Trajektorienplanung*, enthält eine Zusammenfassung der Literaturrecherche. Es werden verschiedene Trajektorienplaner vorgestellt, von denen manche auch in der Praxis eingesetzt werden. Teilweise wird auch auf die Funktionsweise der Pfadplaner im Detail eingegangen.

In Kapitel 3, *Rapidly Exploring Random Tree*, wird auf den in dieser Diplomarbeit umgesetzten Algorithmus eingegangen. Dabei beschäftigt sich der erste Teil mit elementaren Grundfunktionen des *Rapidly Exploring Random Tree*, sowie möglichen Erweiterungen. Im zweiten Teil dieses Kapitels, werden dann Details zur Implementierung behandelt.

Kapitel 4, *Flachheitsbasierte Vorsteuerung für das lineare Einspurmodell*, beginnt mit einer Einführung in die Flachheitsmethodik. Es wird im Allgemeinen die flachheitsbasierte Vorsteuerung für linare Systeme vorgestellt, gefolgt vom Entwurf einer flachheitsbasierten Steuerung für das lineare Einspurmodell.

In Kapitel 5, *Ergebnisse und Diskussion*, werden die Resultate des Trajektorienplaners gezeigt. Es werden die Vorteile dieses Planers, sowie eventuelle Verbesserungen betrachtet. Die Resultate des Planers werden mittels Vorsteuerung analysiert und deren Ergebnisse diskutiert.

2. Trajektorienplanung

Im Zuge dieser Arbeit versteht man unter Trajektorienplanung (auch Bahn- oder Pfadplanung) das Auffinden eines Weges von einer bekannten Startposition zu einem vorgegebenen Ziel. Die Planung der Trajektorie soll von einem Algorithmus durchgeführt werden, wobei gewisse Forderungen eingehalten werden müssen. Vor allem muss die Fahrbarkeit der Trajektorie gewährleistet sein. Um dies sicher zu stellen, wird bereits während der Planung ein gewisses Maß an Komfort berücksichtigt. Desweiteren müssen Kollisionen vermieden werden. Der Algorithmus muss also fähig sein, nicht nur statischen Objekten auszuweichen, sondern muss auch auf Änderungen in seiner Umwelt reagieren können. In manchen Fällen ist auch eine gewünschte Fahrzeugausrichtung an der Zielposition gefordert. Grundlage für den Algorithmus ist eine Karte, die zur Orientierung dient und wesentliche Informationen für die Planung bereitstellt. Diese Karte ist in feine Raster mit unterschiedlichen Farben und unterschiedlichen Gewichten unterteilt. Beispielsweise bedeuten weiße Raster 'befahrbar' und schwarze Raster 'nicht befahrbar'. Andere Fahrzeuge oder Hindernisse in der Karte werden ebenso schwarz dargestellt. Gegenfahrbahnen können beispielsweise durch Grauabstufungen gekennzeichnet werden. Zusätzlich werden graue Felder durch einen Gewichtungsfaktor stärker bestraft, da diese von einem Fahrzeug nur verwendet werden dürfen, um Hindernissen auszuweichen, sofern sich kein entgegenkommendes Fahrzeug auf der Ausweichroute befindet. Das Management der lokalen Karte spielt eine bedeutende Rolle bei der Planung. Falls Sensoren Änderungen in der Umwelt wahrnehmen, muss die lokale Karte aktualisiert werden. Dass heißt, der Algorithmus arbeitet ständig mit aktuellen Informationen und ist somit fähig, auf Objekte zu reagieren und kann auf diese Weise Kollisionen vermeiden.
In dieser Arbeit werden, neben der Potentialfeldmethode, zwei Gruppen von Trajektorienplanern diskutiert. Dies sind zum einen *gitterbasierte* und zum anderen *probabilistische* Algorithmen. Die Gruppe der gitterbasierten Algorithmen betrachtet jeden möglichen Zustand womit diese *vollständig* sind. Mit anderen Worten: Gibt es eine Lösung, finden diese Algorithmen immer die optimale Lösung vom Start bis zum Ziel. Darin liegt aber auch ein großer Nachteil. Da alle möglichen Zustände betrachtet werden, weisen diese Verfahren eine hohe Rechnenzeit auf und benötigen entsprechend viel Speicherplatz. Abhilfe kann man hier durch Heuristiken schaffen [8]. Auf die Bedeutung von Heuristiken und deren Verwendung, wird im Kapitel zu den gitterbasierten Algorithmen näher eingegangen.
Deutlich schnellere Ergebnisse, erzielt man mit probabilistischen Ansätzen. Das Ergebnis und das Verhalten von probabilistischen Algorithmen ist von Zufallskonfigurationen abhängig. Die Analyse stüzt sich dabei nicht auf den gesamten Konfigurationsraum, sondern nur auf eine Menge an zufällig ausgewählten Punkten. Aus diesem Grund sind diese Algorithmen *probabilistisch vollständig* und nicht *vollständig*, da bei zufällig gestreuten Konfigurationen nicht davon ausgegangen werden kann, dass der optimale Weg zwischen zwei Punkten gefunden wird. Das hat zur Folge, dass diese Algorithmen weder eine optimale Lösung finden, noch irgendeine Lösung finden müssen, jedoch steigt die Wahrscheinlichkeit mit längerer Laufzeit.

2.1. Einteilung der Algorithmen

Im Folgenden werden *gitterbasierte* Algorithmen, *probabilistische* Algorithmen und die *Potentialfeldmethode* vorgestellt. Die gitterbasierten, wie auch die probabilistischen Algorithmen, planen auf einer Karte, die in Zellen gleicher Größe unterteilt ist. Jeder dieser Zelle kann mit unterschiedlichen Informationen belegt sein. In dieser Arbeit stellen weiße Zellen befahrbare Bereiche für den Algorithmus dar. Schwarze Zellen hingegen dürfen nicht befahren werden.
Gitterbasierte Pfadplaner bewegen sich von einer weißen Zelle zu einer benachbarten weißen Zelle, wobei dazu nur gerade Pfadsegmente verwendet werden. Dadurch entstehen sehr eckige Pfade die geglättet werden müssen, um von einem Fahrzeug nachgefahren werden zu können. Dazu gibt es Verfahren, wie zum Beispiel das *Elastische-Bänder*-Verfahren ([9]). Hierbei wird die Form eines Pfades mittels virtuell wirkender Kräfte modifiziert und geglättet. Es ist leicht einzusehen, dass durch die Glättung unerwünschter Rechenaufwand entsteht.
Probabilistische Methoden bieten die Möglichkeit nichtholonome-Modelle in der Planung zu berücksichtigen. Dadurch können sich Roboter zwischen weißen Rastern auf der Karte nicht nur durch gerade Segmente fortbewegen, sondern auch durch Kurvensegmente. Vorteilhaft dieser Methoden ist, dass die resultierenden Pfade nicht geglättet werden müssen.
Beim Potentialfeldverfahren wird ein Potentialfeld erzeugt, das zur Pfadplanung dient. Der Roboter wird in diesem Feld als bewegliches Teilchen angesehen, der bestimmten Kräften ausgesetzt ist. Vereinfacht gesagt, hat das Ziel eine anziehende Wirkung auf den Roboter, während Hindernisse den Roboter abstoßen. Ein wesentlicher Nachteil dieser Methode ist, dass hier keine Möglichkeit besteht nicht-holonome Modelle während der Planung zu berücksichtigen.

2.2. Gitterbasierte Algorithmen

Einer der ersten Algorithmen zur Suche des kürzesten Pfades, zwischen zwei Punkten in einem gewichteten Graphen, ist der Algorithmus von *Dijkstra* [10]. Die Idee dahinter ist es, fortwährend dem kürzesten Pfad vom Startknoten aus zu folgen. Damit ist gewährleistet, wenn man am Zielknoten angekommen ist, dass der gefundene Pfad der kürzeste von allen ist. Ein Beispiel des *Dijkstra* Algorithmus ist in Abbildung 2 zu sehen. Auf den Verbindungslinien zwischen zwei Punkten, den Kanten, sind die Wegkosten zwischen zwei Knoten angegeben. Beim Dijkstra Algorithmus müssen die Kantengewichte allerdings immer positiv sein. In Abbildung 2 ist die kürzeste Verbindung zwischen den Knoten A und I grün eingezeichnet. Im Folgenden wird nun die Funktionsweise des Algorithmus, Schritt für Schritt anhand dieses Beispiels von [10] erklärt.

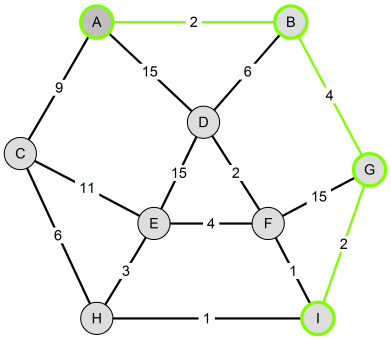


Abbildung 2: Beispiel zum Dijkstra Algorithmus

Gesucht ist also der kürzeste Weg von Knoten A nach Knoten I. Im ersten Schritt wird der Startknoten A rot gekennzeichnet, siehe Abbildung 3 links. Als nächstes, werden alle Kanten die von A wegführen, grün gefärbt und die Knoten, die über diese Kanten erreicht werden können, rot gefärbt (Abbildung 3 rechts).

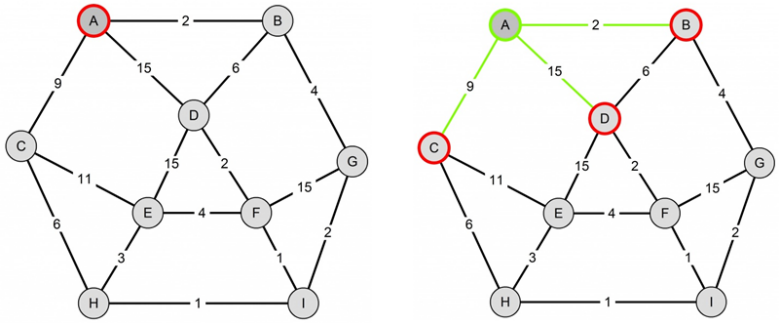


Abbildung 3: Dijkstra Beispiel: Schritt 1 und 2

Abbildung 4 links, zeigt die weitere Vorgensweise. Knoten B ist jener Knoten, mit dem geringsten Kantengewicht zu A, daher wird Knoten B grün eingefärbt. Gleichzeitig wird sein Nachbar, Knoten G, rot eingefärbt. Die Kante zu G wird grün eingefärbt, da sie derzeit die kürzeste Verbindung von A nach G bildet. Die Kante von B nach D wird auch grün gefärbt, da die Verbindungslinie A-B-D mit einer Länge von 8 Einheiten, kürzer ist als die direkte Verbindung von A nach D, mit 15 Einheiten. Daher wird auch die Kante A-D rot gefärbt.
Weiter geht es in Abbildung 4 rechts. Man beginnt wieder mit dem Knoten, der die kürzeste Entfernung vom Startknoten aufweist. Der Pfad bis zu Knoten G hat die kürzeste Entfernung zu A, daher wird dieser gün gefärbt. Alle Kanten die von G wegführen, werden grün gefärbt und alle Knoten die über diese Kanten erreicht werden, werden rot gekennzeichnet.

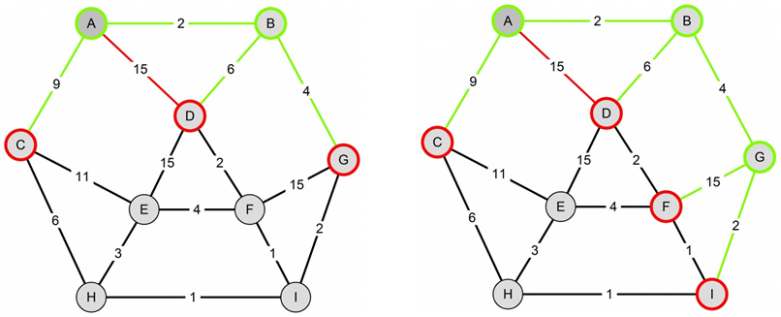


Abbildung 4: Dijkstra Beispiel: Schritt 3 und 4

Es gibt nun zwei rote Knoten, mit der selben kürzesten Entferung von Knoten A. Die Knotenfolge A-B-D hat 8 Einheiten und die Folge A-B-G-I ebenso. Es wird zufällig der Knoten I gewählt, von dem aus weiter gearbeitet wird. In Abbildung 5 links, werden die Nachbarsknoten von I rot eingefärbt. Es gibt zwei Wege die zum Knoten F führen. Diese beiden Wege werden nun verglichen. Die Verbinung mit der Knotenfolge A-B-G-F weist 21 Einheiten auf, die Verbindung A-B-G-I-F mit 9 Einheiten ist jedoch kürzer.

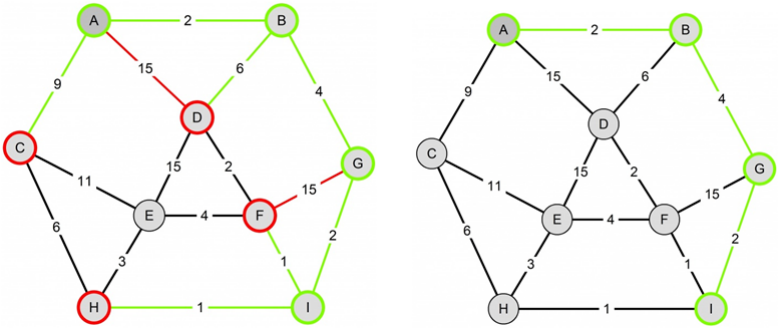


Abbildung 5: Dijkstra Beispiel: Schritt 5 und 6

Im letzten Schritt wird die Kante von I nach F grün gefärbt, die Kante von G nach F rot. Somit ist ein kürzester Weg von A nach I über die Knotenfolge A-B-G-I mit nur 8 Einheiten gefunden (Abbildung 5 rechts).

Nachteilig am Dijkstra Algorithmus ist der hohe Speicherbedarf, da alle Knoten im Speicher gehalten werden müssen. Der Algorithmus ist auch sehr rechenintensiv, da er seine Suche in alle Richtungen ausbreitet und nicht zielgerichtet arbeitet. Es ist auch leicht einzusehen, dass sich eine steigende Knotenanzahl negativ auf die Laufzeit auswirkt. Im Folgenden werden gitterbasierte Algorithmen aufgezählt, die ausgehend vom Dijkstra Algorithmus entwickelt wurden.

2.2.1. A*

Aufbauend auf der Lösung von *Dijkstra* ist der A^* Algorithmus. Dieser versucht durch zusätzliche Informationen dem Suchverfahren eine bestimmte Tendenz zu geben. Der Vorteil liegt in der zielgerichteten Suche, wodurch der Suchaufwand sinkt und so schneller eine Lösung gefunden werden kann. Bei A^* wird die Wahl des nächsten Knotens, durch die Summe aus Wegkosten und einer Heuristik bestimmt.

Definition [11]:
„Heuristik ist die Bezeichnung für ein Lösungsverfahren, das nur zum Teil auf wissenschaftlich gesicherten Erkenntnissen, sondern vorwiegend auf Hypothesen, Analogien oder Erfahrungen aufbaut. Die Güte solcher Verfahren ist deshalb meist nicht beweisbar, sondern wird durch wiederholte Experimente an typischen Problemstellungen nachgewiesen.“

Die Heuristik kann zum Beispiel die Luftlinienentfernung, vom aktuellen Knoten bis zum Zielknoten sein. Wichtig dabei ist, dass die Schätzung der Entfernung zum Ziel, nicht größer ist, als die tatsächliche Entfernung, da man dadurch optimale Pfade eventuell verwirft und somit ein falsches Ergebnis erhält. Die Wahl der Heuristik beeinflusst einerseits die Qualität des Algorithmus, hat aber auch entsprechende Auswirkungen auf die Laufzeit. Der Algorithmus ist optimal sofern die Heuristik zulässig ist, dass heißt es wird immer der kürzeste Weg gefunden [11]. Da auch für diesen Algorithmus die Kenntnis aller Knoten notwendig ist, muss auch hier mit einem hohen Speicheraufwand gerechnet werden. Falls der verfügbare Speicher nicht ausreicht bedeutet dies, dass der Algorithmus auf Grund der vielen Knoten keine Lösung findet und dadurch auf viele Probleme nicht mehr anwendbar ist [12].

Das Beispiel in Abbildung 6 stammt von [11]. Es soll einen Einblick in die Arbeitsweise des A* Algorithmus geben. Gesucht ist der kürzeste Weg von I nach A. Das Kriterium für den nächsten zu besuchenden Knoten ergibt sich aus der Summe der Wegkosten $g(n)$ und einer Heuristik $h(n)$.

$$f(n) = g(n) + h(n) \tag{2.1}$$

In diesem Beispiel wird als Heuristik die Luftlinienentfernung verwendet. Die Luftlinienentfernungen von jedem Knoten zum Zielknoten sind in Tabelle 1 zusammengefasst.

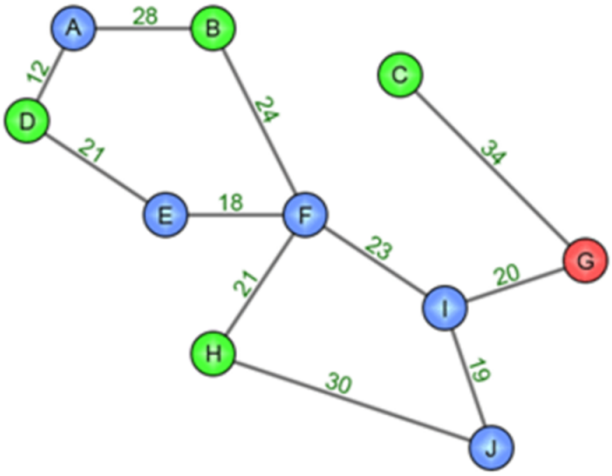


Abbildung 6: A* Beispiel

Luftlinienentfernung	
B nach A	15 $[m]$
C nach A	35 $[m]$
D nach A	11 $[m]$
E nach A	22 $[m]$
F nach A	32 $[m]$
G nach A	60 $[m]$
H nach A	38 $[m]$
I nach A	50 $[m]$
J nach A	60 $[m]$

Tabelle 1: Luftlinienentfernungen

Im ersten Schritt werden die Wegkosten und die Luflinienentfernungen, von jedem Knoten der von I aus besucht werden kann, zusammengefasst, und anschließend wird daraus die Summe gebildet. Siehe Tabelle 2

	F	**G**	**J**
Wegkosten $\mathbf{g(n)}$	23	20	19
Luflinie $\mathbf{h(n)}$	32	60	60
$f(n) = g(n) + h(n)$	55	80	79

Tabelle 2: A* Beispiel: Schritt 1

Da die Summe aus Wegkosten und Luftlinienentfernung für den Knoten F am geringsten ist, wird dieser Knoten als nächstes besucht.
Nun werden alle Knoten überprüft, die von einem der bereits besuchten Knoten, erreicht werden können. Also alle Nachbarsknoten von I und von F. Das sind die Knoten B, E, G, H und J. Die Wegkosten, Luftlinienentfernung und deren Summe ist in Tabelle 3 zusammengefasst.

	B	E	G	H	J
Wegkosten **g(n)**	24+23=47	18+23=41	20	21+23 = 44	19
Luflinie **h(n)**	15	22	60	38	60
$f(n) = g(n) + h(n)$	62	63	80	82	79

Tabelle 3: A* Beispiel: Schritt 2

Der nächste zu besuchende Knoten ist B, da dessen Summe aus Wegkosten und Heuristik am kleinsten ist. Es müssen wieder alle Knoten überprüft werden, die von einem bereits besuchten Knoten aus erreicht werden können. Besuchte Knoten sind I, F und B. Es wird die Summe aus Weg- und Luftlinienentfernung für die Knoten A, E, G, H und J ermittelt. Die Ergebnisse sind in Tabelle 4 zusammengefasst.

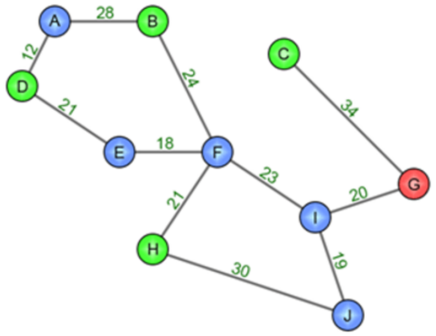


Abbildung 7: A* Beispiel

	A	E	G	H	J
Wegkosten **g(n)**	28+24+23=75	18+23=41	20	21+23 = 44	19
Luflinie **h(n)**	0	22	60	38	60
$f(n) = g(n) + h(n)$	75	63	80	82	79

Tabelle 4: A* Beispiel: Schritt 3

Man könnte nun fälschlicherweise annehmen, dass der nächste zu besuchende Knoten der Zielknoten A ist, da man dann am Ziel wäre. Da als Kriterium aber die Summe aus Wegkosten und Luftlinie ist, wird der Knoten E im nächsten Schritt besucht.
In Tabelle 5 sind die Ergebnisse zusammengefasst.

	A	**D**	**G**	**H**	**J**
Wegkosten **g(n)**	75	62	20	44	19
Luflinie **h(n)**	0	11	60	38	60
$f(n) = g(n) + h(n)$	75	73	80	82	79

Tabelle 5: A* Beispiel: Schritt 4

Es gibt also zwei mögliche Pfade, die zum Zielknoten A führen. Es muss aber noch überprüft werden, welcher der beiden Pfade die günstigeren Gesamtkosten aufweist. Im nächsten Schritt wird Knoten D besucht. Tabelle 6 fasst die Ergebnisse zusammen.

	A über B	**A über D**	**G**	**H**	**J**
Wegkosten **g(n)**	75	74	20	44	19
Luflinie **h(n)**	0	0	60	38	60
$f(n) = g(n) + h(n)$	75	74	80	82	79

Tabelle 6: A* Beispiel: Schritt 5

Wie sich herausgestellt hat, ist der Pfad über den Knoten D kürzer. Der kürzeste Pfad besitzt die Knotenfolge I-F-E-D-A, mit 74 Einheiten.

Es existieren zahlreiche Erweiterungen beziehungsweise Verbesserungen des A*, die versuchen dessen Nachteile zu beheben. Eine Erweiterung von *A** ist beispielsweise *A* Weighted*, bei dem durch zusätzliche Gewichtung der Heuristik, der Suchraum verkleinert wird um so schneller ans Ziel zu gelangen. Jedoch muss man hier Geschwindigkeit gegen Optimalität eintauschen [12].
Da keiner der bereits erwähnten Algorithmen für dynamische Zwecke einsetzbar war, war der nächste Schritt, Algorithmen zu entwickeln, die auf Änderungen im Graph reagieren können. Erste Anstätze wurden durch den *Lifelong Planning A** Algorithmus, kurz *LPA**, realisiert. *LPA** verwendet den gleichen Basisalgorithmus wie *A**, dass heißt es wird der gleiche (optimale) Pfad berechnet. *LPA** berechnet aber wiederholt den kürzesten Pfad zwischen dem Startknoten und dem Zielknoten. Bei Änderungen der Umgebung werden die Ergebnisse von vorherigen Suchdurchläufen genutzt, anstatt komplett von vorne zu beginnen. Aber auch dieser Algorithmus hat Schwächen. *LPA** kann nämlich nicht mit Änderungen während der Bewegung umgehen [13].

2.2.2. D* Lite

Ein wichtiges Ziel war es, Algorithmen zu entwickeln die während einer fortschreitenden Bewegung auf einem bereits gefundenen Pfad, Änderungen im Graphen erkennen und

den Pfad gegebenfalls umplanen. *D** beziehungsweise *D* Lite* behandeln genau diese Problematik. Der Unterschied zwischen den beiden Algorithmen liegt darin, dass bei *D*Lite* die Implementierung wesentlich einfacher und kürzer ist und der Algorithmus auch einfacher erweiterbar ist. Beide Algorithmen berechnen wiederholt den kürzesten Pfad zwischen der aktuellen Position und dem Zielknoten, während man sich bereits in Richtung Ziel bewegt. Auch diese Algorithmen liefern eine optimale Trajektorie vom Startknoten bis zum Zielknoten. Eine detailierte Beschreibung der Funktionsweise des D*Lite ist in [13] zu finden.

2.3. Potentialfeldmethode

Die Ausarbeitung zu dieser Methode beruht auf der Arbeit [14]. Wie der Name bereits sagt, spielt bei dieser Methode eine Potentialfunktion $U(q)$ eine entscheidende Rolle. Zur Erzeugung eines Potentialfeldes werden ein Hauptfeld $U_{att}(q)$ und ein Hindernisfeld $U_{rep}(q)$ berechnet. Die Kombination dieser beiden Felder ergibt ein Gesamtfeld, siehe Abbildung 8. Dabei beinhaltet das Hauptfeld die anziehenden Potentiale und das Hindernisfeld die abstoßenden Potentiale. Abstoßende Potentiale sind Hindernisse, wohingegen das anziehende Potential das Ziel darstellt.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{2.2}$$

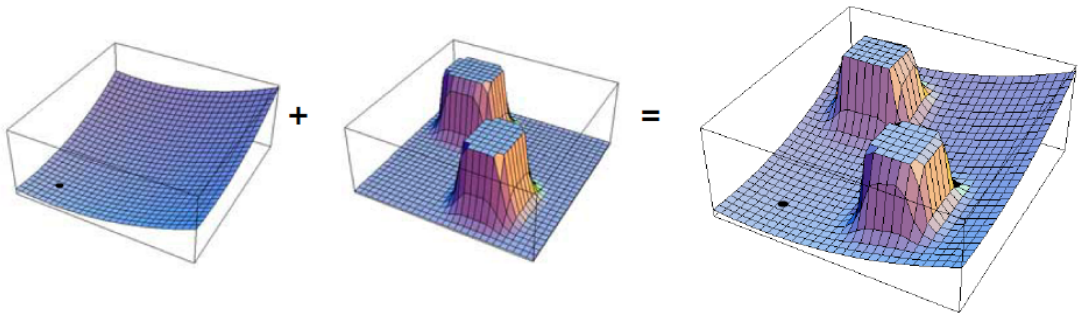


Abbildung 8: Potentialfeld [15]

Grundsätzlich ergibt die Bildung des Gradienten einer Potentialfunktion ein Kraftfeld:

$$\vec{F}(q) = \vec{\nabla} U(q) = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix} \tag{2.3}$$

$\vec{\nabla} U(q)$ bildet den Gradientenvektor in einem Punkt q, er zeigt in die Richtung, welche die Potentialfunktion maximal erhöht. Durch die Negation des Gradientenvektors wird dieser Sachverhalt umgekehrt. Die Kraft wirkt dann in Richtung des Minimums. Der

negative Gradientenvektor ist für die weiteren Betrachtungen von Bedeutung. Ein Roboter in einem Potentialfeld kann als Teilchen angesehen werden, das bestimmten Kräften ausgesetzt ist. Die Summe aus anziehenden und abstoßenden Potentialen, übt eine Kraft auf den Roboter aus, welche dessen Bewegungsrichtung bestimmt. Dabei werden anziehende Kräft als *attraktive* und abstoßende Kräfte als *repulsive* Kräfte bezeichnet. Die Berechnung dieser Kräfte lautet:

$$\vec{F}(q) = -\vec{\nabla}U_{att}(q) - \vec{\nabla}U_{rep}(q) = \vec{F}_{att}(q) + \vec{F}_{rep}(q) \tag{2.4}$$

Ein bekanntes Problem dieser Methode ist die Entstehung lokaler Minima bei der Erzeugung des Potentialfeldes. Dies ist das Phänomen bei u-förmigen Hindernissen wie in Abbildung 9 zu sehen ist. In diesem Fall heben sich die anziehende Kraft des Zieles und die abstoßende Kraft des Hindernisses auf, sodass der Roboter dort hängen bleibt [15].

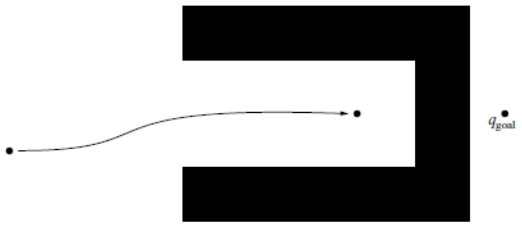


Abbildung 9: lokale Minima [15]

Ein weiteres Problem welches häufig erwähnt wird, sind Oszillationen in engen Passagen, da hier Abstoßungskräfte von beiden Seiten wirken. Fährt das Fahrzeug zu nahe auf der linken Seite eines engen Korridors, wirken Abstoßungskräfte sodass, das Fahrzeug über die Mitte des Pfades fährt, dann aber wirken plötzlich Abstoßungskräfte von der anderen Seite. Dies führt zu oszillierenden Bewegungen der Trajektorie [16].

2.4. Probabilistische Algorithmen

Die im Folgenden erläuterten probabilistischen Algorithem, sind die *Probabilistic Roadmap Method* und der *Rapidly Exploring Random Tree*. Wie bereits erwähnt arbeiten diese Algorithmen zufallsbasiert, womit der Fall eintreten kann, dass eine Lösung der Pfadplanung ausbleibt. Welche Vorteile und Nachteile diese Methoden mit sich bringen, wird in den beiden nächsten Abschnitten erklärt.

2.4.1. Probabilistic Roadmap

Die erste probabilistische Pfadplanungsmethode ist die *Probabilistic Roadmap Method* [17]. Dieses Verfahren gliedert sich in zwei Phasen. In der ersten Phase wird ein Graph konstruiert, die sogenannte Roadmap. Hierfür werden zufällig gewählte Konfigurationen

erzeugt, Nachbarschaften ermittelt und bei der Verbindung von zwei Knoten auf Hindernisse geachtet. Aufgabe der zweiten Phase ist es, einen Weg von einem Startpunkt zu einem Endpunkt, in diesem Graphen zu finden. Dies kann beispielsweise mit einem gitterbasierten Algorithmus wie dem $A*$ erfolgen. Ein bekanntes Problem dieses Verfahrens ist die Streuung von Zufallskonfigurationen in schmalen Passagen. Es ist daher schwierig, Pfade durch enge Gassen zu planen wodurch große Umwege entstehen können. Für solche Fälle existieren aber bereits Abhilfen, um das Samplingverfahren dahingehend zu beeinflussen.

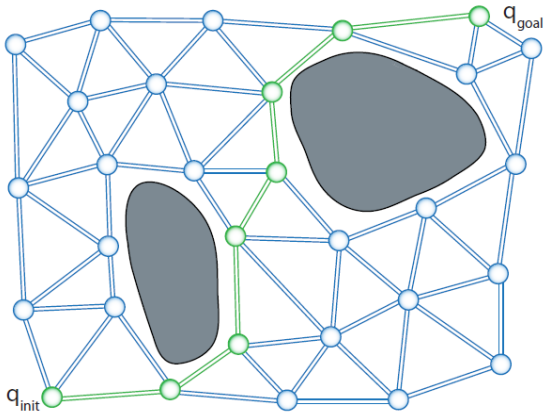


Abbildung 10: Roadmap [18]

Weiters folgt eine detaillierte Beschreibung des Algorithmus durch Pseudocode [17]. Folgende Abkürzungen werden dabei verwendet:

- N_{max} : maximale Anzahl der gestreuten Konfigurationen
- N_{random} : eine zufällig gestreute Konfiguration
- F : der freie Raum im Konfigurationsraum
- $n_{eighbours}$: Nachbarschaften von n
- E : Menge an Verbindungen

Die maximale Anzahl der zufällig gestreuten Konfigurationen ist durch N_{max} festgelegt. Solange N_{max} nicht erreicht ist, werden zufällige Konfigurationen N_{random} erzeugt (Zeile 2). Liegen diese im freien Konfigurationsraum F (Zeile 4), werden alle Nachbarschaften zu dieser Konfiguration ermittelt (Zeile 5). Jede Kante (Verbindung von N_{random} zu einem Nachbarknoten) muss auf Kollisionsfreiheit überprüft werden (Zeile 6 und 7). Ist ein Pfad Kollisionsfrei, wird dieser der Menge aller möglichen Verbindungen E hinzugefügt (Zeile 8).
Es besteht auch die Möglichkeit, Kanten zusätzlich zu gewichten. So kann man zum Beispiel Wege, die knapp an Hindernissen vorbei führen schwerer gewichten als andere.

Dadurch kann bereits bei der Planung ein Sicherheitsabstand zu Hindernissen berücksichtigt werden und man vermeidet eventuelle Kollisionsmöglichkeiten mit Objekten in Kurven.

Algorithm 1 Probabilistic Roadmap

```
1: n_count ← 0
2: while n_count ≠ N_max do
3:     n ← N_random
4:     if n ∈ F then
5:         N = N ∪ n
6:         for all m ∈ n_eighbours do
7:             if {n, m} ∈ F then
8:                 E = E ∪ {n, m}
9:             end if
10:        end for
11:    end if
12: end while
```

2.4.2. Rapidly Exploring Random Tree

Diese Methode ist der *Probabilistic Roadmap Method* sehr ähnlich. Jedoch wird hier kein Graph konstruiert, sondern ein Baum, der sich im gesamten Konfigurationsraum ausgehend von einem Startzustand ausbreitet. Der Algorithmus hat eine Lösung gefunden sobald ein Zielzustand erreicht wurde. Der *RRT* bietet darüber hinaus die Möglichkeit, nicht-holonome Modelle im Zuge der Planung zu berücksichtigen. Da der Aufbau des Baumes aus einfachen Operationen besteht, ist dieser sampling-basierte Ansatz sehr schnell, trotzdem liegt hierin aber auch ein großer Nachteil. Während sich der Baum explorationsartig ausbreitet, wird in keiner Weise auf Optimalität geachtet. Aus diesem Grund gibt es zahlreiche Erweiterungen, die Knoten gegebenfalls umstrukturieren und den Baum somit optimieren. Auch Sampling-Verfahren können abgeändert werden, um dem Baum eine gewisse Tendenz zu geben, in welche Richtung er sich ausbreiten soll. Der *RRT* in Zusammenhang mit den Erweiterungen für die Optimalität wird als *RRT** bezeichnet. Da dieser Algorithmus einen wesentlichen Bestandteil dieser Arbeit darstellt wird diesem Kapitel 3 gewidmet.

2.5. Wahl des Algorithmus

Im Zuge dieser Arbeit wird als Trajektorienplaner für ein automatisiertes Fahrzeug der Rapidly Exploring Random Tree implementiert. An dieser Stelle werden die Vor- und Nachteile der einzelnen Algorithmen noch einmal zusammengefasst und die Gründe für die Wahl des Rapidly Exploring Random Tree aufgezeigt.
Der A*-Algorithmus liefert als Ergebnis zwar den optimalen Pfad zwischen der Start- und der Endposition, aber die lange Rechenzeit und der hohe Speicherbedarf sind nachteilige

Aspekte. Des Weiteren ist der A*-Algorithmus zur dynamischen Pfadplanung nicht einsetzbar.
Der zweite Algorithmus aus der Gruppe der gitterbasierten Algorithmen ist der D* Lite. Dieser wäre grundsätzlich für die dynamische Pfadplanung einsetzbar und hat ebenfalls die Eigenschaft den optimalen Pfad zwischen Start- und Zielposition zu liefern. Aber auch hier gibt es Nachteile. Im D*Lite Algorithmus gibt es keine Möglichkeit ein nicht-holonomes Modell in der Planung zu berücksichtigen. Damit kann während der Planung auch kein Komfort berücksichtigt werden. Des Weiteren sind die gefundenen Pfade sehr kantig und müssen geglättet werden. Dies wirkt sich schließlich negativ auf die Rechenzeit aus.
Die Potentialfeldmethode berechnet zur Trajektorienplanung ein Potentialfeld. Da der Algorithmus auch für dynamische Zwecke einsetzbar sein soll, würde dies bedeuten, dass das Potentialfeld für jede Änderung in der Umwelt neu berechnet werden muss. Es ist leicht einzusehen, dass dies mit intensiven Rechenzeiten verbunden wäre. Des Weiteren gibt es bei der Potentialfeldmethode auch keine Option, ein nicht-holonomes Modell in der Planung zu berücksichtigen.
Die Probabilistic Roadmap Methode ist der Potentialfeldmethode sehr ähnlich. Obwohl diese Methode ein nicht-holonomes Modell in der Planung vorsieht, gibt es einen entscheidenden Nachteil. Bei der Probabilistic Roadmap Methode wird in der ersten Phase der Planung ein Graph konstruiert, auf welchem anschließend geplant wird. Bei jeder Änderung in der Umgebung muss der Graph neu berechnet werden und dies hat lange Rechenzeiten zur Folge.
Der Rapidly Exploring Random Tree setzt bei den Nachteilen der genannten Algorithmen an und versucht diese zu beheben. Ein wesentlicher Vorteil des Rapidly Exploring Random Tree's ist die Berücksichtigung eines nicht-holonomen Modells in der Trajektorienplanung. Dadurch hat man auch die Möglichkeit Komfortkriterien im Zuge der Planung miteinfließen zu lassen. Im Gegensatz zu den gitterbasierten Algorithmen oder der Potentialfeldmethode, müssen die resultierenden Trajektorien des RRT nicht geglättet werden. Diese Aspekte sind der Grund, weshalb der Rapidly Exploring Random Tree in dieser Masterarbeit als Trajektorienplaner für ein automatisiertes Fahrzeug ausgewählt wurde.

3. Rapidly Exploring Random Tree

Der *Rapidly Exploring Random Tree* wurde erfolgreich vom drittplatzierten Team, dem MIT ('Massachusetts Institute of Technology'), bei der DARPA 2007 eingesetzt. Die Idee war es, einen Planer zu entwerfen, der in dynamischer Umgebung einsetzbar ist und auf Änderungen in der Umwelt reagieren kann. Des Weiteren sollten verschiedene Manöver, wie zum Beispiel Einparken oder Überholen, von nur einem Algorithmus bewältigt werden können.
Im Folgenden wird die elementarste Funktion, nämlich der Aufbau des Baumes erklärt [19]. Anschließend folgen zwei Funktionen, *ChooseParent()* und *ReWire()* zur Pfadoptimierung [19] und zwei Samplingverfahren [20].
Zu den einzelnen Funktionen sind jeweils Pseudocode und Abbildungen angeführt, die einem besseren Verständnis dienen. Im Kapitel *Implementierung des RRT* folgen Details zur Implementierung, die im Rahmen dieser Diplomarbeit umgesetzt wurden. Bevor aber auf die einzelnen Erweiterungen im Detail eingegangen wird, werden an dieser Stelle einige Abkürzungen eingeführt, um in weiterer Folge den Pseudocode für die einzelnen Funktionen besser erklären zu können. Die Informationen bezüglich *Rapidly Exploring Random Tree* beruhen auf den Arbeiten [17], [19], [20], [21] und [22].

- z_{new} : neuer Knoten
- $z_{nearest}$: Knoten im Baum mit kürzesten Abstand zu *zrand*
- z_{rand} : neue Zufallskonfiguration (x/y)
- Δ t : Abstand zwischen zwei Knoten in Sekunden
- Z_{near} : Menge der Knoten mit einem Abstand kleiner Δ t zu z_{new}
- z_{near} : ein Element aus Z_{near}
- x_{new} : hindernisfreier Pfad
- u_{new} : Kontrollbefehl zur Ausführung eines Pfades

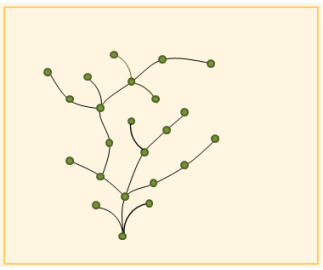
3.1. Aufbau

Der Aufbau des Baumes [19] beginnt damit, dass dem Algorithmus eine Startkonfiguration, bestehend aus Koordinaten und Gierwinkel $x_{start}, y_{start}, \phi_{start}$ und eine gewünschte Endkonfiguration $x_{goal}, y_{goal}, \phi_{goal}$ übergeben wird (Pseudocode siehe *Algorithm* 2 [19]). Die Initialisierung des Baumes erfolgt durch den Startknoten (Zeile 1). Als nächstes wird eine gleichverteilte Zufallskonfiguration z_{rand} erzeugt (Zeile 3) und jener Knoten $z_{nearest}$ im Baum gesucht, mit dem kürzesten Abstand (Zeile 4). Nun bewegt man sich vom Knoten $z_{nearest}$ eine bestimmte Länge Δt in Richtung z_{rand}. Die Endpunktkoordinaten dieses Pfadstückes bilden z_{new} (Zeile 5). Der Pfad zwischen $z_{nearest}$ undz_{new} muss auf Kollisionsfreiheit untersucht werden (Zeile 6). Ist diese Strecke hindernisfrei, wird der Baum um den neuen Knoten z_{new} erweitert (Zeile 7). Es werden solange Zufallskonfigurationen

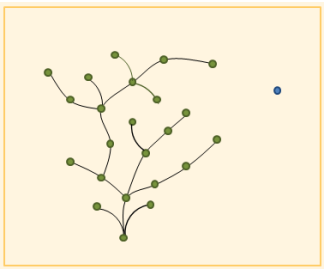
erzeugt bis der Zielzustand erreicht ist. In stark verzweigten Umgebungen kann es vorkommen, dass der Algorithmus sehr lange braucht um den Zielzustand zu erreichen. Aus diesem Grund macht es Sinn den Algorithmus nach einer maximalen Anzahl an Knoten N, oder nach einer bestimmten Zeit terminieren zu lassen, falls kein Pfad gefunden wurde (Abbruchbedingung Zeile 2). Aus der Zeit Δt und der Geschwindigkeit die man für die Planung vorsieht, ergibt sich die Länge eines Pfadstückes: $s = v \cdot \Delta t$. Wie man diese Parameter am besten wählt, ist abhängig vom Einsatz des *RRT*. In Abbildung 11 ist der Aufbau anhand eines Beispiels dargestellt.

Algorithm 2 Build RRT

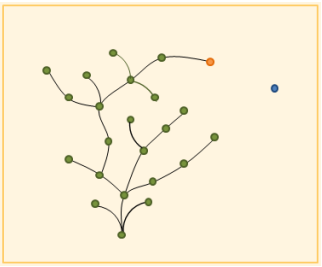
1: $T \leftarrow InsertNode(z_{init})$;
2: **for** $i = 1$ ***to*** N **do**
3: $\quad z_{rand} \leftarrow Sample(i)$;
4: $\quad z_{nearest} \leftarrow Nearest(T, z_{rand})$;
5: $\quad (x_{new}, u_{new}, z_{new}) \leftarrow Steer(z_{nearest}, z_{rand}, \Delta t)$;
6: $\quad$ **if** *ObstacleFree* (x_{new}) **then**
7: $\quad\quad T \leftarrow InsertNode(z_{nearest}, z_{new}, T)$;
8: $\quad$ **end if**
9: **end for**
10: **return** T



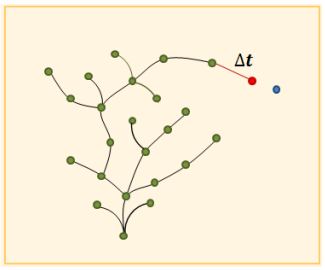
(a) Baum



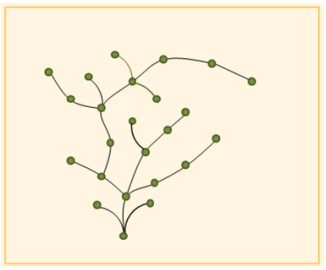
(b) Baum mit neuer Zufallskonfiguration z_{rand}



(c) Knoten $z_{nearest}$ mit kürzestem Abstand



(d) Bewegung um Δt in Richtung $z_{nearest}$



(e) Verbindung von z_{new} mit dem Baum

Abbildung 11: Aufbau RRT

3.2. Bewegungsprimitive

Bei der Pfadplanung geht es nicht nur darum, den kürzesten und hindernisfreien Weg zwischen zwei Punkten zu finden, vielmehr möchte man bereits während der Planung, ein gewisses Maß an Fahrkomfort berücksichtigen. Dazu werden bei der Verbindung zwischen zwei Knoten nicht ausschließlich gerade Segmente verwendet, sondern so genannte Bewegungsprimitive. Bewegungsprimitive beschreiben Bahnen, auf welchen sich ein Roboter unter Berücksichtigung seiner Kinematik bewegen kann. Die Kinematik wird durch Verwendung eines nichtholonomen Modells sichergestellt. Man unterscheidet die Bahnsegmente 'Gerade' und 'Kurve'. Es gibt viele verschiedene Möglichkeiten, Bewegungsprimitive darzustellen. Man muss aber bedenken, je komplexer sich die Berechnung für die Bewegungsprimitive gestaltet, umso mehr Rechenzeit wird benötigt. Die in dieser Arbeit verwendete Art Bewegungsprimitive zu berechnen, bildet folgendes Differentialgleichungssystem ([23]):

$$\begin{aligned} \frac{dx}{dt} &= v * cos(\psi(t)) \\ \frac{dy}{dt} &= v * sin(\psi(t)) \\ \frac{d\psi}{dt} &= \omega \end{aligned}$$

Durch Vorgabe der Geschwindigkeit v und der Gierrate ω, können die genauen Koordinaten sowie der Gierwinkel ψ errechnet werden. Die Grundlage für die Komfortbewertung bildet das Diagramm in Abbilung 12. In diesem Diagramm sind Referenz-Fahrverhalten (Quer- und Längsbeschleunigungen) abgebildet. In der Arbeit von [6] wurden Messungen zum Komfortempfinden von Fahrerassistenzsystemen durchgeführt. Dabei wurden die Probanden anhand eines Fragebogens von [24] in die Kategorien 'dynamisch', 'normal' und 'defensiv' eingeteilt. Die Messwerte in Abbildung 12 stammen von zwölf Testfahrern der Kategorie 'normal', die während einer zweieinhalbstündigen Fahrt aufgezeichnet wurden.

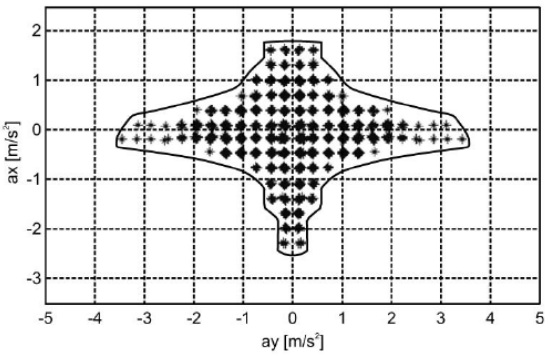


Abbildung 12: Quer- und Längsbeschleunigungen [6]

Aus der Querbeschleunigung a_y lässt sich bei konstanter Geschwindigkeit die zugehörige Gierrate ausrechnen. Dabei ist R der Fahrbahnradius.

$$a_y = \frac{v^2}{R} \quad \Rightarrow \quad R = \frac{v^2}{a_y} \quad \Rightarrow \quad \omega = \frac{v}{R}$$

Für die Trajektorienplanung wählt man am besten ein Set aus einem Bereich an Querbeschleunigungen aus und berechnet dazu das entsprechende Set an Gierraten. Mit diesem Set an Gierraten kann man unterschiedliche Bewegungsprimitive darstellen. Zu beachten ist, dass dieses Set nur für eine konstante Geschwindigkeit gilt. In Abbildung 13 sind eine Gerade und zwei Kurvensegmente dargestellt. Bewegungsprimitive haben zwar den Vorteil, dass der Verlauf an sich glatter ist als beispielsweise der Verlauf eines A*. Aber es wird auch hier eine entsprechende Nachbearbeitung der Trajektorie, beispielsweise mittels Splines, nicht ausbleiben, da der Übergang zwischen zwei Segmenten nicht ruckfrei ist.

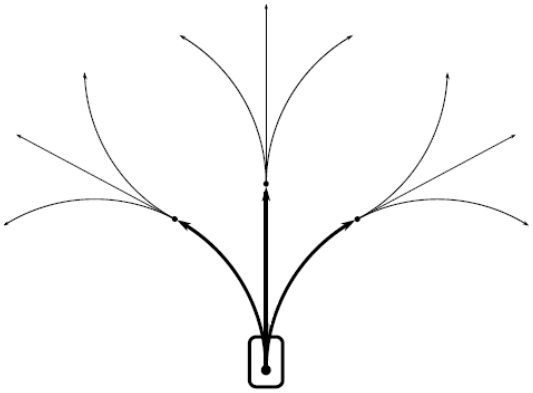


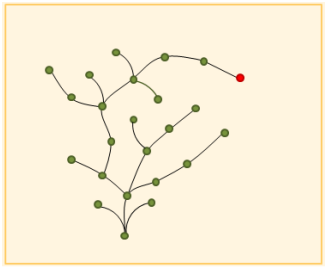
Abbildung 13: Bewegungsprimitive [25]

3.3. *ChooseParent()*

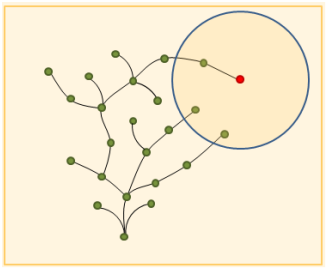
Die Funktion *ChooseParent()*, aus [19], verwendet ein besseres Kriterium zur Verbindung zweier Knoten, im Gegensatz zur Build-Funktion des Standard RRT. Da es sich nun um Erweiterungen des Standard *RRT* handelt, wird ab nun der Algorithmus als *RRT** bezeichnet. Die Idee ist es, zwei Knoten so zu verbinden, dass der neu entstehende Pfad global gesehen, der kürzeste ist. In Abbildung 14 ist die Arbeitsweise von *ChooseParent()* schrittweise dargestellt, die im Folgenden im Detail besprochen wird.
Im ersten Schritt wird ein Kreis um z_{new} erzeugt (Abbildung 14(b)). Die Größe des Kreises sollte dabei so gewählt sein, dass alle anderen Knoten die in diesem Kreis liegen, in der Zeit Δt, von z_{new} aus erreicht werden können. Wie bereits erwähnt, ist die Wahl von Δt abhängig vom Einsatzgebiet des Planers. Nun werden Verbindungen von z_{new} zu allen anderen im Kreis liegenden Knoten z_{near} hergestellt. Man erkennt in Abbildung 14(c), dass z_{new} durch verschiedene Pfade erreicht werden kann. Dazu werden die Kosten von jedem Pfad, der zu z_{new} führt, berechnet. Schließlich wählt man jenen Pfad mit den geringsten Kosten aus (Abbildung 14(d)).
In Zeile 6 des Pseudocodes werden die Gesamtkosten eines Pfades berechnet aus der Summe von $Cost(z_{near})$ und $c(x')$. Dabei ist $Cost(z_{near})$, die Summe aus den einzelnen Pfadlängen von der Wurzel bis z_{near}. Unbekannt sind die Kosten $c(x')$. Die Länge dieses Pfadstückes, von z_{near} zu z_{new}, muss erst berechnet werden. Diese Berechnung ist mit etwas Mehraufwand verbunden und wird im Detail im Abschnitt 4.2 beschrieben.
Vergleicht man das Ergebnis von *ChooseParent()* in Abbildung 14(d), mit dem Ergebnis des Standard-RRT in Abbildung 11(e), dann erkennt man, dass *ChooseParent()* den Knoten z_{new} effizienter mit dem Baum verbindet.

Algorithm 3 ChooseParent($Z_{near}, z_{nearest}, z_{new}, x_{new}$)

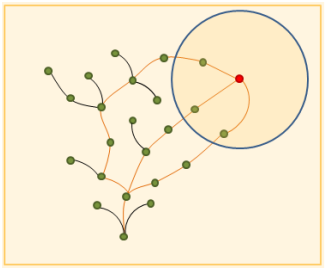
$z_{min} \leftarrow z_{nearest}$;
$c_{min} \leftarrow Cost(z_{nearest}) + c(x_{new})$;
for $z_{near} \in Z_{near}$ **do**
    $(x', T') \leftarrow Steer(z_{near}, z_{new})$;
    if ($ObstacleFree(x') \wedge (x'(T')) = z_{new}$) **then**
        $c' = Cost(z_{near}) + c(x')$;
        if ($c' < c_{min}$) **then**
            $z_{min} \leftarrow z_{near}$;
            $c_{min} \leftarrow c'$;
        end if
    end if
end for
return z_{min}



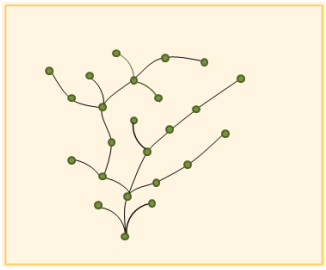
(a)



(b) Kreis mit Radius Δt



(c) alle Pfade die zu z_{new} führen



(d) Pfad mit geringsten Kosten

Abbildung 14: *ChooseParent()*

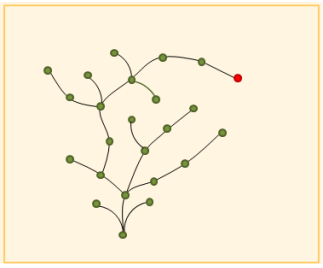
3.4. *ReWire()*

Wie im Falle von *ChooseParent()* wird auch bei *ReWire()* ([19]) ein Kreis mit z_{new} als Mittelpunkt gebildet und alle darin befindlichen Knoten in der Menge Z_{near} zusammengefasst (Abbildung 15(b)). Für jeden Knoten aus z_{near} wird nun überprüft, ob z_{new} ein

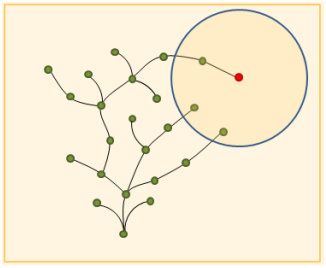
potentieller Vaterknoten sein könnte. Dazu werden Verbindungen von z_{new} zu allen anderen Knoten z_{near} hergestellt und deren Kosten verglichen (Abbildung 15(c) und 15(d)). Wird nun ein Knoten mit z_{new} als Vaterknoten durch geringere Kosten erreicht, dann muss dessen vorherige Vaterbeziehung getrennt werden und eine neue Verbindung hergestellt werden.

Algorithm 4 $ReWire(T, Z_{near}, z_{min}, z_{new})$

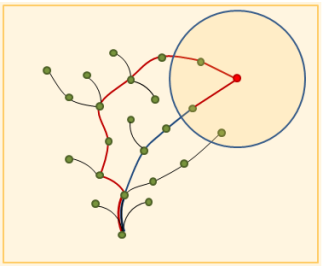
1: **for** $z_{near} \in Z_{near} \setminus \{z_{min}\}$ **do**
2: $(x', T') \leftarrow Steer(z_{near}, z_{new})$;
3: **if** $((ObstacleFree(x') \wedge x'(T') = z_{near}) \wedge (Cost(z_{new}) + c(x') < Cost(z_{near}))$ **then**
4: $T \leftarrow ReConnect(z_{new}, z_{near}, T)$;
5: **end if**
6: **end for**
7: **return** T



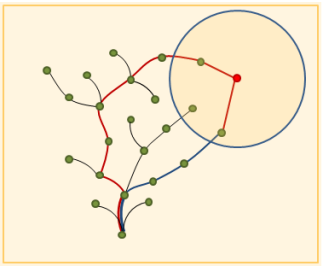
(a)



(b) Kreis mit Radius Δt



(c) z_{new} als Vaterknoten für einen Knoten aus Z_{near}



(d) z_{new} als Vaterknoten für einen Knoten aus Z_{near}

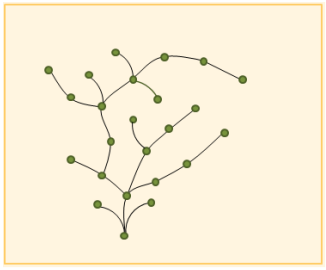


Abbildung 15: *ReWire()*

Natürlich muss bei neuen Verbindungen sichergestellt sein, dass diese hindernisfrei sind. Auch hier sind die Kosten $c(x')$ noch unbekannt und müssen erst ermittelt werden (Pseudocode Zeile 3).

3.5. *local_ bias()*

local_ bias() ([20]) ist nun die erste Variante, neben dem Standard-Samplingverfahren, ein weiteres Samplingverfahren zu verwenden. Dieses Verfahren kommt aber erst zum Einsatz, wenn bereits ein kürzester Pfad gefunden wurde. Die Idee dieser Sample Heuristik ist es, einen bereits gefundenen Pfad stärker zu optimieren, indem eine Zufallskonfiguration zurück geliefert wird, die in Pfadnähe liegt. Diese Zufallskonfiguration lässt sich einfach mit geometrischen Vektoroperationen berechnen, siehe Abbildung 16.

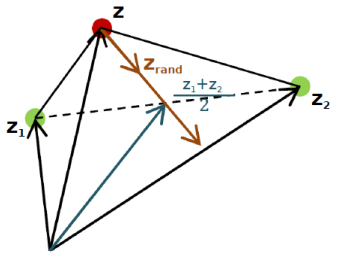


Abbildung 16: Zufallskonfiguration in Pfadnähe

Auch wenn bereits ein kürzester Pfad gefunden wurde, darf man sich nicht ausschließlich darauf fixieren, diesen zu optimieren, sondern muss ein Wachstum in andere Richtungen weiterhin zulassen. Man verwendet dazu einen Schwellwert, um das *local_ bias()* Samplingverfahren und das herkömliche Samplingverfahren mit unterschiedlichem Prozentsatz zu gewichten.

Algorithm 5 lb_sample(i)

1: $p \leftarrow rand(0.0, 1.0)$;
2: **if** $(p > \beta) \wedge (Pathfound)$ **then**
3: **return** $local\_bias\_sample(path)$;
4: **else**
5: **return** $sample(i)$;
6: **end if**

Algorithm 6 local_bias_sample(path)

1: $z \leftarrow random\_node(path)$;
2: $z_1 \leftarrow path(z-1)$;
3: $z_2 \leftarrow path(z+1)$;
4: $z_{tmp} \leftarrow (z_1 + z_2)/2 - z$;
5: $z_{rand} \leftarrow z + \frac{z_{tmp}}{|z_{tmp}|} \cdot rand(r_{min}, r_{max})$;
6: **return** z_{rand};

3.6. *goal_ bias()*

Dieses Samplingverfahren ([20]) macht nur dann Sinn, solange noch kein Zielpfad gefunden wurde. Um dem Baum eine Tendenz zu geben, in welche Richtung er wachsen soll, gibt man als Zufallskonfiguration den Zielzustand zurück. So ist es möglich, das Ziel punktgenau zu erreichen, sofern der Weg hindernisfrei zurückgelegt werden kann. Auch hier wird ein Schwellwert eingeführt, um die Samplingverfahren in unterschiedlichem Maße zum Zug kommen zu lassen.

Algorithm 7 gb_sample(i)

1: $p \leftarrow rand(0.0, 1.0)$;
2: **if** $(p > \alpha) \wedge (\neg Pathfound)$ **then**
3: **return** z_{goal};
4: **else**
5: **return** $sample(i)$;
6: **end if**

Da in gewissen Einsatzbereichen nicht nur ein Erreichen des Zielpunktes gefordert ist, sondern auch eine bestimmte Ausrichtung am Ziel und es bei der Planung durch die Bewegungsprimitive aus Abschnitt 3.2, nicht möglich ist, eine Zielausrichtung zu berücksichtigen, wurde *goal_ bias* in dieser Arbeit modifiziert. Wie dies im Detail aussieht wird im Abschnitt 4.4 beschrieben.

3.7. Mögliche Erweiterungen

In diesem Abschnitt werden zwei Erweiterungen ergänzend erwähnt, die aber im Zuge dieser Masterarbeit nicht implementiert wurden.
Man muss bedenken, dass der Rechenaufwand mit jeder Weiterentwicklung steigt und sich damit die Laufzeit verlängert. Eine der hier erwähnten Erweiterungen, der sogenannte *Bidirektionale RRT* ([22]) setzt genau bei dieser Problematik an.
Das Ziel des *Bidirektionalen RRT* ist es, die Laufzeit deutlich zu verkürzen. Die Idee ist es, zwei Bäume gleichzeitig aufzubauen, wobei sich ein Baum vom Startzustand ausgehend ausbreitet und der zweite vom Endzustand. Durch entsprechende Sampling-Heuristik, versucht man die beiden Bäume jeweils in die entgegengesetzte Richtung stärker wachsen zu lassen. Der Pseudocode ist nachfolgend angeführt zu sehen.

Algorithm 8 ExtExt(z_{init}, z_{goal})

```
1: T_a ← InsertNode(z_init);
2: T_a ← InsertNode(z_goal);
3: for i = 1  to  N do
4:     z_rand ← Sample(i);
5:     if extend(T_a, z_rand) != Trapped then
6:         if (extend(T_b, z_new) = Reached then
7:             return path(T_a, T_b);
8:         end if
9:         swap(T_a, T_b);
10:    end if
11: end for
12: return T
```

Eine weitere interessante Erweiterung ist der *Dynamic Domain RRT* ([21]). Diese Variante beschäftigt sich damit, den Pfad aus engen Passagen herauszuführen. Wie in Abbildung 17 zu sehen ist, kommen nur wenige Zufallskonfigurationen in Frage, um einen Pfad durch den engen Korridor zu planen. Die Idee ist es, Knoten die nahe an Hindernissen liegen einen Radius zu übergeben, sodass nur mehr Sample akzeptiert werden die innerhalb dieser Radien liegen. Durch diese Maßnahme will man erreichen, dass nur Zufallskonfigurationen akzeptiert werden, die auch dazu führen den Weg durch einen engen Korridor herauszufinden. Somit vermeidet man unnötiges Aufrufen der rechenintensiven *Steer()*-Funktion.

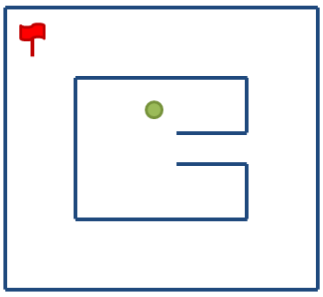


Abbildung 17: Dynamic Domain RRT für enge Passagen

Algorithm 9 DynamicDomainRRT(z_{init})

1: $T \leftarrow InsertNode(z_{init})$;
2: **for** $i = 1$ ***to*** N **do**
3: **repeat**
4: $z_{rand} \leftarrow Sample(i)$;
5: $z_{znearest} \leftarrow Nearest(T, z_{rand})$;
6: **until** $dist(z_{nearest}, z_{rand}) \leq z_{nearest}.radius$;
7: $(x_{new}, z_{new}) \leftarrow Steer(z_{nearest}, z_{rand}, \Delta t)$;
8: **if** $ObstacleFree\ (x_{new})$ **then**
9: $z_{new}.radius \leftarrow \infty$;
10: $T \leftarrow InsertNode(z_{nearest}, z_{new}, T)$;
11: **else**
12: $z_{nearest}.radius \leftarrow R$;
13: **end if**
14: **end for**
15: **return** T

4. Implementierung des RRT

In diesem Kapitel sollen einige Details im Bezug auf die Implementierung der Funktionen *ChooseParent()* und *ReWire()* genauer erläutert werden, welche der Pseudocode aus dem ersten Teil zum RRT offen lässt.
Desweiteren wurde es als sinnvoll empfunden, die Funktion *CalcPathWidth()* zu implementieren, mehr dazu im Abschnitt 4.3.
Um am Ziel auch eine vorgegebene Ausrichtung des Fahrzeuges zu gewährleisten, wurde *goal_ bias()* in dieser Arbeit modifiziert.
Zum Schluß werden Resultate der Trajektorienplanung präsentiert.

4.1. Karte

Spätestens für den online Einsatz des RRT ist es notwendig, die globale und die lokale Karte einzusetzen, um so größere Routen planen und abfahren zu können. Da dies aber den Rahmen dieser Arbeit übersteigt, bleibt man hier bei der einfachen Vorgehensweise. Dem Algorithmus wird lediglich eine lokale Karte übergeben, welche die Umgebung repräsentiert und auf der die Planung stattfindet. Befahrbare Bereiche werden in der Karte durch weiße Felder gekennzeichnet, schwarze Felder sind nicht befahrbare Bereiche oder stellen Hindernisse dar. In dieser Karte wählt man eine beliebige Start- und Zielposition, die dem Algorithmus als Input übergeben werden. Zusätzlich kann man zur Start- und Zielposition jeweils einen gewünschten Gierwinkel angeben. Die Größe dieser Karte beträgt 1000x1000 Pixel, der Abstand zwischen zwei Pixel wurde mit 10 cm festgelegt. Somit bildet eine Karte mit 1000x1000 Pixel in der Realität eine Größe von 100x100 Meter ab. Die in dieser Arbeit verwendeten Karten sind in Abbildung 18 zu sehen. Die Forderungen waren einerseits eine Trajektorie durch einen Kreisverkehr und andererseits eine Trajektorie, um Hindernisse zu planen. Der dargestellte Kreisverkehr hat einen Durchmesser von 25 m und eine Straßenbreite von 8 m. Die Fahrbahn mit den Hindernissen hat eine Breite von 10 m. Auf der Karte mit dem Kreisverkehr wurden die beiden anderen Ausfahrten nicht eingezeichnet, da sie nicht zur Route gehören. Das heißt, dass generell alle Straßen die nicht auf der geplanten Route liegen, als nicht befahrbare Bereiche gekennzeichnet werden.

(a) Kreisverkehr (b) Fahrbahn mit Hindernissen

Abbildung 18: Beispiele für Karten

4.2. Details zu *ChooseParent()* und *ReWire()*

Der Pseudocode zu den Funktionen *ChooseParent()* und *ReWire()* lässt einige Implementierungsdetails offen, die im Folgenden näher erklärt werden. Es wurde bereits im Abschnitt 3.3 erwähnt, dass die Berechnung der Kosten $c(x')$, das sind die Kosten von z_{near} nach z_{new}, erst berechnet werden müssen, um zu überprüfen, welcher Pfad am günstigsten ist. In Abbildung 19 ist dieser Sachverhalt bildlich dargestellt.

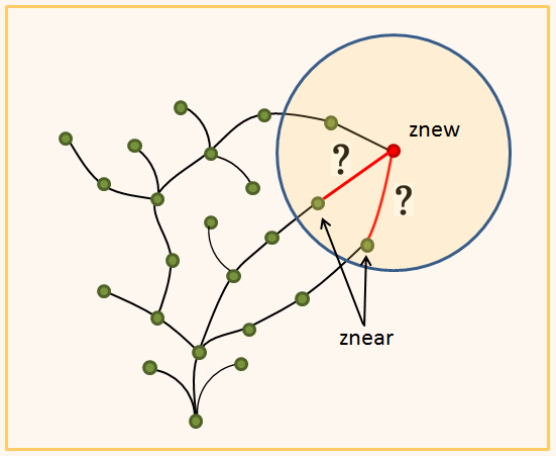


Abbildung 19: Kostenberechnung für die Optimierungsfunktionen

Für die Lösung dieses Problems werden wieder Bewegungsprimitive verwendet, welche bereits in Abschnitt 3.2 detailiert diskutiert wurden. Diesmal müssen die Kenngrößen dieser Bewegungsprimitive aber erst berechnet werden. Zu den Kenngrößen gehören die Zeit Δt und die Gierrate ω. Folgende Informationen sind bekannt: Die Koordinaten der Knoten z_{new} und z_{near} und der Gierwinkel ψ in z_{near}. Mit diesen drei Angaben ist bereits festgelegt, auf welcher Kreisbahn sich das Fahrzeug von z_{near} zu z_{new} bewegt. Im nächsten Schritt sind also der Mittelpunkt und Radius dieser Kreisbahn zu ermitteln. Setzt man in die Kreisgleichung

$$(x - x_M)^2 + (y - y_M)^2 = r^2 \tag{4.1}$$

die Koordinaten des Anfangs- und Endpunktes der Kreisbahn ein, dass heißt z_{near} und z_{new}, erhält man zwei Gleichungen:

$$(x_{new} - x_M)^2 + (y_{new} - y_M)^2 = r^2 \tag{4.2}$$
$$(x_{znear} - x_M)^2 + (y_{znear} - y_M)^2 = r^2 \tag{4.3}$$

Mit der Angabe des Gierwinkels ψ im Punkt z_{near} kann eine Tangente aufgestellt werden:

$$y = k \cdot x + d \qquad \text{mit} \qquad k = \tan(\psi) \tag{4.4}$$

Diese Gleichung nach d umgeformt

$$d = y - k \cdot x \tag{4.5}$$

kann in die Berührbedingung für die Tangente an einen Kreis eingesetzt werden:

$$(k \cdot x_M - y_M + d)^2 = r_2(1 + k^2) \tag{4.6}$$

Die drei Gleichungen (4.2), (4.3) und (4.6) bilden ein Gleichungssystem, aus welchem man den Mittelpunkt MP und den Radius r des Kreises bestimmen kann. Die Lösung dieses Gleichungssystems ermittelt man am besten mit einem Computeralgebrasystem. Mittels Radius r und Geschwindigkeit v lässt sich nun die Gierrate ω bestimmen:

$$\omega = \frac{v}{r} \tag{4.7}$$

Zwar hat man nun die Gierrate ω berechnet, aber es fehlt noch eine wichtige Information. Da es sich bei den Gleichungen (4.2), (4.3) und (4.6), um quadratische Gleichungen handelt, bekommt man nach dessen Lösung keine Information bezüglich Vorzeichen der Winkelgeschwindigkeit ω. In Abbildung 20 ist eine Fallunterscheidung dargestellt, welche die weitere Vorgehensweise, zur Ermittlung des Vorzeichen von ω, unterstützen soll.
Bewegt sich das Fahrzeug im Uhrzeigersinn auf der Kreisbahn, dann liegt der Kreismittelpunkt rechts von der Tangente, die Gierrate ω hat in diesem Fall negatives Vorzeichen.

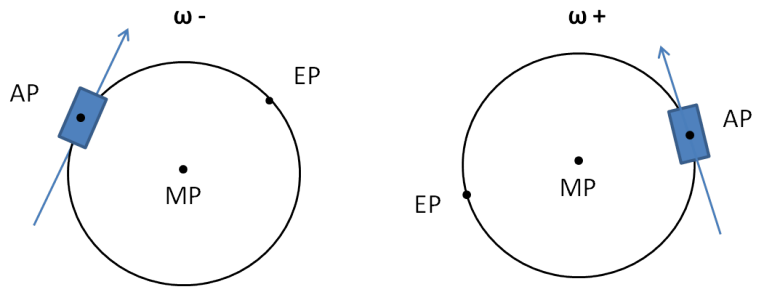


Abbildung 20: Fallunterscheidungen für das Vorzeichen der Gierrate

Umgekehrt, wenn sich das Fahrzeug gegen den Uhrzeigersinn auf der Kreisbahn bewegt, liegt der Kreismittelpunkt auf der linken Seite der Tangente und die Winkeländerung ω ist positiv. Es stellt sich also die Frage, wie man ermittelt, auf welcher Seite der Tangente der Kreismittelpunkt liegt. Anhand der z-Komponente vom Kreuzprodukt und der rechten Hand-Regel, kann die Position eines Punktes zu einer Geraden bestimmt werden.

In Abbildung 21 ist ein Beispiel zur Lagebestimmung eines Punktes bezüglich einer Geraden dargestellt. Die beiden Punkte, C $(3/2/0)$ und D $(1/2/0)$, sowie eine Gerade $\overline{AB}$ sind gegeben. Der Punkt A hat die Koordinaten $(2/1/0)$ und der Punkt B hat die Koordinaten $(2/3/0)$.

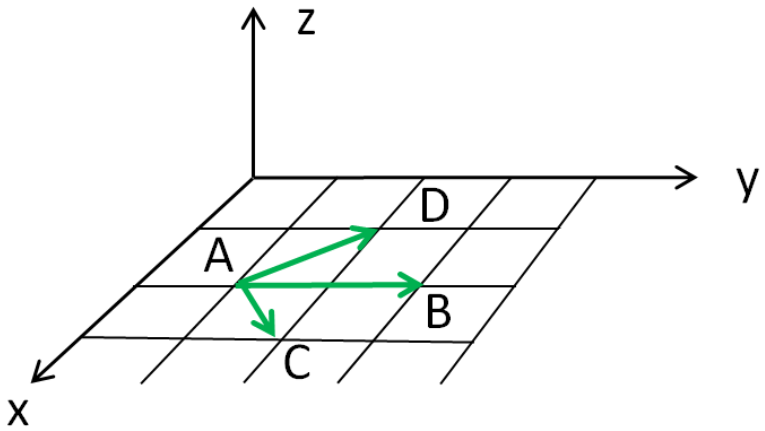


Abbildung 21: Lagebestimmung eines Punktes bezüglich einer Geraden

Zuerst wird die Lage des Punktes C bezüglich der Geraden $\overline{AB}$ untersucht. Dazu bildet

man das Kreuzprodukt der beiden Vektoren:

$$\vec{AC} \times \vec{AB} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} \Rightarrow z - Komponente = 2 \tag{4.8}$$

Die z-Komponente hat ein positives Vorzeichen, das auch die Rechte-Hand-Regel bestätigt. Damit liegt der Punkt C rechts von der Geraden $\overline{AB}$ und die Gierrate ω hat negatives Vorzeichen. Im zweiten Fall bildet man das Kreuzprodukt der beiden Vektoren:

$$\vec{AD} \times \vec{AB} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} \Rightarrow z - Komponente = -2 \tag{4.9}$$

Diesmal erhalten wir ein negatives Vorzeichen für die z-Komponente, die Rechte-Hand-Regel ist verletzt, damit liegt der Punkt D links von der Geraden $\overline{AB}$ und die Gierrate ω hat positives Vorzeichen.

Ist das Vorzweichen von ω bestimmt, sind noch die Länge des Kreisbogens vom Anfangspunkt bis zum Endpunkt, sowie die Zeit, die das Fahrzeug für diesen Kreisbogen benötigt, zu bestimmen. In Abbildung 22 sind vier Fälle dargestellt, aus welchen man die Bogenlänge berechnen kann.

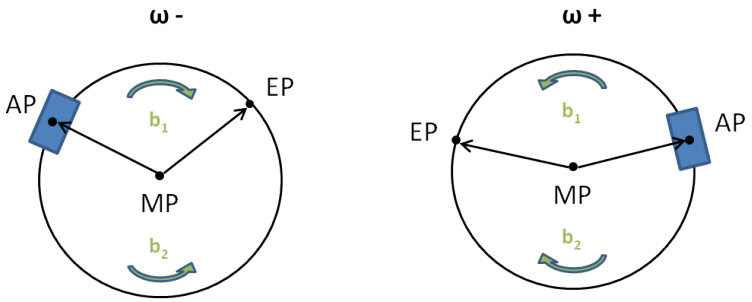


Abbildung 22: Bestimmung des Kreisbogens

- ω^- & EP rechts von der Strecke $\overline{MP, AP} \Rightarrow b_1$
- ω^- & EP links von der Strecke $\overline{MP, AP} \Rightarrow b_2$
- ω^+ & EP links von der Strecke $\overline{MP, AP} \Rightarrow b_1$
- ω^+ & EP rechts von der Strecke $\overline{MP, AP} \Rightarrow b_2$

Auch in diesem Fall macht man sich das Kreuzprodukt zu Nutze, um die Lage des Endpunktes zu bestimmen. Da nun auch die Bogenlänge b bekannt und die Geschwindigkeit v während eines Planungszyklus konstant ist, kann die Zeit Δt, die das Fahrzeug vom Anfangspunkt bis zum Endpunkt benötigt, berechnet werden.
Es ist wichtig, an dieser Stelle zu erwähnen, dass bei der Berechnung der Kenngrößen von Bewegungsprimitiven, berücksichtigt werden muss, dass nicht alle Kreisbögen zulässig sind, da diese Beschränkungen unterliegen, um keine unerwünschten Querbeschleunigungen zu erzielen. Welche Einschränkungen für die Bewegungsprimitive gelten, wurde bereits im Abschnitt 3.2 diskutiert. Die berechneten Gierraten ω sollten jedenfalls nicht höher sein als jene, die grundsätzlich während der Planung verwendet werden. Dadurch wird gewährleistet, dass Pfade die im Zuge einer Umplanung entstehen, ebenso den Forderungen für den Komfort entsprechen. Falls die Berechnung eine zulässige Gierrate ergibt, dann können die Kosten von z_{near} nach z_{new} durch Integration mit Δt und den errechneten Kenngrößen ermittelt werden.

4.3. *CalcPathWidth()*

In dieser Arbeit wurde der *RRT** um eine Funktion erweitert, die zusätzlich zur Kollisionsüberprüfung eines Pfades, die Breite des Pfades überprüft. Der Pfad wird dabei in der Breite nach links und rechts hin untersucht, ob dieser mindestens so breit ist wie ein Fahrzeug plus einem Sicherheitsabstand. Die Idee dahinter ist es, bereits während der Planung zu berücksichtigen, dass ein Fahrzeug welches diesen Pfad fahren würde, auf Grund seiner Breite beziehungsweise Länge, einen definierten Sicherheitsabstand zu Objekten einhält.

4.4. Modifikation von *goal_ bias()*

In der eigenen Arbeit wurde *goal_ bias()* nicht als Samplingverfahren implementiert. Stattdessen wird von jedem Knoten, der dem Baum zuletzt hinzugefügt wurde, die Distanz bis zum Zielknoten berechnet. Ist diese Distanz unter einer vorgegebenen Schranke, werden weitere Kriterien überprüft. Zum einen muss der Pfad auf Kollisionsfreiheit untersucht werden, die Breite des Pfades muss geprüft werden und die Winkeländerung zwischen dem letzten Knoten und dem Zielknoten muss innerhalb eines vorgegebenen Bereiches liegen. Sind diese Kriterien erfüllt, wird eine Bezier-Kurve zwischen diesen beiden Knoten erstellt. Die Änderung der grundsätzlichen Arbeitsweise dieser Funktion, basiert auf der Idee, das Ziel nicht auf Grund einer Zufallszahl zu erreichen, sondern das Ziel zu erreichen, sobald sich eine Möglichkeit anbietet.
Mit Hilfe einer Bezier-Kurve ist es möglich, Winkel im Start- und Endpunkt bei der Planung eines Segments miteinzubeziehen. Wie die Planung mittels Bezier-Kurve im Detail aussieht, wird im nächsten Abschnitt 4.5 genauer erklärt.

4.5. Bezier-Kurve

Die Bezier-Kurve ist eine parametrische Kurve die 1960 unabhängig voneinander, von Pierre Bézier bei Renault und von Paul de Casteljau bei Citroën, für Automobildesign

entwickelt wurde.
Die Inhalte in diesem Abschnitt stammen von [26]. Eine Bezier-Kurve n-ten Grades mit $n+1$ Kontrollpunkten P_0 bis P_n ist definiert als:

$$x(t) = \sum a_k B_k^n(u) \quad \text{mit} \quad 0 \leq k \leq n \quad t \in [0;1] \tag{4.10}$$

wobei

$$B_k^n(u) = \binom{n}{k} \cdot (1-t)^{n-k} \cdot u^k \tag{4.11}$$

als Bernstein-Polynome bezeichnet werden. Bezier-Kurven haben die Eigenschaft, dass sie nicht durch alle Kontrollpunkte verlaufen, sowie dies bei Polynomen oder kubischen Splines der Fall ist. Die Kontrollpunkte spannen ein Kontrollpolygon auf, innerhalb deren konvexer Hülle die Kurve liegt. In Abbildung 23 sind Bezier-Kurven vom Grad 1, 2 und 3 dargestellt.

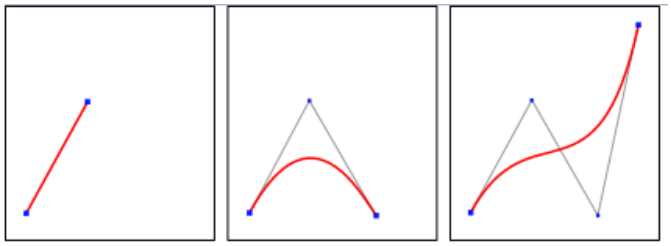


Abbildung 23: Bezierkurven vom Grad 1, 2, und 3 [27]

Üblicherweise haben sich im automobilen Bereich Kurven 3ter Ordnung (siehe Abbildung 24) etabliert, deren Berechnung in weiterer Folge erläutert wird. Gegeben sind der Anfangspunkt P1 (x_s, y_s) und Endpunkt P4 (x_g, y_g), sowie die Winkel α_s und α_g der Tangenten in diesen beiden Punkten. Für die Bezier-Kurve gelten folgende Bedingungen: Die Kurve verläuft durch den Anfangs- und Endpunkt

$$f(x_s) = y_s \tag{4.12}$$
$$f(x_g) = y_g \tag{4.13}$$

Die erste Ableitung der Kurve im Startpunkt, beziehungsweise im Endpunkt, liefert deren Steigung

$$\frac{df(x_s)}{dx} = tan(\alpha_s) \tag{4.14}$$

$$\frac{df(x_g)}{dx} = tan(\alpha_g) \tag{4.15}$$

In Abbildung 24 erkennt man, dass der Kontrollpunkt P2 auf der Tangente liegt die durch

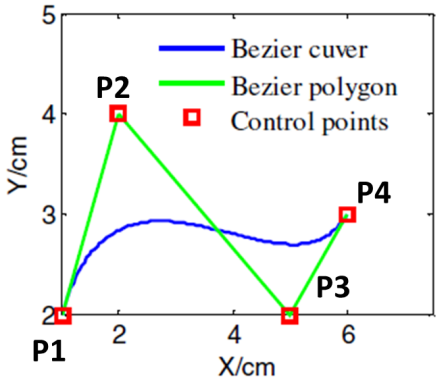


Abbildung 24: Bezier-Kurve

den Anfangspunkt verläuft. Selbiges gilt für den Kontrollpunkt P3 und der Tangente im Endpunkt. Dies wird durch folgende Gleichungen ausgedrückt:

$$(y_2 - y_s) \cdot cos(\alpha_s) = (x_2 - x_s) \cdot sin(\alpha_s) \tag{4.16}$$
$$(y_g - y_3) \cdot cos(\alpha_g) = (x_g - x_3) \cdot sin(\alpha_g) \tag{4.17}$$

Man hat nun zwei Gleichungen, aus welchen die Koordinaten für die Kontrollpunkte berechnet werden können:

$$x_2 = x_s + \lambda_1 * cos(\alpha_s) \tag{4.18}$$
$$y_2 = y_s + \lambda_1 * sin(\alpha_s) \tag{4.19}$$

$$x_3 = x_g - \lambda_2 * cos(\alpha_g) \tag{4.20}$$
$$y_3 = y_g - \lambda_2 * sin(\alpha_g) \tag{4.21}$$

λ_1 und λ_2 entsprechen Längen, die den Einfluss des Richtungsvektors beschreiben. Sie werden aus der Distanz zwischen Start- und Endpunkt berechnet und durch einen Skalierungsfaktor D gewichtet.

$$\lambda_1 = D_1 \cdot \sqrt{(x_s - x_g)^2 + (y_s - y_g)^2} \tag{4.22}$$
$$\lambda_2 = D_2 \cdot \sqrt{(x_s - x_g)^2 + (y_s - y_g)^2} \tag{4.23}$$

Setzt man die Gleichungen für λ_1 und λ_2 in die Gleichungen für (4.18) und (4.19), beziehungsweise in die Gleichungen für (4.20) und (4.21) ein, erhält man die Kontrollpunkte. Nun können die Koordinaten, welche die Bezier-Kurve zwischen Start- und Endpunkt beschreiben, wie folgt ausgedrückt werden:

$$x = x_s(1-u)^3 + 3x_2(1-u)^2u + 3x_3(1-u)u^2 + x_g u^3 \tag{4.24}$$
$$y = y_s(1-u)^3 + 3y_2(1-u)^2u + 3y_3(1-u)u^2 + y_g u^3 \quad \text{mit} \quad u \in [0,1] \tag{4.25}$$

Durch Variation der Skalierungsfaktoren D_1 und D_2 entstehen unendlich viele verschiedene Bezier-Kurven (Abbildung 25), von denen nicht jede die Voraussetzungen der Komforteigenschaften aus 3.2 erfüllt.

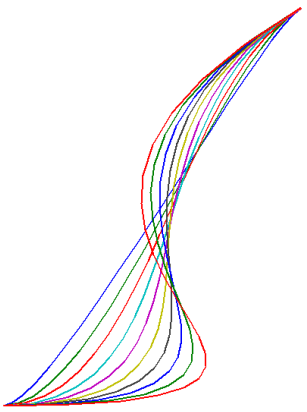


Abbildung 25: Bezier-Kurven

Die Krümmung in einem Punkt lässt sich mittels folgender Formel berechnen

$$\alpha = \arctan(\frac{dy}{dx}) \tag{4.26}$$

Für den Krümmungsradius gilt

$$r = \frac{1}{(d^2y \cdot dx - dy \cdot d^2x)(dx \cdot dx + dy \cdot dy)^{-1.5}} \tag{4.27}$$

Schließlich erhält man die Gierrate ω aus dem Kehrwert der Krümmmung

$$\omega = \frac{1}{r} \tag{4.28}$$

Im Zuge dieser Diplomarbeit wurde kein Konzept ausgearbeitet, wie die Parameter D_1 und D_2 für die Bezier-Kurve gewichtet werden sollen. An jener Stelle im Algorithmus, an der eine Bezier-Kurve geplant wird, ist die Ausgangssituation immer eine andere, da unendlich viele Möglichkeiten existieren, welche Ausrichtung das Fahrzeug zu diesem Zeitpunkt besitzt. Nach welchem Kriterium man die bestmögliche Bezier-Kurve auswählt, wurde hier nicht weiter verfolgt.
Aus diesem Grund wurde im Planer immer eine Bezier-Kurve wie in Abbildung 26 zu

sehen ist, verwendet. Die Kurve wurde mit den Skalierungsfaktoren $D_1 = D_2 = 0.1$ parametriert. Zusätzlich sind hier die Tangenten in den Endpunkten eingezeichnet. Man sieht, dass diese die vorgegebene Steigung von 0° besitzen.

- Startpunkt: (1,1), $\alpha_s = 0°$
- Endpunkt: (8,4), $\alpha_g = 0°$

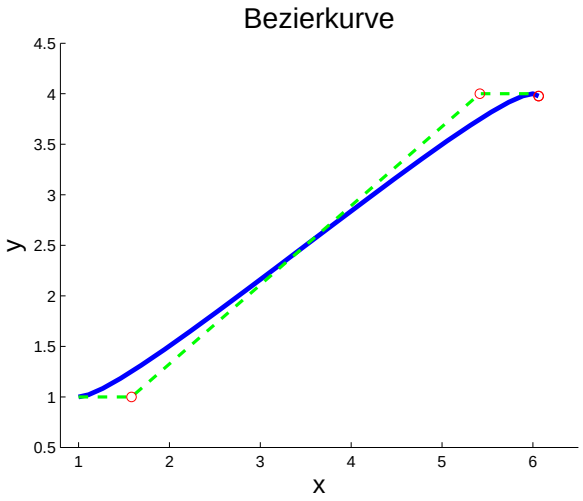


Abbildung 26: Bezier-Kurve

Abbildung 27 enthält die Krümmungen, welche die Bezier-Kurve aufweist. Die Krümmungen dieser Kurve sind am Anfang und am Ende etwas zu hoch. Wie in späteren Kapiteln gezeigt wird, wirken sich diese Krümmungen negativ auf die Fahrbarkeit der Trajektorie aus. Man hätte die Parameter D_1 und D_2 noch feiner skalieren können, jedoch ähnelt die Bezier-Kurve dann immer mehr einer Geraden.

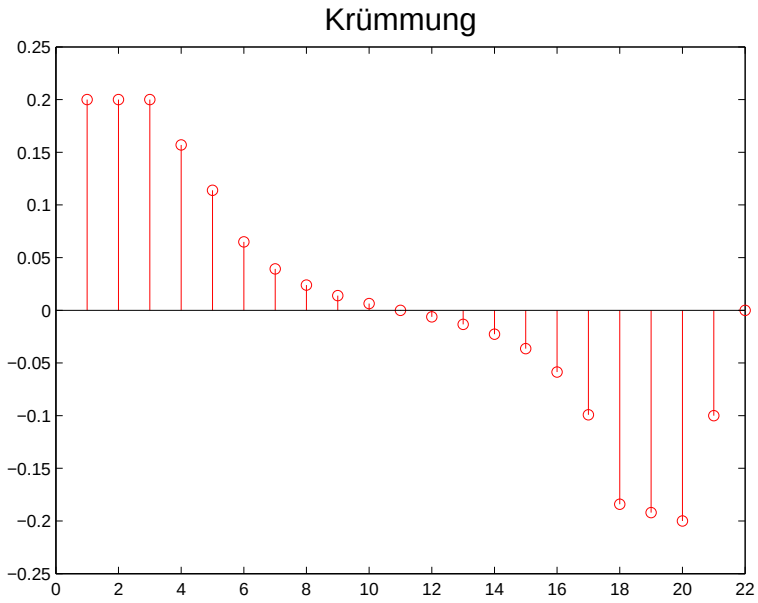


Abbildung 27: Krümmungen der Bezier-Kurve

4.6. Pseudocode zur Implementierung in dieser Arbeit

Algorithm 10 Pseudocode zur Implementierung in dieser Arbeit

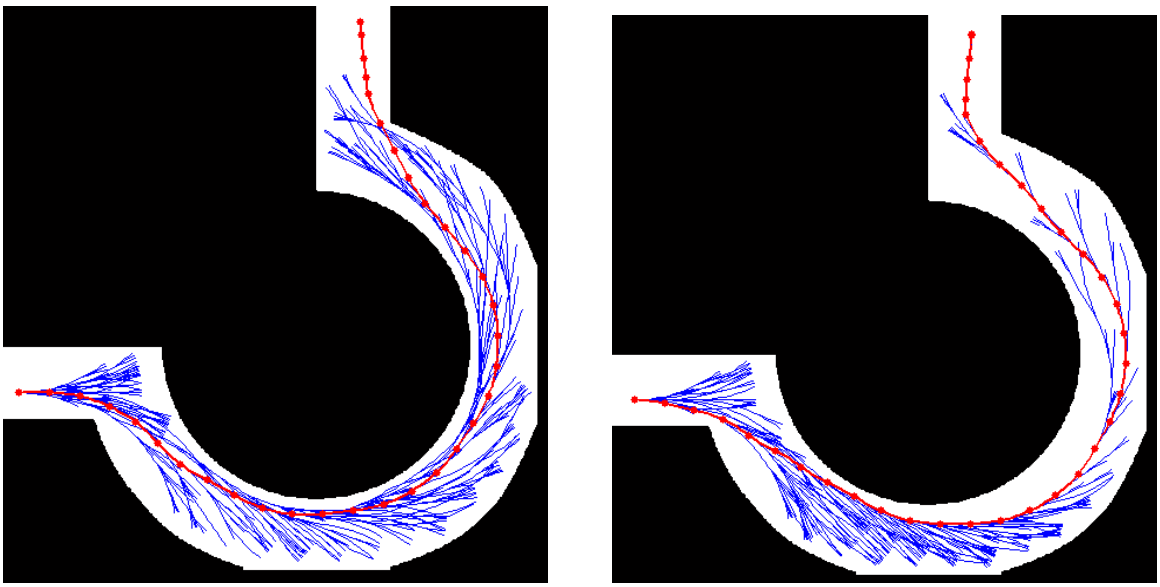
```
1:  while (¬Pathfound()) do
2:      z_rand ← calcZrand();
3:      sample_method ← chooseSampleMethod();
4:      z_nearest ← nearestNeighbour();
5:      path ← Steer();
6:      ctrlCollisionFree(path);
7:      CalcPathWidth(path);
8:      if ( ctrlCollisionFree(path) ∧ ctrlPathWidth(path)) then
9:          Znear ← SetOfZnearest();
10:         if ( Znear ≠ 0 ) then
11:             zmin ← ChooseParent();
12:             addNode(z_min, path);
13:             ReWire();
14:             Bezier();
15:         end if
16:     end if
17: end while
18: return T
```

4.7. Szenarien

Es folgen nun beispielhafte Ergebnisse von Trajektorienplanungen. Im Abschnitt 4.1 wurden bereits jene Karten gezeigt, auf welchen die Trajektorienplanung stattfinden soll. Während der Planung bleibt die Geschwindigkeit konstant. Die Geschwindigkeit für die Trajektorienplanung im Kreisverkehr beträgt 20 km/h, für die Fahrbahn mit den Hindernissen ebenso. Die Bewegungsprimitive wurden in beiden Fällen aus folgenden Querbeschleunigungen berechnet:

- $0\,\mathrm{m/s^2}$
- $1\,\mathrm{m/s^2}$
- $2\,\mathrm{m/s^2}$

Bei der Planung wurde ein Sicherheitsabstand von 1 m zu nicht befahrbaren Bereichen berücksichtigt. Der Algorithmus terminiert, sobald ein minimaler Pfad gefunden wurde und liefert als Ergebnis die Knoten dieses Pfades. Die Abbildungen 28 und 29 zeigen jeweils zwei Resultate der Planer zu jeder Karte. Zu sehen ist der gesamte Baum, der im Zuge der Planung entstanden ist. Der rot gekennzeichnete Pfad, ist der kürzeste Pfad, welchen der Algorithmus gefunden hat. Die Abbildungen 28 und 29 sollen auch verdeutlichen, dass man als Resultat vom RRT-Algorithmus nur mehr die einzelnen Knoten des minimalen Pfades erhält. Die Pfade zwischenden einzelnen Knoten werden auf Grund des zu hohen Speicherbedarfs während der Planung nicht gespeichert.



(a) (b)

Abbildung 28: Ergebnisse - Kreisverkehr

(a) (b)

Abbildung 29: Ergebnisse - Fahrbahn mit Hindernissen

Um die Ergebnisse dieser Planung auf Komfort zu überprüfen und zu beurteilen, ob diese Art der Trajektorienplanung vernünftig ist, wird eine Vorsteuerung für ein Fahrzeugmodell entworfen. Um das querdynamische Verhalten des Fahrzeuges zu analysieren, bietet sich für erste Untersuchungen das Einspurmodell an. Die Thematik um die Vorsteuerung und die Ergebnisse der Trajektorienfolgeregelung werden in den nächsten Kapiteln behandelt.

5. Flachheitsbasierte Vorsteuerung für das lineare Einspurmodell

In der Praxis werden häufig Vorsteuerungen zur Trajektorienfolgeregelung oder zur Realisierung eines Arbeitspunktwechsels entworfen. Dazu wird meist eine Zwei-Freiheitsgrad-Regelkreisstruktur verwendet, wie sie in Abbildung 30 zu sehen ist. Vorteile dieser Struktur sind einerseits die verbesserte Führungsdynamik auf Grund der Vorsteuerung, ohne die Stabilität der Regelung zu beeinflussen, sowie der Entwurf der Steuerung und Regelung unabhängig voneinander.

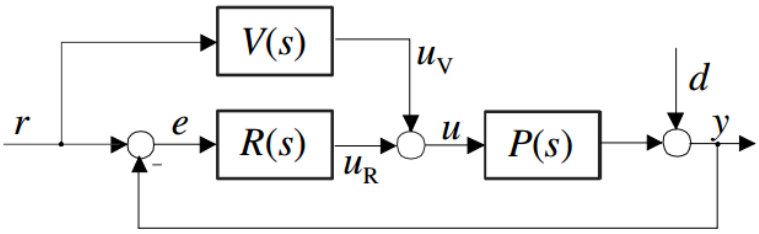


Abbildung 30: 2-Freiheitsgrad-Regelstruktur [28]

In dieser Arbeit wird eine *flachheitsbasierte* Vorsteuerung für das Einspurmodell, zur Trajektorienfolgeregelung, entworfen. Die Inhalte zur Flachheitsmethodik stammen aus den Quellen [29], [28], [30].

5.1. Flachheit

Der Vorteil von flachen Systemen ist, dass der Verlauf des Ausgangsvektors vorgegeben werden kann und daraus der Eingangsgrößenverlauf abgeleitet werden kann. Vor dem Entwurf der Steuerung wird der Begriff von *flachen Systemen* noch genauer erklärt.
Ein System heißt *flach*, auch *differenziell flach*, wenn es möglich ist, die Zustands- und Eingangsgrößen eines Systems, durch den Ausgangsvektor und dessen zeitliche Ableitungen auzudrücken. Solch ein Ausgang wird dann *flacher Ausgang* genannt.
Definition: Flachheit

Ein System

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

sei für $\boldsymbol{x} \in \mathbb{R}^n$ *und* $\boldsymbol{u} \in \mathbb{R}^m$ *mit* $m \leq n$ *definiert und es gelte*

$$rang\left(\frac{\partial \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u})}{\delta \boldsymbol{u}}\right) = m$$

Es heißt flach, wenn ein realer oder fiktiver Ausgangsvektor

$$\boldsymbol{y} = \boldsymbol{h}\left(\boldsymbol{x}, \boldsymbol{u}, \dot{\boldsymbol{u}}, \ldots, \boldsymbol{u}^{(\alpha)}\right) \tag{5.1}$$

mit einem endlichen Wert $\alpha \in \mathbb{N}$ existiert, so dass

1. *der Zustandsvektor $\boldsymbol{x}$ als Funktion von $\boldsymbol{y}$ und einer endlichen Zahl β von Ableitungen $\boldsymbol{y}^{(i)}$ als*

   $$\boldsymbol{x} = \boldsymbol{\Psi}_1\left(\boldsymbol{y}, \dot{\boldsymbol{y}}, \ldots, \boldsymbol{y}^{(\beta)}\right) \tag{5.2}$$

   dargestellt werden kann,

2. *der Eingangsvektor $\boldsymbol{u}$ als Funktion*

   $$\boldsymbol{u} = \boldsymbol{\Psi}_2\left(\boldsymbol{y}, \dot{\boldsymbol{y}}, \ldots, \boldsymbol{y}^{(\beta+1)}\right) \tag{5.3}$$

   darstellbar ist und

3. *für Ein- und Ausgangsvektor*

   $$dim(\boldsymbol{y}) = dim(\boldsymbol{u}) \tag{5.4}$$

   gilt.

Der Ausgangsvektor $\boldsymbol{y}$ heißt flacher Ausgang

In der Definition kommen die Begriffe *realer* Ausgang und *fiktiver* Ausgang vor. Das hat den Grund, dass zum Nachweis der Flachheit nicht erforderlich ist, dass der flache Ausgang ein realer Ausgang ist. Ausgänge die also nicht real existieren, bezeichnet man als *fiktive* Ausgänge.
Durch die Definition wird ersichtlich, warum auch von *differenziell* flachen Systemen gesprochen wird. In den Funktionen (5.2) und (5.3), kommt nur der Ausgang und seine Ableitungen vor, es werden für die Berechnung vom Zustandsvektor $\boldsymbol{x}$ und vom Eingangsvektor $\boldsymbol{x}$ keine Integrale benötigt.
Sehr einfach ist es, wenn der reale Ausgang $\boldsymbol{y}$ ein flacher Ausgang ist, denn dann können der Zustandsvektor und der Eingangsvektor sofort durch den flachen Ausgang und seine Ableitungen parametriert werden. Auch die Steuerung kann somit sehr einfach durch den Eingangsvektor angegeben werden. Wenn der reale Ausgang aber kein flacher Ausgang ist, dann kann dies zu einem erheblichen Aufwand führen. Die Schwierigkeit besteht dann darin, einen flachen Ausgang zu finden, denn es existieren dafür keine allgemeinen Konstruktionsmethoden.

5.2. Flachheit für lineare SISO-Systeme

Die Flachheits-Methodik ist sehr mächtig und wird sowohl für nichtlineare als auch für lineare Systeme eingesetzt. Diese Arbeit beschäftigt sich ausschließlich mit der Flachheit

von linearen SISO-Systemen. Im Falle von linearen Systemen ist der Begriff der Flachheit eng gekoppelt an den Begriff der Steuerbarkeit. Denn für jedes lineare System, das steuerbar ist, kann ein flacher Ausgang berechnet werden, falls der reale Ausgang kein flacher Ausgang ist. Warum dies möglich ist, und in welchen Zusammenhang dies mit der Steuerbarkeit steht, wird in diesem Abschnitt behandelt. Ausgangspunkt für die weiteren Betrachtungen sind lineare SISO-Systeme der Form

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u \\ y &= \boldsymbol{c}^T\boldsymbol{x} \end{aligned}$$

Ein Kriterium zur Überprüfung der Steuerbarkeit, ist die Berechnung der Determinante der Steuerbarkeitsmatrix. Ein System n-ter Ordnung ist steuerbar, wenn die Determinate der Steuerbarkeitsmatrix ungleich Null ist.

$$\boldsymbol{M}_S = \begin{bmatrix} \boldsymbol{b} & \boldsymbol{A}\boldsymbol{b} & \boldsymbol{A}^2\boldsymbol{b} & \cdots & \boldsymbol{A}^{n-1}\boldsymbol{b} \end{bmatrix} \quad \Rightarrow \quad \det\left(\boldsymbol{M}_S\right) \neq 0 \tag{5.5}$$

Lineare SISO-Systeme welche steuerbar sind, können in die Regelungsnormalform transformiert werden.

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_1 & \dots & -a_{n-1} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u \tag{5.6}$$

$$y = \begin{bmatrix} b_0 & b_1 & b_m & 0 & \cdots & 0 \end{bmatrix} \boldsymbol{x}$$

Diese Zustandstransformation ist, wie auch die Flachheit, invariant. Dass heißt, die Systemeigenschaften ändern sich durch die Transformation nicht.
Die Struktur der Regelungsnormalform hat den Vorteil, dass hier sofort die Koeffizienten des Zähler- und Nennerpolynoms der Übertragungsfunktion abgelesen werden können.

$$G\left(s\right) = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0} \tag{5.7}$$

Die Differenz zwischen der Nennerordnung und Zählerordnng der Übertragungsfunktion wird als *relativer Grad* bezeichnet. In diesem Abschnitt wird noch eine weitere Definiton des relativen Grades gezeigt und dass diese Kenngröße in engen Zusammenhang mit dem flachen Ausgang steht. Man kann nun zeigen, dass das System (5.6) für den fiktiven

Ausgang $y_f = x_1$ flach ist:

$$\begin{aligned} x_1 &= y_f \\ x_2 &= \dot{x}_1 = \dot{y}_f \\ x_3 &= \dot{x}_2 = \ddot{y}_f \\ &\vdots \\ x_n &= \dot{x}_{n-1} = {y_f}^{(n-1)} \end{aligned}$$

Der Zustandsvektor lässt sich also durch den flachen Ausgang und dessen Ableitungen darstellen:

$$\boldsymbol{x}^T = \left[y_f \quad \dot{y}_f \quad \ddot{y}_f \quad \cdots \quad y_f^{(n-1)} \right] = \boldsymbol{\Psi_1} \left(y_f, \dot{y}_f, \cdots, y_f^{(n-1)} \right) \tag{5.8}$$

Die Steuerung kann bei Systemen in Regelungsnormfalform sofort abgelesen werden

$$u = a_0 x_1 + a_1 x_2 + \cdots + a_{n-2} x_{n-1} + a_{n-1} x_n + \dot{x}_n$$

Nun wird auch die Steuerung rein durch den flachen Ausgang und seine Ableitungen parametriert

$$\begin{aligned} u &= a_0 y_f + a_1 \dot{y}_f + \cdots + a_{n-2} y_f^{(n-2)} + a_{n-1} y_f^{(n-1)} + y_f^{(n)} \\ &= \boldsymbol{\Psi_2} \left(y_f, \dot{y}_f, \cdots, y_f^{(n)} \right) \end{aligned}$$

Folgende Schlussfolgerungen kann man daraus ziehen: Jedes steuerbare lineare System besitzt die Eigenschaft der Flachheit, da jedes steuerbare lineare System, in Regelungsnormalform transformiert werden kann. Damit ist auch der Zusammenhang zwischen Steuerbarkeit und flachen Systemen erklärt.
Die Regelungsnormalform, beziehungsweise die Transformation in diese, stellt die erste Möglichkeit dar, einen flachen Ausgang für ein lineares SISO-System zu bestimmen.
Es soll aber auch noch eine alternative Berechnungsvorschrift für den flachen Ausgang eines LZI-Systems gezeigt werden.
Es wurde bereits erwähnt, dass es eine Verbindung zwischen dem flachen Ausgang und dem relativen Grad gibt. Allgemein gibt der relative Grad auch die Anzahl an Differentiationen der Ausgangsgröße an, bis zum ersten Mal der Eingang $\boldsymbol{u}$ auftritt. Ein flacher Ausgang besitzt die Eigenschaft, dass dessen relativer Grad, genau der Systemordung n

entspricht. Betrachten wir folgenden flachen Ausgang und seine n Ableitungen:

$$\begin{aligned}
y_f &= \boldsymbol{c}^T\boldsymbol{x} \\
\dot{y}_f &= \boldsymbol{c}^T\dot{\boldsymbol{x}} = \boldsymbol{c}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{b}u \\
\ddot{y}_f &= \boldsymbol{c}^T\boldsymbol{A}\dot{\boldsymbol{x}} + \boldsymbol{c}^T\boldsymbol{b}\dot{u} = \boldsymbol{c}^T\boldsymbol{A}^2\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{A}\boldsymbol{b}u + \boldsymbol{c}^T\boldsymbol{b}\dot{u} \\
&\vdots \\
y_f^{(n-1)} &= \boldsymbol{c}^T\boldsymbol{A}^{n-2}\dot{\boldsymbol{x}}\cdots = \boldsymbol{c}^T\boldsymbol{A}^{n-1}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{A}^{n-2}\boldsymbol{b}u + \boldsymbol{c}^T\boldsymbol{A}^{n-3}\boldsymbol{b}\dot{u} + \cdots + \boldsymbol{c}^T\boldsymbol{b}u^{(n-2)} \\
y_f^{(n)} &= \boldsymbol{c}^T\boldsymbol{A}^{n}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{A}^{n-1}\boldsymbol{b}u + \boldsymbol{c}^T\boldsymbol{A}^{n-2}\boldsymbol{b}\dot{u} + \cdots + \boldsymbol{c}^T\boldsymbol{b}u^{(n-1)}
\end{aligned}$$

Da es sich hier um den flachen Ausgang handelt, erscheint der Eingang u das erste Mal in der Gleichung mit der Ableitung $r = n$, dass heißt alle anderen Gleichungen müssen unabhängig von u und dessen Ableitungen sein. Es gilt:

$$\begin{aligned}
\boldsymbol{c}^T\boldsymbol{b} &= 0 \\
\boldsymbol{c}^T\boldsymbol{A}\boldsymbol{b} &= 0 \\
\boldsymbol{c}^T\boldsymbol{A}^2\boldsymbol{b} &= 0 \\
&\vdots \\
\boldsymbol{c}^T\boldsymbol{A}^{n-2}\boldsymbol{b} &= 0
\end{aligned} \tag{5.9}$$

$$\begin{bmatrix} y_f \\ \dot{y}_f \\ \ddot{y}_f \\ \vdots \\ y_f^{(n-1)} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{c}^T \\ \boldsymbol{c}^T\boldsymbol{A} \\ \boldsymbol{c}^T\boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{c}^T\boldsymbol{A}^{n-1} \end{bmatrix}}_{\boldsymbol{M}_B} \boldsymbol{x} \tag{5.10}$$

Die Matrix $\boldsymbol{M}_B$ ist die Beobachtbarkeitsmatrix. Da der flache Ausgang und seine Ableitungen den Elementen der Beobachtbarkeitsmatrix $\boldsymbol{M}_B$ mit dem flachen Ausgang entsprechen, folgt daraus die Forderung nach Beobachtbarkeit des linearen Systems mit dem flachen Ausgang y_f. Dies ist erfüllt, wenn die Matrix $\boldsymbol{M}_B$ mit dem flachen Ausgang regulär ist.
Damit kann der Zustandsvektor durch die inverse Beobachtbarkeitsmatrix und den fla-

chen Ausgang, samt seinen Ableitungen parametriert werden

$$\boldsymbol{x} = \boldsymbol{M}_B^{-1} \begin{bmatrix} y_f \\ \dot{y}_f \\ \ddot{y}_f \\ \vdots \\ y_f^{(n-1)} \end{bmatrix} = \boldsymbol{\Psi}_1 \left(y_f, \dot{y}_f, \cdots, y_f^{(n-1)} \right) \tag{5.11}$$

Wie bereits erwähnt, gilt für einen flachen Ausgang, dass die Eingangsgröße u in der n-ten Ableitung erscheint:

$$y_f^{(n)} = \boldsymbol{c}^T \boldsymbol{A}^n x + \boldsymbol{c}^T \boldsymbol{A}^{n-1} \boldsymbol{b} u + \boldsymbol{c}^T \boldsymbol{A}^{n-2} \boldsymbol{b} \dot{u} + \cdots + \boldsymbol{c}^T \boldsymbol{b} u^{(n-1)}$$

beziehungsweise

$$y_f^{(n)} = \boldsymbol{c}^T \boldsymbol{A}^n x + \boldsymbol{c}^T \boldsymbol{A}^{n-1} \boldsymbol{b} u \tag{5.12}$$

Der Ausdruck $\boldsymbol{c}^T \boldsymbol{A}^{n-1} \boldsymbol{b}$ muss demnach vom Eingang u abhängen

$$\boldsymbol{c}^T \boldsymbol{A}^{n-1} \boldsymbol{b} = \alpha \quad \text{mit} \quad \alpha \in \mathbb{R} \backslash \{0\} \tag{5.13}$$

Fasst man die Gleichungen (5.9) und (5.13) zusammen erhält man

$$\boldsymbol{c}^T \underbrace{\begin{bmatrix} \boldsymbol{b} & \boldsymbol{A}\boldsymbol{b} & \cdots & \boldsymbol{A}^{n-2}\boldsymbol{b} & \boldsymbol{A}^{n-1}\boldsymbol{b} \end{bmatrix}}_{\boldsymbol{M}_S} = \begin{bmatrix} 0 & 0 & \cdots & 0 & \alpha \end{bmatrix} \tag{5.14}$$

Um Verwechslungen auszuschließen, wird nun $\boldsymbol{c}^T$, aus Gleichung (5.14), durch $\boldsymbol{\lambda}^T$ ersetzt. Alle flachen Ausgänge $y_f = \boldsymbol{\lambda}^T \boldsymbol{x}$ können mit Hilfe der Steuerbarkeitsmatrix $\boldsymbol{M}_S$ gebildet werden durch

$$\boldsymbol{\lambda}^T = \begin{bmatrix} 0 & 0 & \cdots & \alpha \end{bmatrix} \boldsymbol{M}_S^{-1} \tag{5.15}$$

Damit hat man nun eine weitere Möglichkeit, um einen flachen Ausgang zu bestimmen. Es ist üblich, das System in die Flachheitskoordinaten zu transformieren. Dazu werden

neue Koordinaten eingeführt

$$\boldsymbol{z} = \begin{bmatrix} y_f \\ \dot{y}_f \\ \vdots \\ y_f^{(n-1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}^T \\ \boldsymbol{\lambda}^T \boldsymbol{A} \\ \vdots \\ \boldsymbol{\lambda}^T \boldsymbol{A}^{n-1} \end{bmatrix} \boldsymbol{x} = \alpha \underbrace{\begin{bmatrix} [0 & \cdots 0 & 1]\, \boldsymbol{M_S^{-1}} \\ [0 & \cdots 0 & 1]\, \boldsymbol{M_S^{-1}} \boldsymbol{A} \\ & \vdots & \\ [0 & \cdots 0 & 1]\, \boldsymbol{M_S^{-1}} \boldsymbol{A}^{n-1} \end{bmatrix}}_{\boldsymbol{T}} \boldsymbol{x} \tag{5.16}$$

Das lineare zeitinvariante System $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u$ in Flachheitskoordinaten lautet nun

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \dot{y}_f \\ \ddot{y}_f \\ \vdots \\ y_f^{(n-1)} \\ y_f^{(n)} \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ \vdots \\ z_n \\ \boldsymbol{\lambda}^T \boldsymbol{A}^n \boldsymbol{x} + \boldsymbol{\lambda}^T \boldsymbol{A}^{n-1} \boldsymbol{b} u \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ \vdots \\ z_n \\ \boldsymbol{\lambda}^T \boldsymbol{A}^{n-1} \boldsymbol{T}^{-1} \boldsymbol{z} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha \end{bmatrix} u \tag{5.17}$$

mit dem flachen Ausgang $y_f = \boldsymbol{\lambda}^T \boldsymbol{x} = z_1$

5.3. Flachheitsbasierte Vorsteuerung für lineare Systeme

Es werden hier noch einmal die wichtigsten Erkenntnisse aus den letzten Abschnitten zusammengefasst dargestellt. Der reale Ausgang für lineare steuerbare Systeme der Form

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + \boldsymbol{b}u \\ y &= \boldsymbol{c}^T \boldsymbol{x} \end{aligned}$$

ist nur dann ein flacher Ausgang, wenn

$$y = b_0 x_1. \tag{5.18}$$

gilt. Der flache Ausgang hat dabei den relativen Grad n. Sollte der reale Ausgang keinen flachen Ausgang darstellen, dann hat man die Möglichkeit mit Hilfe der Steuerbarkeitsmatrix $\boldsymbol{M}_S$, einen flachen Ausgang zu berechnen (Abschnitt 5.2). Es gilt für alle flachen Ausgänge:

$$y_f = \boldsymbol{\lambda}^T \boldsymbol{x}$$

mit

$$\boldsymbol{\lambda}^T = [0 \quad 0 \quad \cdots \quad \alpha]\, \boldsymbol{M}_S^{-1} \quad \text{und} \quad \alpha \in \mathbb{R} \setminus \{0\}$$

Mittels Regelungsnormalform

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_1 & \dots & -a_{n-1} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} b_0 & b_1 & b_m & 0 & \cdots & 0 \end{bmatrix} \boldsymbol{x}$$

können die Zustandsgleichungen sofort durch den flachen Ausgang und seine Ableitungen parametriert werden

$$\boldsymbol{x} = \boldsymbol{\Psi_1} \left(y_f, \dot{y_f}, \cdots, y_f^{(n-1)} \right) = \begin{bmatrix} y_f \\ \dot{y_f} \\ \ddot{y_f} \\ \vdots \\ y_f^{(n-1)} \end{bmatrix}$$

Aus der letzten Zeile der Regelungsnormalform kann direkt die Steuerung ermittelt werden

$$u = a_0 x_1 + a_1 x_1 + \cdots + a_{n-1} x_n + \dot{x}_n.$$

Parametriert man diese durch den flachen Ausgang und seinen Ableitungen, lautet das Steuergesetz

$$u = a_0 y_f + a_1 \dot{y_f} + \cdots + a_{n-1} y_f^{(n-1)} + y_f^{(n)}. \tag{5.19}$$

Für den Zusammenhang zwischen realem und fiktivem Ausgang gilt

$$y = b_0 x_1 + b_1 x_2 + \cdots + b_m x_{m+1} = b_0 y_f + b_1 \dot{y}_f + \cdots + b_m y_f^{(m)} \tag{5.20}$$

Die Koeffizienten b_0 bis b_m sind genau die Nullstellen der Übertragungsfunktion. Sie beschreiben die interne Dynamik des Systems.

$$G(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} \tag{5.21}$$

Falls der *reale Ausgang ein flacher Ausgang* ist, gilt für die Steuerung Gleichung (5.19).

Ist der reale Ausang kein flacher Ausgang, muss zwischen zwei Fällen unterschieden werden. Im ersten Fall wird ein System mit einem *fiktiven flachen Ausgang* betrachtet, welches *ausschließlich Nullstellen mit negativem Realteil* besitzt. Man erhält den flachen Ausgang durch Lösen der Differentialgleichung (5.20), indem man als Sollwertverlauf $y_{soll}(t)$ den tatsächlichen Verlauf des Ausgangs $y(t)$ vorgibt. Im zweiten Fall wird ein System mit einem *fiktiven flachen Ausgang* betrachtet, welches *Nullstellen mit nicht negativem Realteil* besitzt. Die Differentialgleichung (5.20) ist in diesem Fall instabil. Aus diesem Grund wird für solche Systeme der Sollverlauf für den flachen Ausgang geplant. Die Planung des Sollverlaufes für den Ausgang wird als Arbeitspunktwechsel zwischen zwei Punkten realisiert [30] . Dabei geht man davon aus, dass der Arbeitspunktwechsel von einem Anfangs- zu einem Endpunkt im Intervall $[0, T]$ erfolgt. Anfangs- und Endpunkt werden über ein Polynom mit dem Grad $2n + 1$ verbunden. Die Arbeitspunkte werden dazu in die Flachheitskoordinaten umgerechnet. Es gilt für den Anfangs- und Endpunkt

$$\begin{aligned} z_d(0) &= z_0 = \boldsymbol{\lambda}^T \boldsymbol{x}_0 \\ z_d(T) &= z_T = \boldsymbol{\lambda}^T \boldsymbol{x}_T \end{aligned} \tag{5.22}$$

Für die Ableitungen im Anfangs- und Endpunkt gilt:

$$z_d(0)^i = z_d(T)^i = 0, \quad \forall i = 1, \ldots, n \tag{5.23}$$

Damit ist auch ein stetiger Verlauf der Steuergröße gewährleistet. Setzt man die geplante Solltrajektorie $y_{soll}(t)$ in die Gleichung für die Steuerung ein, erhält man eine stetige Steuertrajektorie. Mit der geplanten Solltrajektorie $z_d(t)$ und der Steuerung, hat man nun alle Komponenten, die man für die Realisierung einer flachheitsbasierten Vorsteuerung benötigt. Die Gleichung für den Ausgang (5.24) wird als Referenzgröße dem Regelkreis zugeführt.

$$y_t = b_0 z_d + b_1 \dot{z}_d + \cdots + b_m z_d^{(m)} \tag{5.24}$$

Der Regelkreis kann zum Beispiel durch eine Ausgangsregelung, wie in Abbildung 31 dargestellt, geschlossen werden.

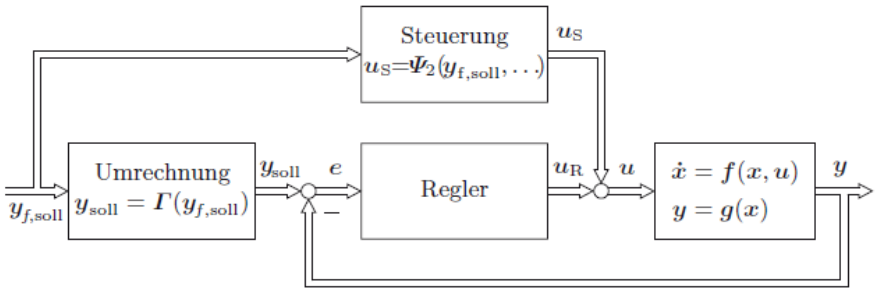
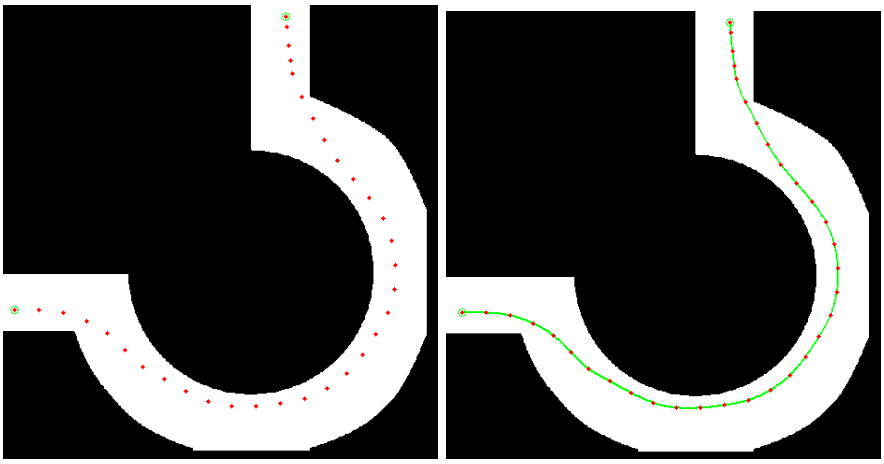


Abbildung 31: Regelkreis mit Trajektorienplanung und Vorsteuerung [29]

5.4. Vorsteuerung für das lineare Einspurmodell

In diesem Abschnitt wird eine Vorsteuerung für das lineare Einspurmodell entworfen (Details zum Einspurmodell sind dem Anhang zu entnehmen). In Abbildung 32(a) ist ein beispielhaftes Ergebnis eines RRT dargestellt. Die Punkte sind die Knoten des minimalen Pfades, welcher mittels RRT ermittelt wurde. Diese Punkte werden für die Vorsteuerung durch einen Spline verbunden (Abbildung 32(b)). Der Grund dafür wird in Kapitel 6 geklärt. Da die Vorsteuerung in Matlab/Simulink entworfen wurde, wurde für den Spline, die von Matlab zur Verfügung gestellte Funktion, verwendet.



(a) Planung mit RRT

(b) Knotenpunkte für die Vorsteuerung mit Spline

Abbildung 32: Beispiel einer Trajektorienplanung mit dem RRT

Die Aufgabe der Vorsteuerung ist nun, der Trajektorie (Abbildung 32(b)) so gut wie möglich zu folgen. Dazu wird zunächst das lineare zeitinvariante Zustandsraummodell des Einspurmodells betrachtet (5.25). Die Zustände sind der Gierwinkel ψ (x_1), die $\dot{\omega}$

(x_2) und der Schwimmwinkel β (x_3). Die Eingangsgröße ist der Lenkwinkel δ. Es wird nun Schritt für Schritt der Entwurf der Steuerung, wie im Abschnitt 5.3 gezeigt, durchgeführt.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{c_{\alpha v} l_v^2 + c_{\alpha h} l_h^2}{\theta v} & -\frac{c_{\alpha v} l_v + c_{\alpha h} l_h}{\theta} \\ 0 & -1 - \frac{c_{\alpha v} l_v + c_{\alpha h} l_h}{m v^2} & -\frac{c_{\alpha v} + c_{\alpha h}}{m v} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{c_{\alpha v} l_v}{\theta} \\ \frac{c_{\alpha v}}{m v} \end{bmatrix} \tag{5.25}$$

In Tabelle 7 sind die Parameter zusammengefasst, welche für das Einspurmodell verwendet wurden:

Fahrzeugparamter	
Abstand Schwerpunkt von der Vorderachse l_v	1.203 $[m]$
Abstand Schwerpunkt von der Hinterachse l_h	1.605 $[m]$
Schräglaufsteifigkeit Vorderachse $c_{\alpha V}$	68220 $[N/-]$
Schräglaufsteifigkeit Hinterachse $c_{\alpha H}$	72000 $[N/-]$
Masse m	1976 $[kg]$
Massenträgheitsmoment des Fahrzeuges um die Hochachse θ	4950 $[kgm^2]$

Tabelle 7: Fahrzeugparameter

Da die Strecke (5.25) nach (5.5), für die Parameter aus Tabelle 7 steuerbar ist, kann ein flacher Ausgang berechnet werden. Aus Abschnitt 5.2 ist bekannt, dass für ein lineares steuerbares System alle flachen Ausgänge mit der Beziehung (5.15) berechnet werden können.

$$\boldsymbol{\lambda}^T = [0 \quad 0 \quad \cdots \quad \alpha]\, \boldsymbol{M}_S^{-1}$$

Es ergibt sich ein flacher Ausgang für das System 5.25 mit den Koeffizienten $c_{f1}, c_{f3} > 0$ und $c_{f2} < 0$ zu

$$\boldsymbol{y}_f \quad = \quad c_{f3} x_1 + c_{f2} x_2 + c_{f1} x_3$$

Im nächsten Schritt wird die Stabilität der internen Dynamik überprüft. Dazu betrachtet man die die Übertragungsfunktion der Strecke mit $b_1, b_0 > 0$ und $a_1, a_2, a_3 > 0$

$$G(s) = \frac{b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s}$$

Die Übertragungsfunktion weist nicht negative Nullstellen auf. Es muss also ein Sollver-

lauf für den flachen Ausgang entworfen werden. Man hat hier mehrere Arbeitspunktwechsel hintereinander, daher wiederholt sich dieser Vorgang, sobald ein Arbeitspunkt erreicht wurde. Wichtig dabei ist, dass die Übergänge zwischen den Punkten stetig verlaufen. Der flache Ausgang bleibt aber für den gesamten Pfad derselbe, da er nicht von den Arbeitspunkten, sondern lediglich vom System selbst abhängig ist. Da sich der flache Ausgang aus allen Zustandsgrößen zusammensetzt, benötigt man die Kenntnis der Kenngrößen in allen Arbeitspunkten (Abbildung 33). Für diese Kenngrößen gelten folgende Beziehungen:
Für den Gierwinkel ψ gilt:

$$\psi = \arctan \frac{\Delta y}{\Delta x}. \tag{5.26}$$

Die Gierrate ω ist die Änderung des Winkels nach der Zeit:

$$\omega = \frac{\Delta \omega}{\Delta t}. \tag{5.27}$$

Der Schwimmwinkel β ist durch folgende Formeln gegeben:

$$\beta = \frac{l_h}{R} - \frac{m a_y}{c_{\alpha H}} \frac{l_v}{l_v + l_h}. \tag{5.28}$$

Die Abstände vom Schwerpunkt zur Vorderachse l_v beziehungsweise zur Hinterachse l_v und die Schräglaufsteifigkeit der Hinterachse $c_{\alpha H}$ sind bekannte Kenngrößen. Der Krümmungsradius R und die Querbeschleunigung a_y müssen noch ermittelt werden.
Der Krümmungsradius R kann aus der Geschwindigkeit v und der Gierrate ω berechnet werden:

$$R = \frac{v}{\omega}. \tag{5.29}$$

Die Querbeschleunigung ergibt sich aus dem Krümmungsradius R und der Geschwindigkeit v:

$$a_y = \frac{v^2}{R}. \tag{5.30}$$

Die Arbeitpsunkte werden über ein Polynom 7ten Grades $(2n+1)$ verbunden.

$$z_d(t) = \alpha_7 t^7 + \alpha_6 t^6 + \alpha_5 t^5 + \alpha_4 t^4 + \alpha_3 t^3 + \alpha_2 t^2 + \alpha_1 t + \alpha_0 \tag{5.31}$$

Anfangs- und Endpunkt müssen in die Flachheitskoordinaten transformiert werden:

$$z_d(0) = z_0 = \boldsymbol{\lambda}^T \boldsymbol{x}_0 = [c_{f3} \quad c_{f2} \quad c_{f1}] \boldsymbol{x}_0$$

$$z_d(T) = z_T = \boldsymbol{\lambda}^T \boldsymbol{x}_T = [c_{f3} \quad c_{f2} \quad c_{f1}] \boldsymbol{x}_T$$

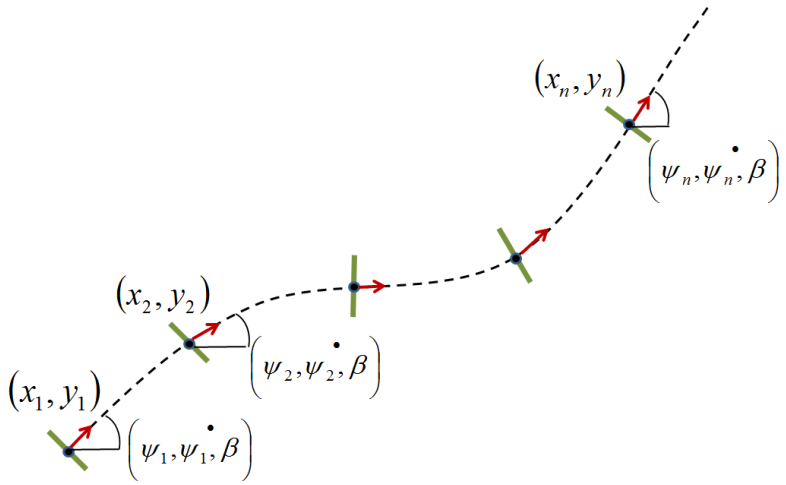


Abbildung 33: Zustände

Für die Ableitungen gilt nach (5.23):

$$z_d(0)^i = z_d(T)^i = 0 \quad \forall \quad i = 1 - 3 \tag{5.32}$$

Da man hier aber eine Reihe von Arbeitspunktwechseln hat, wird die erste Ableitung

$$z_d(0)^i = \dot{y}_f(T_1)$$
$$z_d(T)^i = \dot{y}_f(T_2)$$

gesetzt.

Es ergibt sich somit folgendes Gleichungssystem:

$$\begin{bmatrix} t^7 & t^6 & t^5 & t^4 & t^3 & t^2 & t & 1 \\ 7t^6 & 6t^5 & 5t^4 & 4t^3 & 3t^2 & 2t & 1 & 0 \\ 42t^5 & 30t^4 & 20t^3 & 12t^2 & 6t & 2 & 0 & 0 \\ 210t^4 & 120t^3 & 60t^2 & 24t & 6 & 0 & 0 & 0 \\ t^7 & t^6 & t^5 & t^4 & t^3 & t^2 & t & 1 \\ 7t^6 & 6t^5 & 5t^4 & 4t^3 & 3t^2 & 2t & 1 & 0 \\ 42t^5 & 30t^4 & 20t^3 & 12t^2 & 6t & 2 & 0 & 0 \\ 210t^4 & 120t^3 & 60t^2 & 24t & 6 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_7 \\ \alpha_6 \\ \alpha_5 \\ \alpha_4 \\ \alpha_3 \\ \alpha_2 \\ \alpha_1 \\ \alpha_0 \end{bmatrix} = \begin{bmatrix} z_d(0) \\ \dot{z}_d(0) \\ \ddot{z}_d(0) \\ \dddot{z}_d(0) \\ z_d(T) \\ \dot{z}_d(T) \\ \ddot{z}_d(T) \\ \dddot{z}_d(T) \end{bmatrix} \tag{5.33}$$

Durch Lösung dieses Gleichungssystems, hat man alle Unbekannten für das Polynom z_d.

Dieses Polynom hat Gültigkeit im Intervall $[T_1, T_2]$. Für alle weiteren Punkte, muss dieses Gleichungssystem erneut zwischen zwei Punkten berechnet werden. Da man davon ausgehen kann, dass es hier zu numerischen Ungenauigkeiten kommt, wird das Einspurmodell die Punkte nicht immer exakt erreichen. Aus diesem Grund, werden die Punkte, die das Fahrzeug am Ende einer Trajektorienplanung erreicht hat, als Anfangspunkt, für die nächste Planung zurückgegeben. Das diese Zustände wieder an den Eingang der Trajektorienplanung zurückgeführt werden, ist auch im Koppelplan, in Abbildung 34, zu sehen.

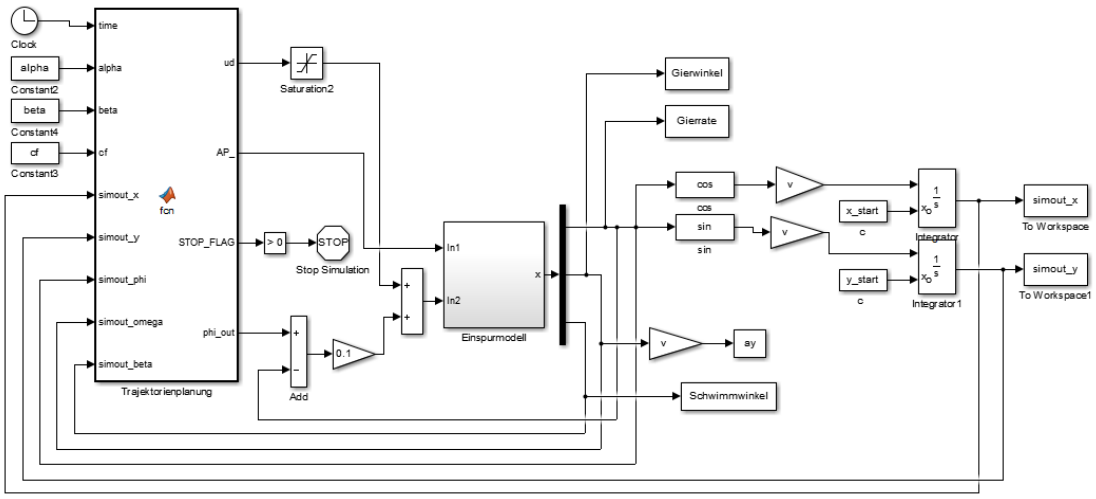


Abbildung 34: Simulink-Koppelplan

6. Ergebnisse und Diskussion

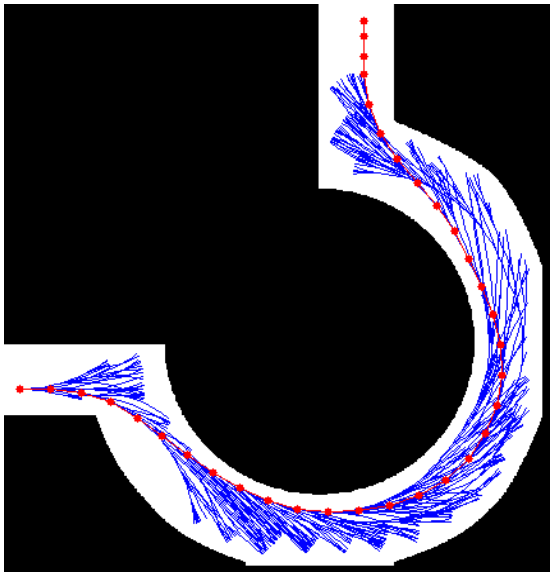
In diesem Kapitel werden die Ergebnisse des Trajektorienplaners mittels flachheitsbasierte Vorsteuerung evaluiert. Besonderes Augenmerk wird dabei auf die Querdynamik gelegt. Bei der Trajektorienplanung wurde durch sorgfältige Wahl der Bewegungsprimitive darauf geachtet, keine zu hohen Querbeschleunigungen in die Planung miteinfließen zu lassen. Außerdem wird anhand der Vorsteuerung auch untersucht, wie gut die geplante Trajektorie nachgefahren werden kann und welche Abweichungen hier zustande kommen.

6.1. Kreisverkehr

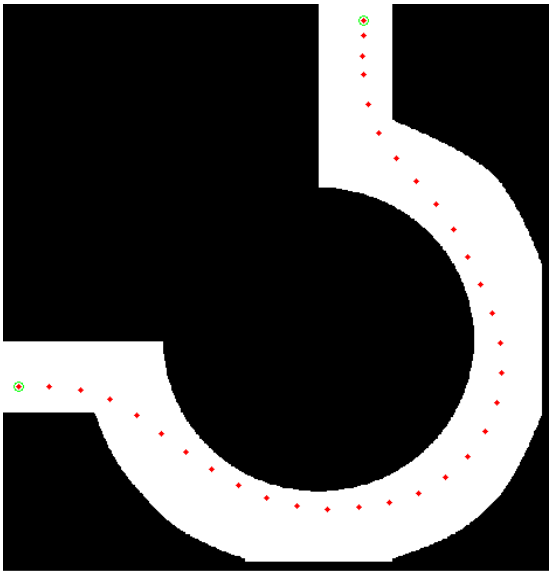
Die Planung der Trajektorie durch den Kreisverkehr erfolgte mit einer Geschwindigkeit von 20 km/h. Die für die Planung berechneten Bewegungsprimitive basieren auf folgenden Querbeschleunigungen:

- $a_{y1} = 0\,\mathrm{m}s^2$
- $a_{y2} = 1\,\mathrm{m}s^2$
- $a_{y3} = 2\,\mathrm{m}s^2$

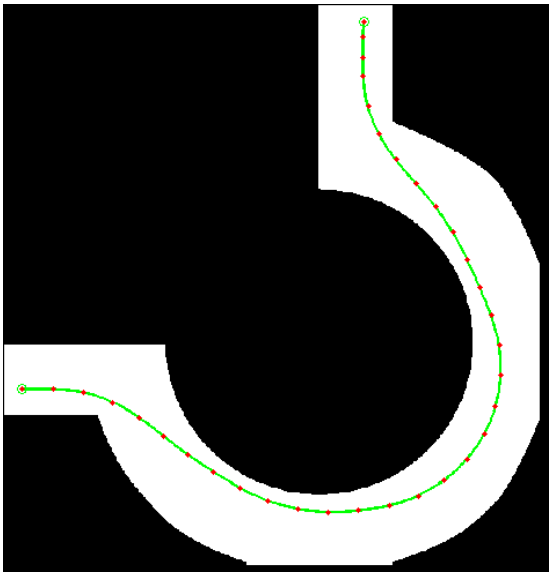
Aus diesen drei Querbeschleunigungen kann man fünf Bewegungsprimitve erzeugen, siehe dazu Abbildung 13. Weiters wurde ein Abstand von 1 m zu Hindernissen in der Planung berücksichtigt. Abbildung 35(a) zeigt ein Ergebnis der Planung mittels RRT. Die Integrationszeit Δt wurde mit $0.6sec$ festgelegt, sodass zwei Knotenpunkte einen maximalen Abstand von $3.3m$ zueinander haben. Dieser Abstand ist für die Vorsteuerung allerdings zu gering, weshalb ein Spline durch die Punkte gelegt wird (Abbildung 35(c)). Dadurch kann die Vorsteuerung Punkte auf diesem Spline anfahren, die einen größeren Abstand zueinander haben. Der kürzeste gefundene Pfad wurde rot eingezeichnet. In Abbildung 35(b) sind nur mehr die Knotenpunkte dargestellt, welche vom Algorithmus zurückgeliefert und anschließend an die Vorsteuerung übergeben werden.



(a) Planung mit RRT



(b) Resultat des RRT



(c) Knotenpunkte für die Vorsteuerung mit Spline

Abbildung 35: RRT Ergebnisse - Kreisverkehr

Hinsichtlich Komfortbewertung ist nun interessant, welche Querbeschleunigungen während der Trajektorienfolge entstehen. In Abbildung 36 ist der Verlauf der Querbeschleunigung und des Lenkwinkels dargestellt. Vergleicht man die maximalen Werte für die Querbeschleuniungen, mit den Querbeschleunigungen aus dem Diagramm 12 von [6], liegen diese innerhalb eines tolerierbaren Bereiches. Trotzdem ist auffällig, dass die Querbeschleunigungen an manchen Stellen jene Querbeschleunigungen, mit welchen die Bewe-

gungsprimitive in der Trajektorienplanung berechnet wurden, leicht übersteigen. Zusätzlich ist in Abbildung 36 auch der Lenkwinkelverlauf dargestellt. Die Schwankungen im Lenkwinkel lassen sich dadurch begründen, dass die Vorsteuerung versucht der Trajektorie so genau wie möglich zu folgen. Diese oszillierende Bewegung spiegelt sich allerdings auch in der Querbeschleunigung wieder und beinträchtigt den Fahrkomfort der Trajektorie.

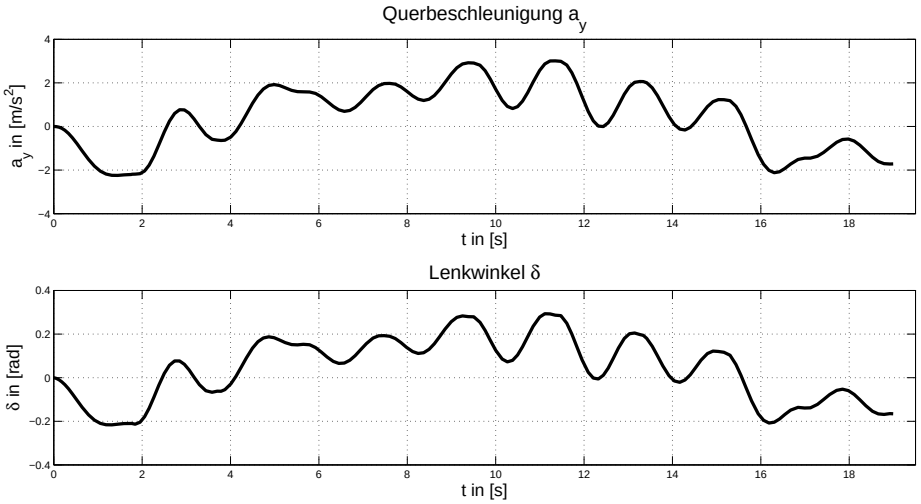


Abbildung 36: Querbeschleunigung und Lenkwinkel

Abbildung 37 präsentiert den vorgegebenen Spline und das Ergebnis der Vorsteuerung. Man sieht, dass nur geringe Abweichungen zwischen den beiden Trajektorien liegen. Die Aufgabe der Trajektorienfolge wird also von der Vorsteuerung sehr gut erfüllt.

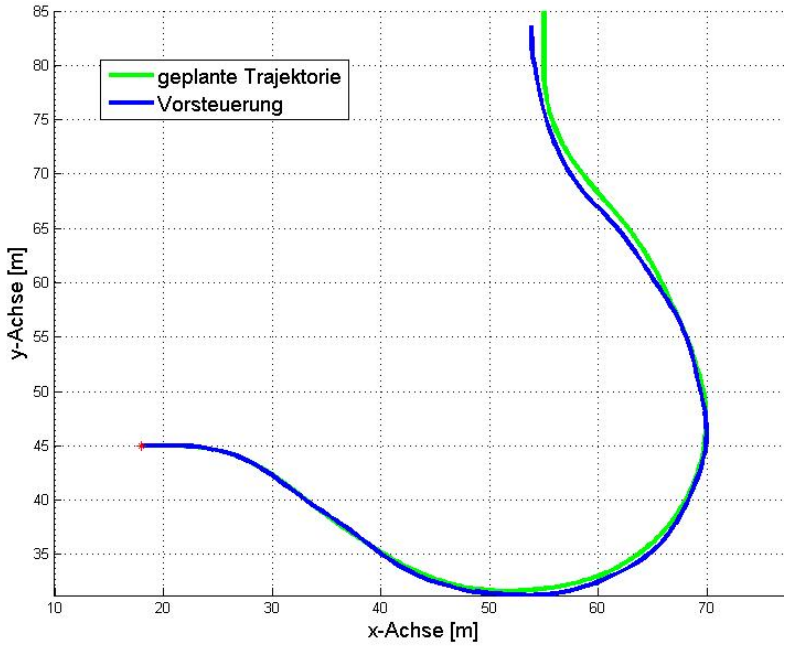


Abbildung 37: Ergebnis 1 der Vorsteuerung - Kreisverkehr

Wie bereits aus vorigen Kapiteln bekannt ist, werden bei der Trajektorienplanung nur Knoten gespeichert. Pfadsegmente zwischen Knoten werden nicht gespeichert, da dies zu viel Speicher beanspruchen würde. Die Knoten die man als Ergebnis vom RRT erhält, werden für die Vorsteuerung durch einen Spline verbunden. Der Grund für die erhöhten Querbeschleunigungen liegt genau in diesem Spline, da dieser höhere Krümmungen aufweist.
Es folgt nun noch ein Ergebnis einer Trajektorienplanung im Kreisverkehr. Dabei wurde die Geschwindigkeit nicht verändert, aber die Querbeschleunigungen für die Bewegungsprimitive wurden abgeändert. Es wird sich zeigen, dass eine Erhöhung der Querbeschleunigung, auch eine Erhöhung derselben durch die Vorsteuerung hervorgerufen wird.

- $a_{y1} = 0\,\mathrm{m}s^2$
- $a_{y2} = 2\,\mathrm{m}s^2$
- $a_{y3} = 3\,\mathrm{m}s^2$

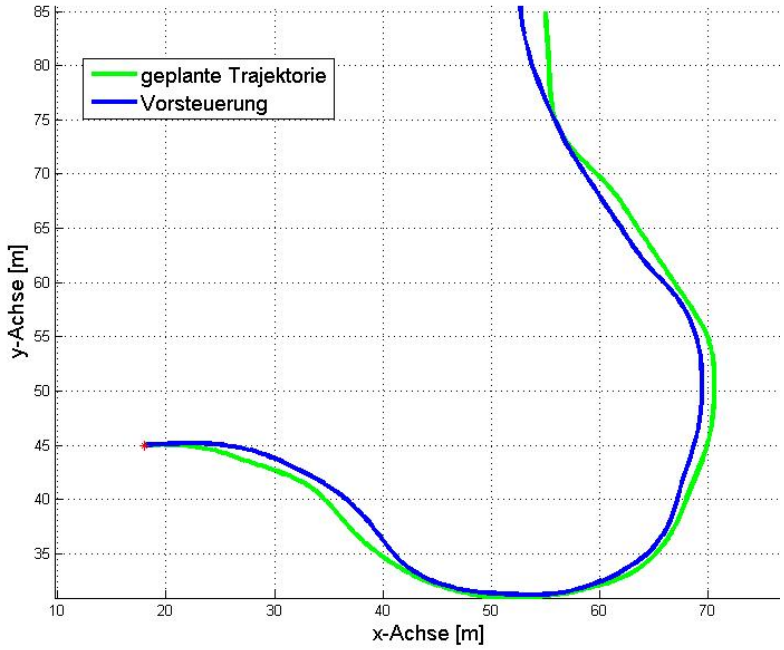


Abbildung 38: Ergebnis 2 der Vorsteuerung - Kreisverkehr

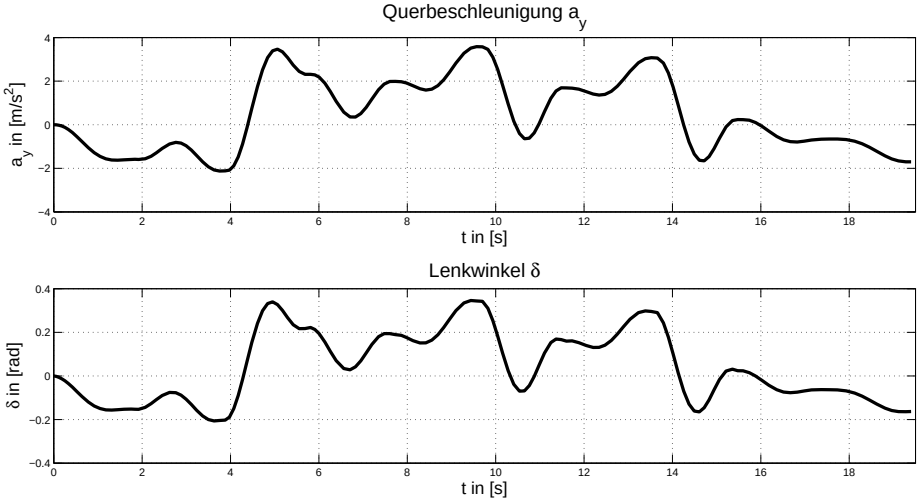


Abbildung 39: Querbeschleunigung und Lenkwinkel

Obwohl diese Planung mit größeren Querbeschleunigungen erfolgt, bleiben die resultierenden Querbeschleunigungen noch in einem vernünftigen Bereich. Man kann also davon ausgehen, dass Beschränkungen für die Bewegungsprimitive im Planungsalgorith-

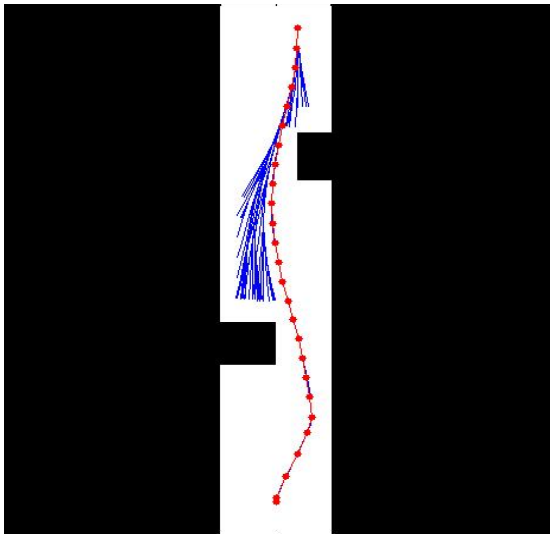
mus Sinn machen. Die Forderung, dass die Querbeschleunigungen innerhalb der Toleranzgrenzen, siehe Abbildung 12, bleiben, ist erfüllt. Aber auch hier sind unerwünschte Lenkwinkelschwankungen zu sehen, die den Fahrkomfort beeinträchtigen. Auffällig in Abbildung 38 ist der letzte Teil der Trajektorie. Das letzte Teilstück wird in dieser Arbeit durch eine Bezier-Kurve geplant. Die Vorsteuerung verursacht in diesem Bereich keine unangenehmen Querbeschleunigungen, aber man sieht, dass die Vorsteuerung dem letzten Teil schlecht folgen kann.

6.2. Fahrbahn mit Hindernissen

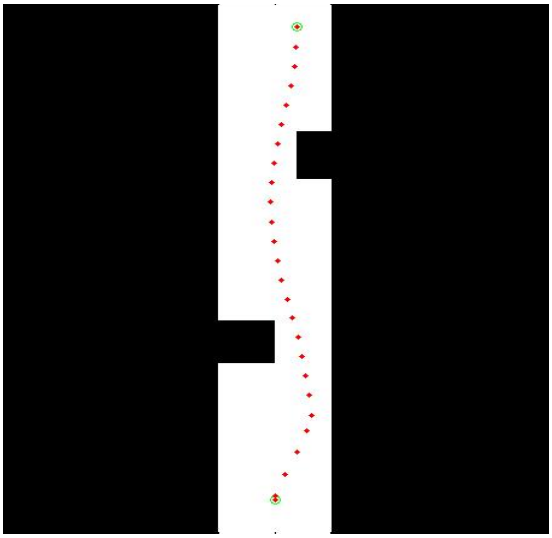
Für die Planung der Trajektorie auf der Fahrbahn mit Hindernissen, wurde eine konstante Geschwindigkeit von 20 km/h gewählt. Die Bewegungsprimitive wurden aus folgenden Querbeschleunigungen berechnet:

- $a_{y1} = 0\,\mathrm{m}s^2$
- $a_{y2} = 1\,\mathrm{m}s^2$
- $a_{y3} = 2\,\mathrm{m}s^2$

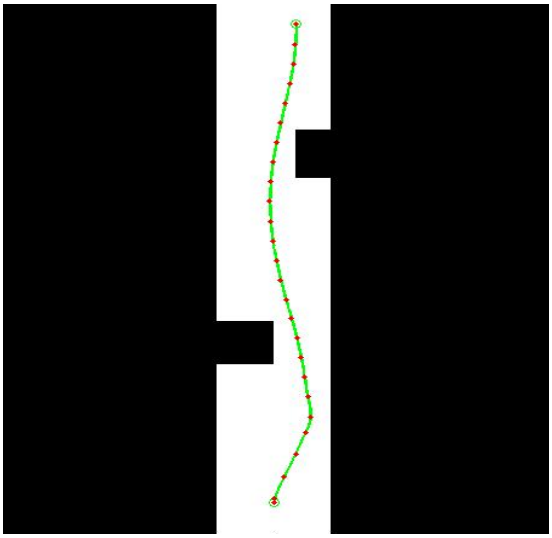
Während der Planung wurde ein Abstand von 1 m zu Hindernissen berücksichtigt. In Abbildung 40(a) ist das Resultat der Planung mittels RRT abgebildet. Abbildung 40(b) zeigt die Knotenpunkte, welche aus der Planung resultieren und an die Vorsteuerung übergeben werden. Die letzte Abbildung (Abbildung 40(c)) zeigt den Spline der durch die Knotenpunkte führt.



(a) Planung mit RRT



(b) Resultat des RRT



(c) Knotenpunkte für die Vorsteuerung mit Spline

Abbildung 40: RRT Ergebnisse - Fahrbahn mit Hindernissen

In Abbildung 41 ist der Verlauf der Querbeschleunigung dargestellt. Hier weist das letzte Stück der Trajektorie, welches mittels Bezier-Kurve geplant wurde, sehr hohe Querbeschleunigungen auf. Die Querbeschleunigungen im restlichen Bereich sind zufriedenstellend und übersteigen auch nicht jene Werte mit welchen geplant wurde. Das zweite Diagramm zeigt den Verlauf des Lenkwinkels. Auch hier sind Lenkwinkelschwankungen erkennbar.

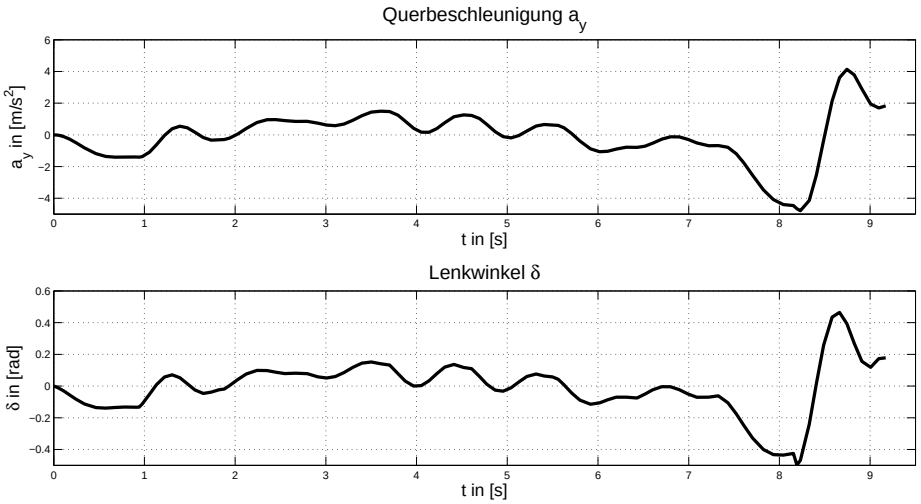


Abbildung 41: Querbeschleunigung und Lenkwinkel

Die Bezier-Kurve erzeugt nicht nur zu hohe Querbeschleunigungen, es ist auch für die Vorsteuerung schwierig, der Bezier-Kurve zu folgen (Abbildung 42). Der Bereich vor der Bezier-Kurve, kann von der Vorsteuerung sehr gut verfolgt werden.

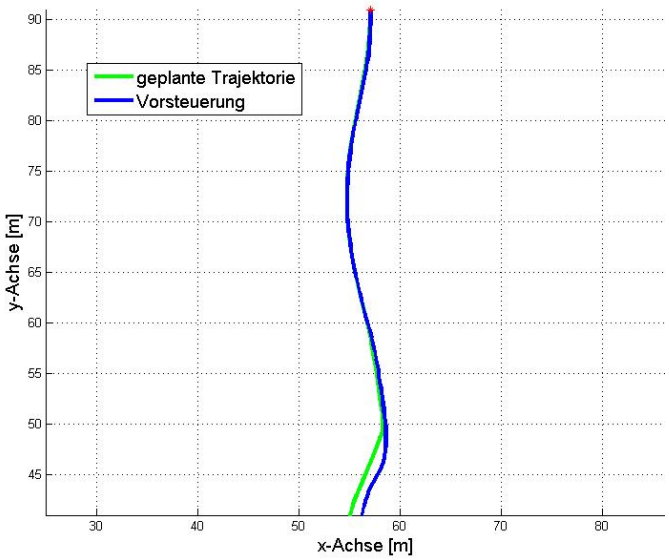


Abbildung 42: Ergebnis 1 der Vorsteuerung - Fahrbahn mit Hindernissen

Es folgt nun ein weiteres Resultat einer Planung auf der Fahrbahn mit Hindernissen. Die zweite Planung unterscheidet sich zu ersten Planung, nur durch eine Abänderung der

Querbeschleunigungen.

- $a_{y1} = 0\,\mathrm{m}s^2$
- $a_{y2} = 2\,\mathrm{m}s^2$
- $a_{y3} = 3\,\mathrm{m}s^2$

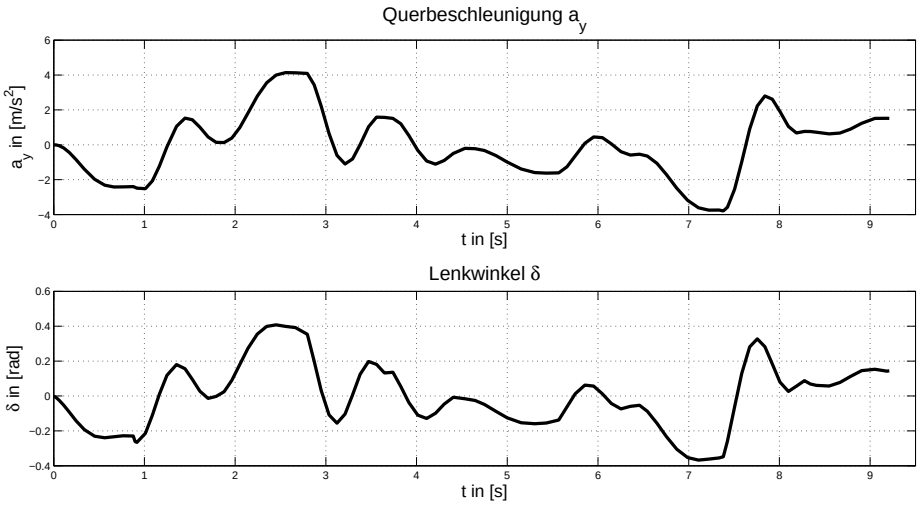


Abbildung 43: Querbeschleunigung und Lenkwinkel

Es war zu erwarten, dass eine Erhöhung der Querbeschleunigung bei der Planung der Trajektorie, sich auch auf das Ergebnis auswirkt. Vergleicht man die resultierenden Querbeschleunigungen mit 12, dann sind die Querbeschleunigungen des Pfades an manchen Stellen zu hoch. Das Ergebnis der Vorsteuerung ist wieder sehr positiv, hat aber auch hier den Nachteil, dass der Lenkwinkel oszilliert.

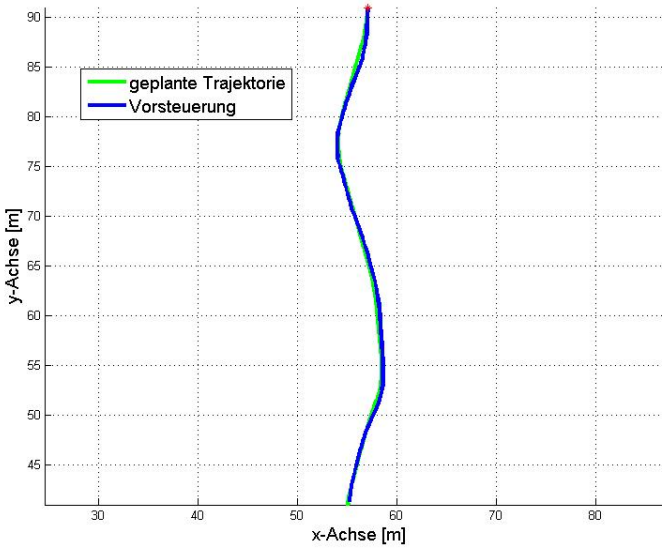


Abbildung 44: Ergebnis 2 der Vorsteuerung - Fahrbahn mit Hindernissen

7. Zusammenfassung und Ausblick

Zusammenfassend kann gesagt werden, dass die Ergebnisse aus Kapitel 6 sehr zufriedenstellend sind. Der Algorithmus zur Trajektorienplanung ist in der Lage, um Hindernisse herum zu planen und einen gegebenen Sicherheitsabstand zu Hindernissen einzuhalten. Die Herausforderungen, eine Trajektorie durch einen Kreisverkehr zu planen, stellt für diesen Algorithmus ebfenfalls kein Problem dar. Eine interessante Erweiterung wäre den RRT in dynamischer Umgebung zu testen. Die Beschränkungen durch die Querbeschleunigungen zur Berechnung der Bewegungsprimitive, sind eine sinnvolle Maßnahme, um bereits während der Planung ein gewisses Maß an Komfort zu berücksichtigen. Dies zeigt auch die Evaluierung der Trajektorien durch die Vorsteuerung. Vergleicht man die resultierenden Querbeschleunigungen mit dem Diagramm 12 von [6], dann bleiben diese im tolerierbaren Bereich. Auch die Trajektorienfolge an sich liefert zufriedenstellende Ergebnisse. Die Trajektorien können von der Vorsteuerung sehr gut verfolgt werden. Negativ auf den Fahrkomfort wirken sich die oszillierenden Lenkwinkel aus. Es besteht hier der Bedarf einer Verbesserung. Der Lenkwinkel wurde in dieser Arbeit weder im RRT noch in der Vorsteuerung berücksichtigt. Verbesserungsbedarf hat auch die Planung mittels Bezier-Kurve. Lediglich eine Bezier-Kurve während der Planung zu verwenden ist sehr einschränkend. Eine Möglichkeit wäre es, ein Kriterium zu entwerfen, nach welchem man eine vernünftige Bezier-Kurve auswählt. Eine andere Option wäre, sich eine gänzlich andere Methode zu überlegen wie man Ausrichtungen während der Pfadplanung berücksichtigen kann.

A. Einspurmodell

Das in dieser Arbeit verwendete Fahrzeugmodell zur Trajektorienfolgeregelung, ist das *lineare Einspurmodell* . Es ist das, in der Praxis vielfach eingesetzte Modell zur Untersuchung der Querdynamik eines Fahrzeuges. Bevor auf die Bewegungsgleichungen für das lineare Einspurmodell eingegangen wird, folgen noch ein paar Begriffe aus der Fahrdynamik. Die Inhalte zu diesem Kapitel, stammen aus dem Vorlesungsunterlagen von [31].

A.1. Fahrzeugbewegungen

Die allgemeine Bewegung eines Fahrzeuges wird durch sechs Freiheitsgrade beschrieben [31]. Zu den translatorischen Bewegungen gehören das *Fahren*, *Schieben* und *Heben*. *Fahren* beschreibt die Bewegung entlang der x-Achse (Längsbewegung), *Schieben* die Bewegung entlang der y-Achse (Querbewegung) und *Heben* die Bewegung entlang der z-Achse (Hubbewegung).
Zu den Rotationsbewegungen zählen *Gieren*, *Nicken* und *Wanken*. *Gieren* bezeichnet die Drehung um die z-Achse, *Nicken* um die y-Achse und *Wanken* um die x-Achse.

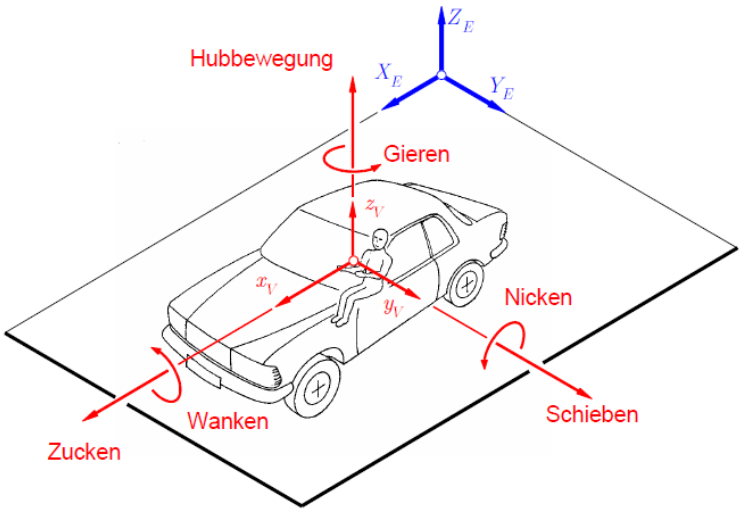


Abbildung 45: allgemeine Bewegungen eines Fahrzeuges

A.2. Fahrzeugfestes Koordinatensystem

Das fahrzeugfeste Koordinatensystem ist ein rechtwinkeliges Koordinatensystem, bei welchem der Ursprung O_V meist im Fahrzeugschwerpunkt gewählt wird. Die x-Achse X_V zeigt in Längsrichtung vom Schwerpunkt aus nach vorne, die y-Achse Y_V zeigt im rechten Winkel zur x-Achse in Querrichtung und die z-Achse Z_V zeigt nacht oben.

A.3. Bewegungsgleichungen des lineares Einspurmodells

Das Einspurmodell behält seine Gültigkeit für Fahrten mit Querbeschleunigungen a_y bis 4 $\frac{\mathrm{m}}{\mathrm{s}^2}$. Die Querdynamik wird im wesentlichen durch zwei Freiheitgrade beschrieben. Dies ist auf der einen Seite der Schwimmwinkel β, welcher ein Maß für die Bewegung in Querrichtung darstellt. Auf der anderen Seite die Gierrate $\dot{\psi}$, die ein Maß für die Drehung um die Hochachse ist.

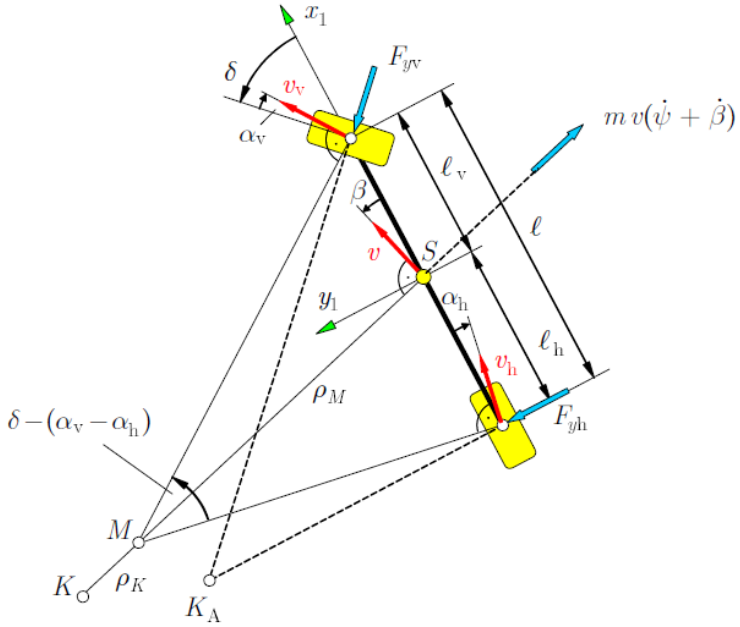


Abbildung 46: Lineares Einspurmodell

Beim Einspurmodell wird das Fahrzeug auf eine Spur reduziert, indem die Räder achsweise zu je einem Rad zusammengefasst werden. Der Schwerpunkt wird auf Fahrbahnhöhe gelegt. Die Lage des Fahrzeuges ist durch die x/y- Position sowie dem Gierwinkel eindeutig festgelegt. Weiters geht man davon aus, dass es keine Beschleunigung in Längsrichtung gibt. Mit anderen Worten, die Geschwindigkeit ist konstant. Da die Fahrzeugmasse im Schwerpunkt zusammengefasst ist, gilt für die Fahrzeuggeschwindigkeit des Schwerpunktes im fahrzeugfesten Koordinatensystem:

$$\vec{v} := \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v cos\beta \\ v sin\beta \\ 0 \end{bmatrix} \tag{A.1}$$

Für die Beschleunigung des Schwerpunktes, im fahrzeugfesten Koordinatensystem mit

der Winkelgeschwindigkeit ω gilt:

$$\vec{a} := \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \frac{dv}{dt} + \omega \times v = \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{\omega_z} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_z \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} -v\left(\dot{\psi} + \dot{\beta}\right) sin\beta \\ v\left(\dot{\psi} + \dot{\beta}\right) cos\beta \\ 0 \end{bmatrix} \tag{A.2}$$

Mit der Transformationsmatrix T, der Winkelgeschwindigkeit ω und dem Geschwindigkeitsvektor v gilt folgender Zusammenhang zum Lagevektor:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \underbrace{\begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{T} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{A.3}$$

Als Reifenmodell wird in dieser Arbeit das lineare Reifenmodell verwendet. Dieses behält seine Gültigkeit bei kleinen Schräglaufwinkeln α, dass heisst, dass die Querkraft durch eine Gerade approximiert wird.

$$F_{yv} = c_{\alpha v}\alpha_v \tag{A.4}$$

$$F_{yh} = c_{\alpha h}\alpha_h \tag{A.5}$$

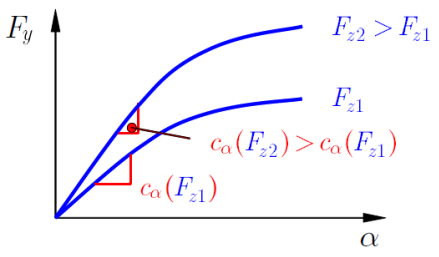


Abbildung 47: linearisierte Seitenkraftkennlinie

Als Radlast wird die Kraft senkrecht zur Fahrbahn, die auf das Rad einwirkt bezeichnet. Auf Grund konstanter Radlasten an der Vorder- und Hinterachse können Hub- und Nickbewegung vernachlässigt werden.

$$F_{zv} = mg\frac{l_h}{l} \tag{A.6}$$

$$F_{zh} = mg\frac{l_v}{l} \tag{A.7}$$

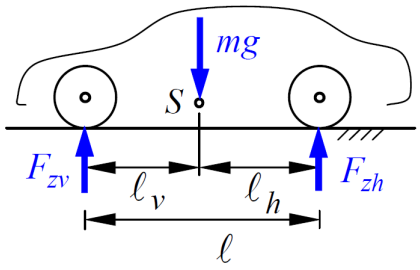


Abbildung 48: Radlast

Die Bewegungsgleichungen lassen sich mit Hilfe des Impuls- und Drallsatzes aufstellen:
Impulssatz in Fahrzeugquerrichtung (y-Achse)

$$ma_y = F_{yv}cos\delta + F_{yh} \tag{A.8}$$

$$mv\left(\dot{\psi} + \dot{\beta}\right)cos\beta = c_{\alpha v}\alpha_v cos\delta + c_{\alpha h}\alpha_h \tag{A.9}$$

Drallsatz um die Fahrzeughochachse (z-Achse)

$$\Theta\ddot{\psi} = F_{yv}cos\delta l_v - F_{yh}l_h \tag{A.10}$$

$$\Theta\ddot{\psi} = c_{\alpha v}\alpha_v cos\delta l_v - c_{\alpha h}\alpha_h l_h \tag{A.11}$$

Für kleine Lenkwinkel und Giergeschwindigkeiten gilt folgende Näherung der Schräglaufwinkel am Vorder- bzw. Hinterrad:

$$\alpha_v = \delta - \beta - l_v\frac{\dot{\psi}}{v} \tag{A.12}$$

$$\alpha_h = -\beta + l_h\frac{\dot{\psi}}{v} \tag{A.13}$$

Linearisierung der Bewegungsgleichungen, unter der Annahme kleiner Winkel und Winkelgeschwindigkeiten führt zu:

$$mv\left(\dot{\psi}+\dot{\beta}\right) = c_{\alpha v}\alpha_v + c_{\alpha h}\alpha_h \tag{A.14}$$

$$\theta\ddot{\psi} = c_{\alpha v}\alpha_v l_v - c_{\alpha h}\alpha_h l_h \tag{A.15}$$

Setzt man in diese Gleichungen noch die Schräglaufwinkel erhält man:

$$mv\dot{\beta} + \left(mv^2 + c_{\alpha v}l_v - c_{\alpha h}l_h\right)\frac{\dot{\psi}}{v} + \left(c_{\alpha v}+c_{\alpha h}\right)\beta = c_{\alpha v}\delta \tag{A.16}$$

$$\theta\ddot{\psi} + \left(c_{\alpha v}l_v^2 + c_{\alpha h}l_h^2\right)\frac{\dot{\psi}}{v} + \left(c_{\alpha v}l_v - c_{\alpha h}l_h\right)\beta = c_{\alpha v}l_v\delta \tag{A.17}$$

Das lineare zeitinvariante System im Zustandsraummodell lautet:

$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & -\frac{c_{\alpha v}l_v^2+c_{\alpha h}l_h^2}{\theta v} & -\frac{c_{\alpha v}l_v+c_{\alpha h}l_h}{\theta}\\ 0 & -1-\frac{c_{\alpha v}l_v+c_{\alpha h}l_h}{mv^2} & -\frac{c_{\alpha v}+c_{\alpha h}}{mv}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ \frac{c_{\alpha v}l_v}{\theta}\\ \frac{c_{\alpha v}}{mv}\end{bmatrix} \tag{A.18}$$

Die Zustandsgrößen sind der Gierwinkel $x_1 = \psi$, die Gierrate $x_2 = \dot{\psi}$ und der Schwimmwinkel $x_3 = \beta$. Die Eingangsgröße stellt der Lenkwinkel δ dar.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] R. Neuhold, M. Rudigier, A. Kerschbaumer und M. Fellendorf, "Analysing the capacity of autonomous transportation systems with microscopic traffic flow simulation", *International Scientific Conference on Mobility and Transport*, 2015, pages 1-12.

[2] *Geschichte des autonomen fahrens*, Website, 2012, Online unter `http://www.autonomes-fahren.de/geschichte-des-autonomen-fahrens/`; abgerufen am 17.03.2016.

[3] M. Goebel, "Eine realzeitfähige Architektur zur Integration kognitiver Funktionen", Dissertation, Technische Universität München, 2009.

[4] C. Urmson, C. Baker, J. Dolan, P. Rybski, B. Salesky, W. Whittaker, D. Ferguson und M. Darms, "Autonomous driving in traffic: Boss and the urban challenge", *AI MAGAZINE*, Bd. 30, Nr. 2, 2015, Summer Issue.

[5] *Meilensteine der autonomen Fahrzeuge*, Website, Online unter `http://www.greengear.de/autonomes-fahren-autonome-fahrzeuge-autos-meilensteine/`; abgerufen am 17.03.2016, März 2015.

[6] M. Wegscheider, "Modellbasierte Komfortbewertung von Fahrerassistenzsystemen", Dissertation, Technische Universität Graz, 2009.

[7] M. Oubbati, *Einführung in die Robotik*, Universität Ulm, Vorlesungsskript, 2009/2010, online unter `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.130/Mitarbeiter/oubbati/RobotikWS1113/OubbatiSkript.pdf`; abgerufen am 17.03.2016.

[8] C. T. Eiserloh, "Schnell wachsende Suchbäume zur Pfadplanung für allgemeine Gliederfahrzeuge", Masterarbeit, Universität Koblenz Landau, 2013.

[9] J. Schröder, "Adaptive Verhaltensentscheidung und Bahnplanung für kognitive Automobile", Dissertation, Universität Karlsruhe, 2009.

[10] A. Krause und R. Stange, *Kürzeste Wege*, Website, 2014, Online unter `http://programmingwiki.de/AuK/K%C3%BCrzeste_Wege_WS13-14`; abgerufen am 17.03.2016.

[11] F. S. Winkler, *A*-Algorithmus*, Website, Online unter `http://me-lrt.de/tiefensuche-breitensuche-dijkstra-a-star`; abgerufen am 17.03.2016.

[12] M. Likhachev, *A* and weighted a* search*, Carnegie Mellon University, Vorlesungsfolien, 2010, online unter `https://www.cs.cmu.edu/~motionplanning/lecture/Asearch_v8.pdf`; abgerufen am 17.03.2016.

[13] S. Koenig und M. Likhachev, "D* lite", *International Scientific Conference on Mobility and Transport*, 2002, pages 1-12.

[14] M. Schmidt, "Evaluierung gitterbasierter Pfadplanungs-Algorithmen für die Hardwarebeschleunigung mit FPGAs", Dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013.

[15] K. Arras, *Human-Oriented Robotics*, University of Freiburg, Vorlesungsfolien, 2013/2014, online unter `http://srl.informatik.uni-freiburg.de/teachingdir/ws13/slides/12-RobotMotionPlanning.pdf`; abgerufen am 17.03.2016.

[16] Y. Koren und J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation", *IEEE Conference on Robotics and Automation*, 1991.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006, online unter `http://planning.cs.uiuc.edu/`; abgerufen am 17.03.2016.

[18] D. Leidner, "Pfadplanung für Aufgaben in der realen Welt ausgeführt durch reale Robotersysteme", Masterarbeit, Hochschule Mannheim, 2011.

[19] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli und S. Teller, "Anytime Motion Planning using the RRT*", *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[20] B. Akgun und M. Stilman, "Sampling Heuristics for Optimal Motion Planning in High Dimensions", *IEEE International Conference on Robots and Systems (IROS)*, 2011.

[21] A. Yershova, L. Jaillet, T. Simeon und S. M. LaValle, "Dynamic-Domain RRTs:Efficient Exploration by Controlling the Sampling Domain", *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[22] J. J. Kuffner Jr und S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning", *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[23] M. Safonova, *Forward Arc Motion Primitives*, Website, Online unter `http://sbpl.net/node/53`; abgerufen am 17.03.2016.

[24] E. Assmann, "Untersuchung über den Einfluss einer Bremsweganzeige auf das Fahrerverhalten", Dissertation, Technische Universität München, 1985.

[25] J. Boger, "Bahnplanungsframework für ein autonomes Fahrzeug", Masterarbeit, Carl von Ossietzky Universität Oldenburg, 2013.

[26] S. Chun-Yi, R. Subhash und L. Honghai, *Intelligent Robotics and Applications*. Springer Berlin Heidelberg, 2012.

[27] T. Steinfeld, *Bezierkurven*, Website, Online unter `http://www.mathepedia.de/Bezierkurven.aspx`; abgerufen am 17.03.2016.

[28] M. Horn, *Regelungstechnik*, TU Graz, Vorlesungsskript, 2015, online unter `http://www.tugraz.at/fileadmin/user_upload/Institute/IRT/Skripten/Regelungstechnik_Horn_Juni_2015_.pdf`; abgerufen am 17.03.2016.

[29] J. Adamy, *Nichtlineare Systeme und Regelungen*. Springer Vieweg, 2014.

[30] M. Zeitz, "Differenzielle Flachheit: Eine nützliche Methodik auch für lineare SISO-Systeme", *Automatisierungstechnik*, Bd. 58, Nr. 1, Jan. 2010.

[31] M. Woernle, *Fahrmechanik*, Universität Rostock, Vorlesungsskript, 2005.

Acknowledgements

Diese Arbeit entstand am VIRTUAL VEHICLE Research Center in Graz, Österreich. Die Autoren bedanken sich für die Förderung im Rahmen des COMET K2 - Competence Centers for Excellent Technologies Programms des Österreichischen Bundesministeriums für Verkehr, Innovation und Technologie (bmvit), des Österreichischen Bundesministeriums für Wissenschaft, Forschung und Wirtschaft (bmwfw), der Österreichischen Forschungsförderungsgesellschaft mbH (FFG), des Landes Steiermark sowie der Steirischen Wirtschaftsförderung (SFG).