Hochschule für Angewandte Wissenschaften Hamburg

Hamburg University of Applied Sciences

Masterarbeit

Johann Heitsch

Verteilte Koordination und Pfadplanung von autonomen Fahrzeugen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Johann Heitsch

Verteilte Koordination und Pfadplanung von autonomen Fahrzeugen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Stephan Pareigis
Zweitgutachter : Prof. Dr. Franz Korf

Abgegeben am 30. August 2010

Johann Heitsch

Thema der Masterarbeit

Verteilte Koordination und Pfadplanung von autonomen Fahrzeugen

Stichworte

verteilte Koordination, Pfadplanung, Trajektorienplanung, Fahrzeugkinematik, verteilte Systeme, autonome Fahrzeuge, Roboter

Kurzzusammenfassung

Autonome Fahrzeuge benötigen eine robuste Pfadplanung. Diese Arbeit beschreibt verschiedene Ansätze zur Pfadplanung. Dabei werden kinematische Zwangsbedingungen von Fahrzeugen berücksichtigt. Konkrete Verfahren sind umgesetzt und miteinander verglichen worden. Zur Vermeidung von Kollisionen zwischen mehreren Fahrzeugen ist zudem ein verteilter Algorithmus beschrieben, mit dessen Hilfe Fahrzeuge dezentral koordiniert werden können. Die Koordination ist optimal bezüglich einer verteilten Gesamtbewertungsfunktion.

Johann Heitsch

Title of the paper

Decentralized coordination and pathplanning for autonomous vehicles

Keywords

distributed coordination, pathplanning, trajectory-planning, kinematics, distributed systems, autonomous vehicles, robots

Abstract

This paper deals with the pathplanning of autonomous vehicles. Thereby the kinematic constraints of the vehicle are regarded. Concrete methods had been implemented and compared. Due to the avoidance of collisions among vehicles an distributed algorithm for the decentralized coordination of autonomous vehicles is described. This algorithm is optimal in respect of an distibuted weighting function.

Inhaltsverzeichnis

1 Einführung

Bereits seit vielen Jahren gibt es ein großes Interesse an dem Forschungsgebiet der autonomen Roboter. Die Forschung wurde duch die wissenschaftlichen Fortschritte im Bereich der Technologie, Elektronik und Computertechnik stark vorangetrieben. Somit gibt es bereits viele Anwendungen, bei denen autonome Roboter die Tätigkeiten von Menschen übernehmen. Dieses ist meistens der Fall, wenn es bei den Tätigkeiten auf Schnelligkeit, Präzision und Zuverlässigkeit ankommt oder wenn es sich um Fließbandarbeiten handelt. Das klassische Beispiel hierfür ist die Automobilindustrie. Die Notwendigkeit der kostenoptimierten Fertigung von Automobilen für den Massenmarkt und die steigenden Ansprüche an die Qualität und Variantenvielfalt waren der Startschuss für den Einsatz von Robotertechnik (vgl. [Jacob (2004)]). Die meisten Roboter in der Industrie haben nur wenig „Intelligenz“. Die Bewegungsabläufe und Aktionen sind fest vorgegeben. Autonome mobile Roboter, wie sie im nächsten Kapitel beschrieben sind, können nicht durch feste Bewegungsabläufe gesteuert werden.

1.1 Autonome mobile Roboter

Unter autonome mobile Systeme fallen alle Roboter, die nicht stationär sind und die Aufgaben selbstständig ausführen können. Jedoch gibt es keine einheitliche Definition. Diese Arbeit beschäftigt sich mit autonomen Fahrzeugen. Dabei handelt es sich um eine Teilmenge der autonomen mobilen Roboter. Diese Fahrzeuge benötigen für die Lösung ihrer Aufgaben mehr „Intelligenz“ als die stationären Industrieroboter. Mittels einer Umweltkarte und den Zielvorgaben ist ein Pfad zu berechnen, der das Fahrzeug zum Ziel führt. Besteht die Möglichkeit, dass sich Fahrzeuge in der Umwelt begegnen, ist eine Koordination notwendig.

Es gibt in der Industrie bereits eine Menge von Anwendungen, bei denen autonome mobile Fahrzeuge zum Einsatz kommen. Seit dem Jahr 2002 kommen in Containerterminal Altenwerder *Automated Guided Vehicles* für den vollautomatischen Umschlag von Containern zum Einsatz [Hamburger Hafen und Logisik AG (2010)]. In der *Gläsernen Manufaktur* der Volkswagen AG in Dresden werden die Werkzeuge und Bauteile von *Fahrerlosen Transportsystemen* (FTS) *just in time* zu ihrem Einsatzort befördert [Volkswagen AG;FTS (2009)]. Die Firma STILL- GmbH [STILL GmbH (2009)] hat eine Pilotanlage aufgebaut, in der autonome

Gabelstabler im Mischbetrieb mit manuellen Gabelstablern zusammenarbeiten. Ausserhalb der Industrie gibt es auch eine Menge von autonomen Fahrzeugen. So wurde 2004 die *DARPA Grand Challenge* von der Technologieabteilung des US-amerikanischen Verteidigungsministeriums ins Leben gerufen [DARPA]. Bei diesem Wettbewerb haben autonome Fahrzeuge die Aufgabe, einen vorgegebenen Parcour innerhalb einer festgelegten Zeit zu überwinden. Im RoboCup [RoboCup] treten diverse Arten von Robotern in verschiedenen Disziplinen, wie Roboterfußball, gegeneinander an. Im Carolo-Cup [Cup (2010)] der TU Braunschweig treten autonome Modellfahrzeuge gegeneinander an, wobei es das Ziel ist, den Regeln entsprechend einen Rundkurs möglichst schnell und fehlerfrei zu fahren.

Bei allen Anwendungen ist die Pfadplanung und die Koordination der Fahrzeuge von sehr wichtiger Bedeutung. Nach [Weisser u. a. (1999)] gehört die Pfadplanung auch in das Themengebiet der *Fahrerassistenzsysteme*, das an der Hochschule für Angewandte Wissenschaften Hamburg in dem Projekt FAUST erforscht wird. Diese Arbeit enstand im Rahmen dieses Projektes.

1.2 Pfadplanung

Die Pfadplanung für ein autonomes Fahrzeug hat die Aufgabe, einen Pfad zu berechnen, auf dem das Fahrzeug von seiner aktuellen Position zu einer vorgegebenen Zielposition fahren kann. Dabei darf es nicht zu Kollisionen mit den in der Umwelt vorhandenen Hindernissen kommen. Die Arbeiten Ersson und Hu (2001), [Koenig und Likhachev (2002)], [Rebai u. a. (2007)] und die Reihe [Berns und Luksch (2008)] behandeln unter anderem die Pfadplanung von Fahrzeugen. Da sich Fahrzeuge nicht immer in alle Richtungen bewegen können, sondern kinematischen und dynamischen Zwangsbedingugen unterliegen, ist es notwendig, diese in der Pfadsuche mit zu berücksichtigen.

Die Arbeiten [Rebai u. a. (2007)], [Bobyr und Lumelsky (1999)], [Latombe (2008)], [Samuel und Keerthi (1993)] und [Zulli u. a. (1995)] erweitern die Pfadsuche, so dass Pfade für Fahrzeuge gefunden werden können, die sich, wie ein Auto, nicht auf der Stelle drehen können.

Um von der Pfadplanung und den genauen Eigenschaften eines Fahrzeuges zu abstrahieren, wird ein Konfigurationsraum eingeführt. Dieses ist in den Arbeiten [Latombe (1991)], [Lozano-Perez (1981)] und [Lozano-Perez (1983)] beschrieben. Ein Fahrzeug wird im Konfigurationsraum als ein Punkt betrachtet. Alle Hindernisse werden um die Abmessungen des Fahrzeuges erweitert. Der Konfigurationsraum wird mithilfe von diversen Algorithmen durchsucht. Ziel ist es, die Komplexität der Pfadsuche möglichst gering zu halten. Dazu kann der Konfigurationsraum in Zellen aufgeteilt und anschließend in diesen Zellen ein Pfad gesucht

werden. Effizientere Ansätze für die Zerlegung des Konfigurationsraumes, wie der *Sampling Based* Ansatz oder der *rapidly-exploring random tree*, sind in [Melchior und Simmons (2007)], [LaValle (2006)], [Shan u. a. (2009)] und [Kuffner und LaValle (2000)] beschrieben.

Der Potentialfeldansatz, bei dem ein Fahrzeug als Partikel in einem Kraftfeld betrachtet wird, wurde in den Arbeiten Koditschek (1987) und Rimon und Koditschek (1992) erstmalig dargestellt. Durch diesen Ansatz ensteht eine Navigationsfunktion, die zu jeder Position die optimale Richtung zum Ziel angibt. Ansätze für komplette Navigationssysteme sind in [Schubert (2006)], [Mojaev (2001)], [Deutsch (2004)] und [Gutmann (2000)] beschrieben.

1.3 Koordination

Befinden sich mehrere Fahrzeuge in derselben Umwelt, was meistens der Fall ist, so ist eine Koordination dieser Fahrzeuge notwendig. In den derzeitigen industriellen Anlagen werden alle Fahrzeuge zentral koordiniert. Die Pfadsuche findet auf einem zentralen Koordinierungsrechner statt, der alle Informationen über alle Fahrzeuge zu jeder Zeit hat. Der Zustandsraum wächst linear mit der Anzahl der Fahrzeuge und erreicht schnell eine sehr hohe Dimension.

Nur durch viel Rechenleistung können Pfade geplant werden. Wichtig bei der Planung der Pfade für mehrere Fahrzeuge ist, dass es nicht zu Kollisionen und Deadlocks kommt. In [Erdmann und Lozano-Perez (1986)], [Svestka und Overmars (1995)], [Akella und Hutchinson (2002)], [Peasgood u. a. (2008)], [Bennewitz (2004)], [van den Berg und Overmars (2005)] und [Ferrari u. a. (1995)] sind Ansätze für zentrale Lösungen beschrieben. Dabei kommen entkoppelte und priorisierende Ansätze zum Einsatz. Bei entkoppelten Ansätzen wird für jedes Fahrzeug ein Pfad geplant und in einem zweiten Schritt werden die Pfade so modifiziert, dass es zu keinen Konflikten kommt. Bei den priorisierenden Ansätzen wird für jedes Fahrzeug, beginnend mit dem Fahrzeug mit der höchsten Priorität, ein Pfad geplant, der Fahrzeuge mit höherer Priorität als Hindernis ansieht. Das Problem der Koordination wird hierbei auf das Problem der Suche nach einem geeigneten Priorisierungsschema verschoben.

Um die Rechenleistung und damit die Komplexität der Pfadplanung und Koordination auf alle Fahrzeuge zu verteilen, kann die Planung auf die einzelnen Fahrzeuge ausgelagert werden. In diesem Fall plant jedes Fahrzeug autonom einen Pfad und koordiniert sich selbstständig mit anderen Fahrzeugen. Es gibt somit keine zentrale Koordinationsstelle mehr. Ansätze sind in [Mutambara und Durrant-Whyte (1993)], [Roszkowska (2008)], [Takahashi und Ohnishi (2003)], [Azarm und Schmidt (1997)], [Lumelsky und Harinarayan (1997)], [Naumann u. a. (1998)], [Naumann u. a. (1997)] und [Pallottino u. a. (2007)] beschrieben.

1.4 Zielsetzung

In dem Projekt Fahrerassistenz- und Autonome Systeme [FAUST HAW Hamburg] wird ein Navigationssystem für autonome Fahrzeuge entwickelt. Eine hierfür benötigte Lokalisierung und Kartenerstellung ist in der Arbeit [Rull (2010)] beschrieben. Ziel dieser Arbeit ist es, einen verteilten Koordinationsalgorithmus zu entwickeln, der Fahrzeuge gemäß einer verteilten Gesamtbewertungsfunktion optimal koordiniert. Diese Bewertungsfunktion sollte aus mehreren, pro Fahrzeug individuellen, Bewertungsfunktionen bestehen. Zudem wird der Stand der Forschung im Bereich der Pfad- und Bewegungsplanung für Fahrzeuge ermittelt und für die Fahrzeuge der Hochschule für Angewandte Wissenschaften Hamburg aus dem FAUST Projekt umgesetzt werden. Durch die Verknüpfung der Arbeiten im FAUST Projekt ensteht somit ein Navigationssystem für autonome Fahrzeuge.

1.5 Gliederung

Die Arbeit gliedert sich in zwei Teile. Das Kapitel 2 beschreibt die Pfadplanung. Dabei wird in Kapitel 2.1 die allgemeine Problemstellung der Pfadplanung beschrieben. In Kapitel 2.2 sind die Problemstellung und Lösungsansätze für die diskrete Pfadplanung beschrieben. Dazu gehören unter anderem Suchalgorithmen für Graphen.

Das Kapitel 2.3 behandelt die geometrische Repräsentation der Umwelt in einem Computer. Zur Abstraktion vom konkreten Pfadplanungproblem wird in Kapitel 2.4 der Konfigurationsraum eingeführt. Dabei wird dieser für einfache starre Roboter in Kapitel 2.4.1 definiert. In Kapitel 2.4.2 ist beschrieben, wie Hindernisse in den Konfigurationsraum transformiert werden. Anschließend wird in Kapitel 2.4.3 die Problemstellung aus Kapitel 2.1 mit Hilfe des Konfigurationsraumes verallgemeinert.

Im Kapitel 2.5 werden Ansätze für die Pfadplanung im Konfigurationsraum vorgestellt. Kapitel 2.5.1 führt die abtastenden Ansätze ein, wobei Kapitel 2.5.2 kombinatorische Ansätze enthält. Das Kapitel 2.5.3 beschreibt die Notwendigkeit von rückgekoppelten Algorithmen und Lösungsansätzen. Um die Planung in nicht zeitveränderlichen Umwelten zu ermöglichen, ist in Kapitel 2.6 beschrieben, auf welche Weise die Zeit in der Planung mitberücksichtigt werden kann. Für die Planung vom Pfaden für autoähnlichen Fahrzeugen ist Wissen über die Kinematik eines Fahrzeuges notwendig.

Das Kapitel 2.7 beschreibt, wie das kinematische Modell für ein autoähnliches Fahrzeug mathematisch formuliert werden kann. Das Kapitel 2.8 bringt die Fahrzeugkinematik und die Pfadplanung zusammen. Es ist erläutert, wie Pfade für Fahrzeuge gefunden werden können, die kinematischen Zwangsbedingungen unterliegen. In Kapitel 2.9 ist die Umsetzung und Implementierung der einzelnen Ansätze und Algorithmen beschrieben.

Der zweite Teil dieser Arbeit ist die Koordination in Kapitel 3. Zu Beginn wird in Kapitel 3.1 das Problem der Koordination von Fahrzeugen erläutert. In Kapitel 3.2 sind allgemeine Ansätze zur Koordination beschrieben. Das Kapitel 3.3 erklärt den Ansatz und die Umsetzung des neuen Koordinationsalgorithmus. Dabei wird in Kapitel 3.3.1 darauf eingegangen, wie Pfade dargestellt werden. Die Konflikterkennung zwischen zwei Pfaden ist in Kapitel 3.3.2 beschrieben. Der Algorithmus in Kapitel 3.3.3 gibt an, wie Pfade modifiziert werden können und das Kapitel 3.3.4 beschreibt, wie die Modifikationen bewertet werden. Im Anschluss in Kapitel 3.3.5 ist der Koordinationsablauf und das Protokoll beschrieben, über das die Fahrzeuge koodiniert werden.

2 Pfadplanung

Die Pfadplanung ist für viele Aufgaben in der Robotik von sehr großer Bedeutung. Sie taucht in der Robotik an vielen Stellen und in verschiedene Variationen auf. Bei Roboterarmen mit Werkzeugen wird eine Pfadplanung benötigt, die das Werkzeug an einer vorgegebenen Position positioniert. Bei autonomen Fahrzeugen wird die Pfadplanung benötigt,um die Fahrzeuge zu ihren Zielpositionen zu bewegen.

2.1 Problemstellung

Allgemein kann das Problem der Pfadplanung wie folgt beschrieben werden: Gegeben ist eine Startposition (die aktuelle Position), eine Zielposition und eine Umgebungskarte. Gesucht ist eine Abfolge von Aktionen, die den Roboter kollisionsfrei von der Start- zur Zielposition bewegt. Kollisionsfrei bedeutet, dass der Roboter, während er den Pfad abfährt, nicht mit Hindernissen zusammenstößt.

Es gibt eine Vielzahl von Ansätzen,die das Pfadplanungproblem löst. Ein guter Ansatz sollte für eine möglichst große Klasse von Roboterarten und ihren Einsätzen geeignet sein. Somit ist es nötig,die Pfadplanung auf einer abstrahierten Ebene auszuführen. Dazu wird im wird im folgenden Kapitel (Kapitel 2.4) ein Konfigurationsraum eingeführt, der von dem Aufbau und Eigenschaften des eigentlichen Roboters abstrahiert.

2.2 Diskrete Planungsverfahren

Seit den 1970er Jahren sind Pfadplanungsalgorithmen Gegenstand der Forschung. Es gibt diverse Ansätze, die das Problem der Pfadplanung versuchen zu lösen, jedoch gibt es bis heute kein Verfahren, das alle Fragen und Probleme der Pfadplanung allgemein löst. 1990 veröffentlichte Latombe das bis heute wichtigste Standardwerk zu diesem Thema: [Latombe (1991)]. Zum erstenmal wurde das Konzept des Konfigurationsraumes eingeführt. Lavalle

veröffentlichte 2006 das Buch [LaValle (2006)]. In diesem sind viele Grundlagen, neue Ansätze und Anwendungen von Planungsverfahren beschrieben. Eine mathematische und systemtheoretische Betrachtung ist unter der Berücksichtigung von kinematischen Bedingungen in [Laumond (1998)] gegeben.

Im Folgenden werden Verfahren für die Pfadplanung vorgestellt. Diese betrachten ein Fahrzeug ohne ihre kinematischen Beschränkungen, also als ein *free flying object*, welches sich in jede Richtung frei bewegen kann. Die dynamischen Beschränkungen werden vernachlässigt, was bedeutet, dass ein Fahrzeug innerhalb einer unendlich kurzen Zeit komplett abbremsen und auf die volle Geschwindigkeit beschleunigen kann.

Zum Ende des Kapitels wird beschrieben, wie dynamische und kinetische Beschränkungen in die Planung mit aufgenommen werden können.

Jedes, der im folgenden beschriebenden Pfadplanungsverfahren, führt eine *Diskretisierung* durch, um damit die unendliche Menge an möglichen Konfigurationen zu verringern. Diese Verringerungen entstehen durch eine Überführung des Konfigurationsraumes in einen Graphen. Daher wird im folgenden auch auf die Graphentheorie eingegangen.

2.2.1 Problemstellung

Die Planung von Pfaden basiert auf einem Zustandsraummodell. Bei diesem Modell wird jede mögliche Situation in der Umwelt als ein *Zustand* modelliert, der mit x bezeichnet wird. Der Zustandsraum wird mit X bezeichnet. Für die diskrete Pfadplanung ist es notwendig, dass X abzählbar ist. In den meisten Fällen ist X zusätzlich endlich. Ein Zustand enthält so wenig wie möglich Informationen, aber so viele wie für die Pfadplanung nötig sind, damit eine Lösung gefunden werden kann. Die Umwelt kann durch *Actionen* u beeinflusst werden, die vom Planer ausgewählt werden. Jede Aktion, die auf einen Zustand x angewandt wird, produziert einen neuen Zustand x'. Diese Transformation wird als *Zustandsübergangsfunktion* f bezeichnet. Die dazugehörige *Zustandsübergangsgleichung* lautet

$$x' = f(x, u) \tag{2.1}$$

$U(x)$ sei der Aktionsraum für jeden Zustand x, der eine Menge von Aktionen darstellt, die auf x angewandt werden können. So gilt für zwei unterschiedliche Zustände $x, x' \in X$, dass $U(x)$ und $U(x')$ nicht disjunkt sein müssen. Es können also die gleichen Aktionen auf verschiedene Zustände angewandt werden. Die Menge aller Aktionen (der gesammte Aktionsraum) ist:

$$U = \bigcup_{x \in X} U(x) \tag{2.2}$$

Ein Teilproblem der Pfadplanung ist es, zu einer gegebenen Teilmenge $X_G \subset X$, die als *Zielzutände* bezeichnet werden, eine endliche Folge von Aktionen zu finden, die den *Startzustand* x_i in einen Zustand aus X_G überführen. Zusammengefasst ist die Problemstellung:

Problem 2.1. *Diskrete Pfadplanung*

1. *Ein nicht leerer Zustandsraum X, der eine endliche oder abzählbare Menge von Zuständen enthält.*
2. *Für jeden Zustand x ist ein endlicher Aktionsraum $U(x)$ vorhanden.*
3. *Eine* Zustandsübergangsfunktion *f ist definiert, die für alle $x \in X$ und $u \in U(x)$ einen $f(x, u) \in X$ produziert. Die* Zustandsübergangsgleichung *ist von f abgeleitet $x' = f(x, u)$.*
4. *Ein Anfangszustand x_i*
5. *Eine Menge von Zielzuständen X_G.*

Das Problem 2.1 wird häufig in einem *Zustandsübergangsgraphen* dargestellt. Dabei sind die Zustände des Zustandsraumes X die Knoten. Eine gerichtete Kante zwischen $x \in X$ und $x' \in X$ exisitiert nur, wenn es ein $u \in U(x)$ gibt, für das $x' = f(x, u)$ gilt. Der Anfangszustand und die Zielzustände werden als spezielle Knoten markiert. Somit genügt der Zustandsübergangsgraph dem Problem 2.1.

Beispiel 2.1. *Bewegung in einem 2D Raster*
Angenommen: ein Roboter bewegt sich in einem Raster, in dem jede Zelle eine Koordinate der Form (i, j) hat. Der Roboter kann diskrete Schritte in eine der vier Richtungen (oben, unten, links, rechts) machen.

Der Zustandsübergangsgraph ist in Abbildung 2.1 dargestellt. Mit dem Problem 2.1 kann das System folgendermaßen spezifiziert werden: Der Zustandsraum X enthält Paare von (i, j), wobei $i, j \in \mathbb{N}$ gilt.Sei der Aktionsraum $U = \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$ mit $U(x) = U$ für alle $x \in X$. Die Zustandsübergangsgleichung sei $f(x, u) = x + u$, betrachet als zweidimensionale Vektoraddition, mit $x \in X$ und $u \in U$. Angenommen: $x_i = (0, 0)$ und $X_G = \{(100, 100)\}$, so ist es einfach, eine Folge von Aktionen zu finden, die $(0, 0)$ in $(100, 100)$ überführen.

Erschwert werden kann das Beispiel dadurch, dass einige Zellen als Hindernisse gekennzeichnet werden, die für den Roboter als nicht passierbar gelten. Für den Zustandsübergangsgraphen bedeutet dieses, dass die Knoten, die ein Hindernis darstellen, mit allen ihren Kanten entfernt werden.

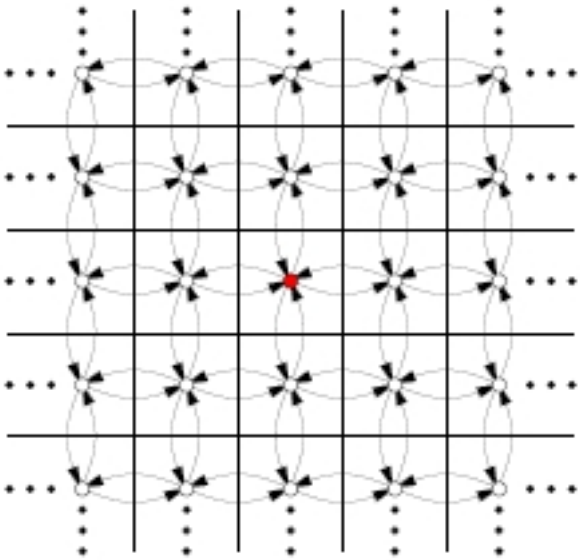


Abbildung 2.1: Zustandsübergangsgraph für ein unendliches Gitter (aus [LaValle (2006), S 30])

Die Berechnung des kompletten Zustandsübergangsgraphen ist in der Regel nicht notwendig oder möglich. Bei unendlichen Zustandsräumen X würde ein unendlicher Graph entstehen, dessen Vorausberechnung nicht möglich ist. Das Beispiel 2.1 zeigt, dass nur ein sehr kleiner Ausschnitt aus dem Zustandsraum für die Planung notwendig ist. Dieser kann während der eigentlichen Suche im Graphen erstellt werden.

Ein endlicher Zustandsraum zusammen mit dem Problem 2.1 ist äquivalent zu der Definition eines *endlichen Automaten*. Diesbezüglich können die Aktionen als Eingabewerte und die Zustände als Ausgaben des Automaten bezeichnet werden. Die Pfadplanung kann als Automat betrachtet werden, bei dem getestet wird, ob eine Eingabesequenz zur gewünschten Ausgabe führt.

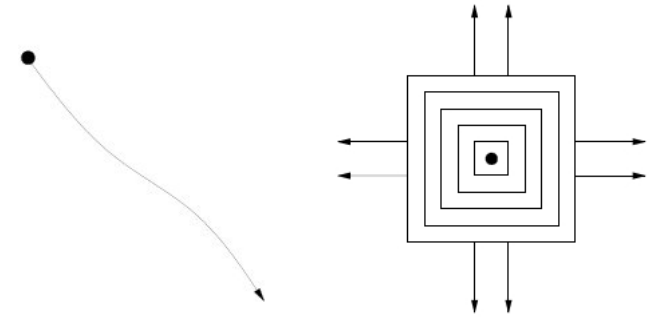
Eine ähnliche Verbindung kann auch zu *deterministischen endlichen Automaten (DEA)* gezogen werden. Die Definition eines DEAs enthält eine Menge von Finalzuständen, die den Zielzuständen X_G entsprechen und einem Startzustand, der dem Startzustand x_i aus dem Problem 2.1 entspricht. Die Pfadplanung mit einem DEA kann als die Suche nach einer Eingabesequenz interpretiert werden, die der DEA akzeptiert. Auch kann das Problem 2.1 als Petrinetz aufgefasst werden. Hierbei besteht die Suche darin, eine Schaltfolge von Transitionen zu finden, so dass das Petrinetz in einem Endzustand endet.

Diese Menge an Repräsentationen und Interpretationen des Pfadplanungsproblems ermöglichen eine kompakte Darstellung und diverse Herangehensweisen an das Problem.

2.2.2 Suche in einem Graphen

Die in diesem Abschnitt beschriebenen Algorithmen für die Suche in Graphen beruhen auf [LaValle (2006), S.32ff] und [Cormen u. a. (2001), S.527ff].

Definition 2.1. *Endliche Graph*
Ein endlicher Graph ist ein Tupel $G = (V, E)$ und besteht aus einer endlichen Menge von Knoten V und einer endlichen Menge von Kanten E. Dabei sind die Kanten Paare von Knoten $e = (v_1, v_2)$, die diese miteinander verbinden.



(a) Nicht systematisch (b) systematisch

Abbildung 2.2: Systematik von Suchalgorithmen (aus [LaValle (2006),S 32])

Eine wichtige Vorraussetzung für die in diesem Abschnitt vorgestellten Algorithmen ist die *Systematik*. Ist ein Graph endlich, so besucht der Algorithmus jeden erreichbaren Zustand. Somit ist in einer endlichen Zeit entscheidbar, ob eine Lösung existiert. Dieses impliziert, dass sich der Algorithmus merkt, welche Zustände bereits bearbeitet wurden, um Zyklen zu verhindern. Diese Systematik wird für unendliche Graphen nicht erweitert. Exisitert ein Pfad, so muss dieser in endlicher Zeit gefunden werden. Ist kein Pfad vorhanden, so kann der Algorithmus unendlich laufen. Bezogen auf das Beispiel 2.1, das einen unendlichen Graphen hat, bedeutet dieses, dass eine Suche wie in Abbildung 2.2(a) keine Systematik hat, da diese nur in eine Richtung sucht und ein großer Raum unaufgedeckt bleibt. Eine wellenförmige Suche wie in Abbildung 2.2(b) ist hingegen systematisch.

Allgemeine Vorwärtssuche

Der in Algorithmus 2.1 beschriebene Algorithmus beschreibt eine einfache Vorwärtssuche. Diese dient als Basis für die im Folgenden beschriebenen Algorithmen. Jedem Zustand $x \in X$ kann während der Suche einer der folgenden Zustände zugewiesen werden:

Algorithm 2.1 Allgemeine Vorwätssuche

```
1: Q.insert(x_i) and mark x_i as visited
2: while Q ≠ ∅ do
3:    x ← Q.getFirst()
4:    if x ∈ X_G then
5:       return SUCCESS
6:    end if
7:    for all u ∈ U(x) do
8:       x' = f(x, u)
9:       if x' not visited then
10:         mark x' as visited
11:         Q.insert(x')
12:      else
13:         Resolve duplicate x'
14:      end if
15:   end for
16: end while
17: return FAILURE
```

1. **Unvisited (unbesucht):** Zustände, die bisher nicht besucht worden sind. Initial: $X \setminus \{x_i\}$.
2. **Dead (tot):** Zustände, die besucht worden sind und deren Folgezustände alle besucht worden sind. Initial: $\emptyset$.
3. **Alive (lebend):** Zustände, die gefunden wurden, aber deren Folgezustände noch unbesucht sind. Initial: x_i.

Die Menge der lebenden Zustände wird ein einer priorisierten Queue Q gepeichert. Hierzu ist eine Priorisierungsfunktion notwendig. Die Suchalgorithmen unterscheiden sich im wesentlichen in der Wahl dieser Priorisierungsfunktion. Der beschriebene Algorithmus wählt jeweils den nächsten Zustand $x \in Q \subset X$ aus und fügt alle Folgezustände in die Queue Q ein. Zu beachten ist, dass dieser Alorithmus in dieser Form nur feststellen kann, ob es eine Lösung gibt. Um einen Pfad (Sequenz von Aktionen) zu berechnen, kann nach der Zeile 8 eine Funktion eingesetzt werden, die das x' mit seinem Vorfolger x assoziiert.

An dem Algorithmus 2.1 sind bereits viele Probleme der Suche in einem Graphen aufzeigbar. Die Prüfung, ob $x \in X_G$ in Zeile 4ist, scheint trivial. Jedoch ist diese abhängig von der Repräsentation des Zustandes x und der Zielzustände X_G. Die Zeilen 9 und 10 sind konzeptuell einfach, jedoch bringt die Umsetzung viele Probleme mit sich. Zum Beispiel die Entscheidung, wann zwei Zustände gleich sind. Abhängig von der den Zuständen zugrundeliegenden Modellierung sind mehrere Möglichkeiten vorhanden, die besuchten Zustände

abzuspeichern. Sind die Zustände, wie in Abbildung 2.1, in einem Raster angeordnet, so kann in einer Lookup Table gespeichert werden, ob ein Zustand bereits besucht wurde. Ein allgemeinerer Ansatz ist, die Vorfolgerassoziation in einem Baum vorzunehmen. Befindet sich bereits ein Zustand in diesem Baum, so ist dieser bereits besucht.

Einige Algorithmen erweitern die allgemeine Vorwärtssuche um eine Kostenfunktion. Diese kann Einfluss auf die Priorisierungsfunktion der Queue haben. Zudem können in Zeile 13 Berichtigungen der Kosten hinzukommen.

Spezielle Vorwärtssuche

Im folgenden werden einige spezielle Suchalgorithmen vorgestellt, die im wesentlichen auf der allgemeinen Vorwärtssuche 2.1 basieren. Sie sind klassische Algorithmen für die Suche in Graphen [Cormen u. a. (2001)].

Breitensuche Der Algorithmus 2.1 beschreibt die Breitensuche, wenn Q als FIFO (First-In First-Out) spezifiziert ist. Die Breitensuche besucht zuerst alle Knoten, die den Abstand k zum Startknoten haben, bevor sie einen Knoten mit dem Abstand $k + 1$ besucht. Das Suchmuster ähnelt einer Serie von Wellen, daher handelt es sich um einen systematischen Algorithmus (siehe Abbildung 2.2(b)). Die Komplexität beträgt $\mathcal{O}(|V| + |E|)$, wobei $|V|$ und $|E|$ die Anzahl der Knoten und Kanten sind. Bei dieser Abschätzung wird jedoch davon ausgegangen, dass die Operationen als auch die Entscheidung, ob ein Knoten bereits besucht wurde, in einer konstanten Zeit verarbeitet werden. Da dieses jedoch selten der Fall ist, muss dieses in der Berechnung der Komplexität mit berücksichtigt werden.

Tiefensuche Im Gegensatz zur Breitensuche wird bei der Tiefensuche Q als LIFO (Last-In First-Out) spezifiziert. Durch diesen Stack verändert sich die Suchweise. Die Tiefensuche besucht alle Knoten, die vom zuletzt gefundenen Knoten erreichbar sind. Es wird in der Tiefe des Graphen gesucht. Ist X eine endliche Menge, so ist die Tiefensuche systematisch. Für eine unendliche Menge X jedoch nicht, da sich die Suche wie in Abbildung 2.2(a) verhalten kann.

Dijkstra Algorithmus Bisher ist die Auswahl des als nächsten zu untersuchenden Knoten nur abhängig von der gewählten Listenart von Q gewesen. Es wurden keine Knoten priorisiert. Der Dijkstra Algorithmus [Dijkstra (1959)] führt hier eine Neuerung ein. Eine Kostenfunktion $w : E \to \mathbb{R}$ bestimmt für jede Kante $e \in E$ die Kosten $w(e)$. Die Funktion kann auch als $w(x, u)$ beschrieben werden und liefert die Kosten, die entstehen, wenn die

Action u auf den Zustand x angewandt wird. $g : X \times X \to \mathbb{R}$ beschreibt die gleiche Kostenfunktion. $g(x, x'), x, x' \in X$ liefert ∞, wenn es keine Kante zwischen den Knoten gibt, also wenn gilt $\neg(\exists u \in U(x) : f(x, u) = x')$. Ansonsten wird $w(x, u)$ geliefert. Somit gilt, wenn $x' = f(x, u)$, dann ist $w(x, u) = g(x, x')$. Die Gesammtkosten für einen Pfad $p = \langle v_0, v_1, \ldots, v_k \rangle$ sind somit:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i). \tag{2.3}$$

Die Kosten des kürzesten Pfades von x_i nach x_g werden durch

$$\delta(x_i, x_g) = \begin{cases} \min\{w(p) : x_i \overset{p}{\rightsquigarrow} x_g\} & \text{falls ein Pfad von } x_i \text{ nach } x_g \text{ exisitiert} \\ \infty & \text{sonst} \end{cases} \tag{2.4}$$

definiert. Ein kürzester Pfad von x_i nach x_g ist ein Pfad p mit den Kosten $w(p) = \delta(x_i, x_g)$.

Beim Dijkstra Algorithmus wird die Liste Q nach einer Funktion $C : X \to [0, \infty]$ sortiert. Die Funktion $C(x)$ liefert die Kosten vom Startzustand x_i zum Zustand x. Die Funktion $C^*(x)$ liefert die optimalen Kosten vom Startzustand x_i zum Zustand x. Initial ist $C^*(x_i) = 0$. Jedesmal, wenn ein neuer Folgezustand x' berechnet wird, werden die Kosten zu dem Zustand $C(x') = C^*(x) + w(e) = C^*(x) + w(x, u) = C^*(x) + g(x, x')$ ermittelt.

An dieser Stelle kann noch nicht $C^*(x')$ geschrieben werden, da noch nicht bekannt ist, ob der berechnete Weg der optimale ist. Ist der Zustand x' bereits in Q enthalten (Zeile 9 in Algorithmus 2.1) und sind die Kosten $C(x')$ geringer, so werden dem Zustand in Q die neuen Kosten und ein neuer Vorgänger zugewiesen. Zudem wird Q neu sortiert (Zeile 13).

Die Kosten $C(x)$ werden zu $C^*(x)$, wenn der Zustand x aus Q entfernt wird. Das bedeutet, dass in der weiteren Suche kein günstigerer Pfad zu x gefunden werden kann. Die Komplexität beträgt $\mathcal{O}(|V| \lg |V| + |E|)$, wobei die selben Randbedingungen gelten, wie bei der Komplexitätsabschätzung der Breitensuche.

A-Stern Algorithmus Der A-Stern Algorithmus ist eine Erweiterung vom Dijkstra Algorithmus, der versucht, mit einer Metrik die Anzahl der zu besuchenden Knoten zu verringern. Die Metrik schätzt dabei die restlichen Kosten von einem gegebenen Zustand zum Ziel. Zu den Kosten $C(x)$, die schon im Dijkstra Algorithmus definiert wurden, wird eine weitere Funktion $G : X \to \mathbb{R}$ definiert, die die Restkosten von einem Zustand x zu einem Zielzustand in X_G liefert. Die Funktion $G(x)$ darf die Restkosten nicht überschätzen.

In vielen Anwendungen ist es möglich eine Funktion zu finden, die die Kosten einfach schätzt. Abbildung 2.1 ist ein gutes Beispiel. Ist (i, j) eine Koordinate und (i', j') eine andere, so ist $|i - i'| + |j - j'|$ eine Schätzung, da dieses die Länge des direkten Weges ohne Hindernisse

ist. Kommen Hindernisse in den Weg, so können die Kosten des Pfades nur wachsen, da der Roboter versucht, um diese herumzufahren. In vielen Fällen ist es jedoch nicht möglich, eine Metrik anzugeben, die den realen Restkosten zum Ziel sehr nahe kommt und diese nicht überschätzt. Die optimate Metrik wird im folgenden als $G^*(x)$ bezeichnet.

Der A-Stern Algorithmus arbeitet in derselben Weise, wie der Dijkstra Algorithmus, lediglich die Sortierung der Liste Q ist verändert. Diese wird jetzt nach der Summe $C^*(x') + G^*(x')$ sortiert. Also nach den geringsten Restkosten von x' nach X_G. Liefert $G^*(x)$ die wahren optimalen Restkosten für alle $x \in X$, so ist garantiert, dass der A-Stern Algorithmus den optimalen Pfad findet. Je näher die Funktion $G^*(x)$ an die wahren optimalen Kosten herankommt, desto weniger Knoten müssen, im Gegensatz zum Dijkstra Algorithmus, besucht werden. Ist $G(x) = 0 \forall x \in X$, so verhält sich der Algorithmus, wie der von Dijkstra. Der Algorithmus ist für endliche und unendliche Mengen X systematisch.

Beide Algorithmen, der Dijkstra und der A-Stern, sind mit Hilfe der dynamischen Programmierung umsetzbar. Die Funktion $C^*(x)$ löst dabei die Teilprobleme, die nicht immer neu berechnet werden müssen. $G^*(x')$ ist jedoch nicht auf diese Weise lösbar.

2.2.3 Optimale diskrete Planung

Das Problem 2.1 wird in diesem Kapitel so erweitert, dass optimale Pfade gefunden werden. Anstatt einen möglichen Pfad zu suchen, wird ein Pfad gesucht, der optimal bezüglich festgelegter Kriterien ist. Diese Kriterien können die Länge, Zeit, der Energieverbrauch oder anderes sein.

Um Pfade mit optimaler (minimaler) Länge zu suchen, wird folgende Notation eingeführt: Mit π_K wird ein Plan mit K Schritten bezeichnet, der eine Sequenz $(u_1, u_2, \ldots, u_K)$ von K Aktionen ist. Ist x_i und π_K gegeben, so ist eine Sequenz von Zuständen $(x_1, x_2, \ldots, x_K)$ daraus mit der Zustandsübergangsfunktion f generierbar, wobei $x_1 = x_i$ und $x_{k+1} = f(x_k, u_k)$ gilt.

Pfade fester Länge

Zu Beginn soll die Suche nach Pfaden mit einer festen Länge betrachtet werden. Diese kann später so erweitert werden, dass auch Pfade mit nicht vorgegebener Länge gefunden werden können.

Das Problem 2.1 wird um die Kostenfunktion L erweitert. Somit entsteht das folgende Problem:

Problem 2.2. *Optimale diskrete Planung mit fester Länge*

1. *Alle Bestandteile aus Problem 2.1 werden direkt übernommen: X, $U(x)$, f, x_i, X_G. Dabei wird davon ausgegangen, dass X endlich ist.*
2. *Eine Zahl K wird definiert, die die genaue Anzahl von Schritten eines Plans ist.*
3. *Ein Kostenfunktional L wird für den K-Schritt Plan π_K definiert. Es leifert die Kosten eines Pfades.*

$$L(\pi_K) = \sum_{k=1}^{K} w(x_k, u_k) + w_F(x_F) \tag{2.5}$$

Dabei ist $F = K + 1$, also der Zielschritt, $w(x_k, u_k)$ die Kostenfunktion, die für jedes $x_k \in X$ und $u_k \in U(x_k)$ die Kosten berechnet. $w_F(x_F)$ ist ausserhalb der Summe und gibt an: $w_F(x_F) = \begin{cases} 0 & \text{wenn } x_F \in X_G \\ \infty & \text{sonst} \end{cases}$

Ziel ist es nun einen Pfad zu finden, der L minimiert. Der Bruteforce Ansatz ist es, alle Aktionssequenzen der Länge K zu berechnen und zu bewerten. Jedoch ist die Zeitkomplexität dafür $O(|U|^K$ und somit zu groß. An dieser Stelle kann jedoch der Ansatz der dynamischen Programmierung genutzt werden. Denn die Lösung des Gesamtproblemes lässt sich in kleine Teilprobleme zerlegen. Denn Teilaktionssequenzen einer optimalen Gesamtsequenz sind auch optimal. Wäre dieses nicht so, so wäre auch die Gesamtsequenz nicht optimal. Die Berechnung kann mit dem *Werte Iterations Verfahren* erfolgen, bei dem iterativ die optimalen Kosten berechnet werden.

Rückwärts Iterations Verfahren Beim Rückwärts Iterations Verfahren werden die Restkosten vom Zielzustand x_F zu allen Zuständen berechnet. Für k von 1 bis F, sei G_k^* die Restkosten vom k-ten Zustand zum Zustand x_F:

$$G_k^*(x_k) = \min_{u_k, \ldots, u_K} \left\{ \sum_{i=k}^{K} w(x_i, u_i) + w_F(x_F) \right\} \tag{2.6}$$

$$G_k^*(x_k) = \min_{u_k} \left\{ w(x_k, u_k) + G_{k+1}^*(x_{k+1}) \right\} \tag{2.7}$$

In dem $\min$ in Gleichung 2.6 sind die Kosten der letzten $F - k$ Schritte summiert. Für die Randbedingung $k = F$ reduziert sich die Gleichung auf

$$G_F^*(x_F) = w_F(x_F) \tag{2.8}$$

.

Die exakte Herleitung der Gleichungen ist [LaValle (2006), S.46] zu entnehmen.

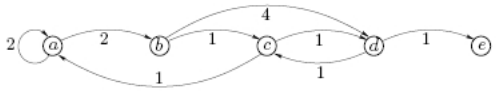


Abbildung 2.3: Fünf- stufiges Beispiel. Knoten repräsentieren Zustände, Kanten die Möglichkeit der Ausführbarkeit einer Aktion und die Kantengewichte repräsentieren $w(x_k, u_k)$ (aus [LaValle (2006),S 47])

Beispiel 2.2. *Rückwärts Iterations Verfahren*

	a	b	c	d	e
G_5^*	∞	∞	∞	0	∞
G_4^*	∞	4	1	∞	∞
G_3^*	6	2	∞	2	∞
G_2^*	4	6	3	∞	∞
G_1^*	6	4	5	4	∞

Abbildung 2.4: Die optimalen Restkosten, berechnet mit dem Rückwärts Iterations Verfahren (aus [LaValle (2006),S 48])

Abbildung 2.3 zeigt den Zustandsgraph für das Pfadplanungproblem, bei dem $X = \{a, b, c, d, e\}$ *ist. Angenommen* $x_i = a$, $X_G = \{d\}$ *und* $K = 4$*, so werden* $G_4^*, G_3^*, G_2^*, G_1^*$ *und die Zielstufe* G_5^* *berechnet. Das Ergebnis ist in Abbildung 2.4 abgebildet. Bei der Berechnung von* G_4^* *haben nur* b *und* c *endliche Werte bekommen, da nur diese Zustände in einem Schritt das Ziel erreichen können. Für die Berechnung von* G_3^* *sind nur die Werte* $G_4^*(b) = 4$ *und* $G_4^*(c) = 1$ *notwendig, denn alle Pfade müssen über* b *oder* c *verlaufen, um* d *zu erreichen.*

Vorwärts Iterations Verfahren Beim Vorwärts Iterations Verfahren werden die Kosten von dem Startzustand x_i zu allen Zuständen berechnet. Dabei bezeichnet C_k^* die optimalen Kosten vom ersten bis zum k-ten Zustand. Um Pfade zu verhinden, die nicht an x_i beginnen, ist für C_1^* folgende Definition gegeben:

$$C_1^*(x_1) = w_l(x_1) \tag{2.9}$$

Dabei ist $w_l(x) = \begin{cases} 0 & \text{wenn } x = x_i \\ \infty & \text{sonst} \end{cases}$. Für die Zustände aus dem Intervall $[x_2, x_K]$ gilt:

$$C_k^*(x_k) = \min_{u_1,\ldots,u_k-1} \left\{ w(x_1) + \sum_{i=1}^{k-1} w(x_i, u_i) \right\} \tag{2.10}$$

$$C_k^*(x_k) = \min_{u_k^{-1} \in U^{-1}(x_k)} \left\{ C_{k-1}^*(x_{k-1}) + w(x_{k-1}, u_{k-1}) \right\} \tag{2.11}$$

Sei $x_{k-1} = f^{-1}(x_k, u_k^{-1})$ so ist $u_k^{-1} \in U(x_{k-1})$ die Aktion, die zu $u_k^{-1} \in U^{-1}(x_k)$ gehört. U^{-1} ist der Rückwärtsaktionsraum, der für ein $x' \in X$ als

$$U^{-1}(x') = \left\{ (x, u) \in U^{-1} \mid x' = f(x, u) \right\} \tag{2.12}$$

definiert ist. Sei f^{-1} eine Rückwärtszustandsübergangsfunktion, die x aus x' und $u^{-1} \in U^{-1}((x')$ berechnet wird, so ergibt sich die Rückwärtszustandsübergangsgleichung $x = f^{-1}(x', u^{-1})$. Im Zustandsgraphen bedeutet dieses einen Schritt entgegen der Richtung der Kante. In Gleichung 2.10 ist zu beachten, dass $u_i \in U(x_i)$ für alle $i \in [1, k-1]$ das aus $f(x_k, u_{k-1}$ resultierende x_k gleich dem x_k auf der linken Seite der Gleichung sein muss.

Die exakte Herleitung der Gleichungen ist [LaValle (2006), S.48ff] zu entnehmen.

Beispiel 2.3. *Vorwärts Iterations Verfahren*

	a	b	c	d	e
C_1^*	0	∞	∞	∞	∞
C_2^*	2	2	∞	∞	∞
C_3^*	4	4	3	6	∞
C_4^*	4	6	5	4	7
C_5^*	6	6	5	6	5

Abbildung 2.5: Die optimalen Restkosten, berechnet mit dem Vorwärts Iterations Verfahren (aus [LaValle (2006),S 50])

Wird in der Problemstellung aus Beispiel 2.2 mit dem Vorwärts Iterations Verfahren ein Pfad mit $K = 4$ berechnet, so ergeben sich die in Abbildung 2.5 abgebildeten Ergebisse. Die erste Zeile entsteht durch die Anwendung von w_l. Die zweite Zeile hat nur endliche Werte für a und b, da nur diese von a aus in einem Schritt erreichbar sind.

Pfade variabler Länge

Das Werte Iterations Verfahren für Pfade fester Länge kann einfach für Pfade variabler Länge umgewandelt werden. Dabei ist die Pfadlänge nicht mehr an K gebunden. Somit entsteht eine echte Verallgemeinerung des Problems 2.1, da beliebig lange Pfade erzeugt werden können. Dazu ist die Spezifizierung von K nicht mehr nötig, jedoch eine spezielle Aktion u_T.

Problem 2.3. *Optimale diskrete Planung*

1. *Alle Bestandteile aus Problem 2.1 werden direkt übernommen:* X, $U(x)$, f, x_i, X_G. *Zudem die Notation der Schritte aus Problem 2.2.*
2. *Ein Kostenfunktional* L *wird für den* K*-Schritt Plan* π_K *definiert. Es liefert die Kosten eines Pfades.*
$$L(\pi_K) = \sum_{k=1}^{K} w(x_k, u_k) + w_F(x_F) \tag{2.13}$$
Dabei ist im Gegensatz zum Problem 2.2 der Parameter K *nicht konstant, sondern gibt die Länge des Pfades* π_K *an. Der Definitionsbereich von* L *ist somit größer.*
3. *Jedes* $u(x)$ *enthält eine Endaktion* u_T. *Wird diese Aktion auf einen Zustand* x *angewandt, so kann sich der Zustand nicht mehr verändern. Es gilt:* $\forall i \geq k, u_i = u_T, x_i = x_k$ *und* $w(x, u_T) = 0$

Für die Berechnung von Pfaden mit variabler Länge ist es notwendig, eine Abbruchbedingung zu definieren, nach der die Suche abgebrochen wird. Angenommen, ein Pfad mit $K = 4$ wird berechnet, es exsistiert aber schon eine Lösung nach zwei Schritten (u_1, u_2), die X_G von x_i aus erreicht. So ist dieser Plan gleich mit (u_1, u_2, u_T, u_T), da die Endaktion weder den Zustand noch die Kosten beeinflusst. Aufgrund der Unabhängigkeit von K gibt es keinen Grund, die Rückwärts Iteration unendlich lange fortzusetzen ($G_0^*, G_{-1}^*, \ldots$),es sei denn, dass X endlich ist. Es gibt einen Punkt, an dem genug Iterationsschritte vollzogen sind. Von diesem Schritt k an, nehmen die Werte der Restkosten feste Werte ein, dass heißt: $\forall x \in X, \forall i \leq k, G_{i-1}^*(x) = G_i^*$. Die Restkosten sind also nicht mehr von der Anzahl der Schritte abhängig. Somit kann der Index k entfernt werden:

$$G^*(x) = \min_u \{w(x, u) + G^*(f(x, u))\} \tag{2.14}$$

Dieses gilt immer, solange $w(x, u) \geq 0$.

Beispiel 2.4. *Werte Iterations Verfahren für Pfade variabler Länge*

	a	b	c	d	e
G_0^*	∞	∞	∞	0	∞
G_{-1}^*	∞	4	1	0	∞
G_{-2}^*	6	2	1	0	∞
G_{-3}^*	4	2	1	0	∞
G_{-4}^*	4	2	1	0	∞
G^*	4	2	1	0	∞

(a) Restkosten bestimmt durch Rückwärtsiteration

	a	b	c	d	e
C_1^*	∞	0	∞	∞	∞
C_2^*	∞	0	1	4	∞
C_3^*	2	0	1	2	5
C_4^*	2	0	1	2	3
C^*	2	0	1	2	3

(b) Kosten bestimmt durch Vorwärtsiteration

Abbildung 2.6: Die optimalen (Rest)-kosten, berechnet mit dem Iterations Verfahren (aus [LaValle (2006), S 54f])

Für das Beisiel wird erneut die Problemstellung aus Beispiel 2.2 hergenommen. Die maximale Länge der Pfade ist nicht mehr vorgegeben. Abbildung 2.6(a) zeigt die Restkosten von jedem Zustand zu dem Zielzustand $x_G = d$. Nach wenigen Iterationsschritten verändern sich die Kosten nicht mehr. Ab diesem Schritt werden die Pfade nur noch mit der Endaktion aufgefüllt und die Suche kann abgebrochen werden. Die resultierenden Restkosten sind als G^* definiert. Da d nicht von e aus erreichbar ist, gilt: $G^*(e) = \infty$. Mit der Rückwärtsiteration ist Abbildung 2.6(b) berechnet worden. Hier sind die Kosten ausgehend vom Startknoten $x_i = b$ aufgetragen.

Vergleich: Vorwärtsiteration - Dijkstra Es wurden zwei unterschieliche Ansätze der dynamischen Programmierung vorgestellt. Das Werte Iterations Verfahren in Kapitel 2.2.3, bei dem wiederholt die Berechnungen über den Zustandsraum ausgeführt werden. Beim Dijkstra Algorithmus 2.2.2 wird jeder Zustand nur einmal besucht, jedoch mit dem Nachteil, dass alle noch *lebenden* Zustände gepeichert werden müssen. Die Parallelen zwischen den beiden Ansätzen sind einfach zu erkennen. Die Zustände, deren Kosten unendlich und die somit noch nicht besucht sind, sind die *unbesuchten* Zustände. Zustände, deren optimale Kosten berechnet wurden, sind *tot*. Alle anderen Zustände, für die bereits Kosten vorliegen, die jedoch noch nicht optimal sind, sind die *lebenden* Zustände. Diese Zustände befinden sich in der Liste Q. Obwohl sich die beiden Algorithmen sehr ähnlich und viele Probleme der Werte Iteration mit dem Dijkstra Algorithmus lösbar sind, gibt es Fälle, in denen die Liste Q des Dijkstra Algorithmus zu schwer zu verwalten ist. Das Werte Iterations Verfahren kann auf deutlich mehr Probleme angewandt werden.

Der Dijkstra Algorithmus gehört zu der Familie der *Label aktualisierenden Algorithmen*, die die allgemeine Vorwärtssuche aus Algorithmus 2.1 erweitert.

Algorithm 2.2 Vorwärtssuche mit Aktualisierungen, die eine Verallgemeinerung des Dijkstr Algorithmus darstellt.

```
1:  Set C(x) = ∞ for all ∀x ∈ X \ {x_i}, and set C(x_i) = 0
2:  Q.insert(x_i)
3:  while Q ≠ ∅ do
4:      x ← Q.getFirst()
5:      for all u ∈ U(x) do
6:          x' = f(x, u)
7:          if C(x) + w(x, u) < min{C(x'), C(x_G)} then
8:              C(x') ← C(x) + w(x, u)
9:              if x' ≠ x_G then
10:                 Q.insert(x')
11:             end if
12:         end if
13:     end for
14: end while
```

Algorithmus 2.2 zeigt die Erweiterung. Der wichtigste Unterschied liegt in den Zeilen 7 bis 12. In diesen wird getestet, ob der neue Pfad zu dem Zustand x' besser ist als ein evtl. bereits vorhandener. Sind die neuen Kosten höher als die zu dem Zielzustand, so wird der neu gefundene Zustand ignoriert. Im Algorithmus 2.1 ist dieses in Zeile 13 zu finden.

2.3 Geometrische Repäsentation

In diesem Kapitel wird beschrieben, wie die Welt, in der sich Roboter bewegen können, mit einfachen Mitteln dargestellt werden kann. Dazu wird eine Welt $\mathcal{W}$ definiert, die in einer 2D-Welt $\mathcal{W} = \mathbb{R}^2$ und in einer 3D-Welt $\mathcal{W} = \mathbb{R}^3$ ist. Im Folgenden soll jedoch nur eine 2D-Welt betrachet werden. Es gibt in der Welt zwei verschiedene Arten von Einheiten:

Hindernisse: Teile in der Welt, die nicht belegt sind.

Roboter: Körper die mittels eines Bewegungsplans bewegt werden können.

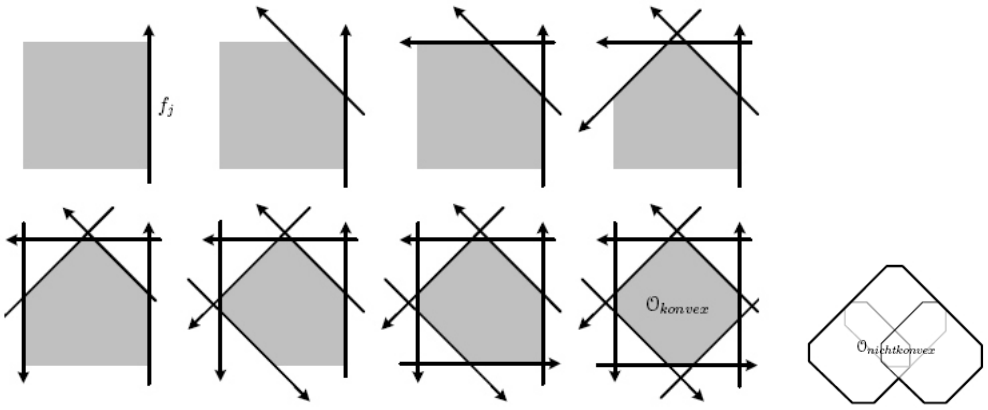
Für beide Arten wird eine Repräsentation benötigt. Grundsätzlich sind Hindernisse und Roboter jeweils Teilmengen von $\mathcal{W}$. Die *Hindernisregion* wird als die Menge $\mathcal{O} \subseteq \mathcal{W}$ definiert, die alle Punkte enthält, die in einem Hindernis oder mehreren Hindernissen liegen. Es ist nun eine systematische und aussagekräftige Repräsentation notwendig, die effektiv berechnet werden kann. Roboter werden auf dieselbe Weise repräsentiert.

2.3.1 Polygonmodelle

Eine einfache Möglichkeit ist es, Hindernisse (und Roboter) durch Polygone zu beschreiben (vgl. [LaValle (2006), S.82]). Ein Polygon ist durch Primitive H_i repräsentiert, die sich einfach mittels Mengenoperationen verknüpfen und berechnen lassen. Jedes Primitiv repräsentiert eine Teilmenge der Welt $\mathcal{W}$. Eines der einfachsten Primitiven ist dabei die *Halbebene*, die als

$$H_i = \{(x, y) \in \mathcal{W} \mid f_i(x, y) \leq 0\} \tag{2.15}$$

definiert ist, wobei $f(x, y)$ eine Geradengleichung ist und H_i die Menge alle Punkte beschreibt, die sich auf der Halbebene befinden. Ein *konvexes* Polygon kann als Schnittmenge



(a) konvexes Polygon, konstruiert aus acht Halbebenen (aus LaValle (2006), S.83)

(b) nichtkonvexes Polygon, konstruiert aus drei convexen Polygonen (aus Schubert (2006), S.16)

Abbildung 2.7: Beispiele für die Konstruktion von Polygonen

mehrerer Halbebenen beschrieben werden:

$$\mathcal{O} = H_1 \cap H_2 \cap \ldots \cap H_m \tag{2.16}$$

Die Konstruktion eines konvexen Polygons aus acht Halbebenen ist in Abbildung 2.7(a) dargestellt. Nichtkonvexe Polygone lassen sich als Vereinigung von mehreren konvexen Polygonen darstellen:

$$\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \ldots \cup \mathcal{O}_n \tag{2.17}$$

Dabei müssen zwei $\mathcal{O}_i, \mathcal{O}_j, i \neq j$ nicht disjunkt sein. Dieses ist anhand von drei Polygonen in Abbildung 2.7(b) dargestellt.

Um komplexere Hindernisse durch Primitive beschreiben zu können, kann anstatt einer Trennlinie auch jede andere nicht lineare Funktion $f(x, y) = 0$ genutzt werden. In diesem Fall wird von einer *semi-algebraischen* Darstellung gesprochen.

2.3.2 Kollisionsprüfung

Mithilfe der Repräsentation der Hindernisse als Polygone und der Darstellung der Polygone als Primitive, lässt sich ein einfaches logisches Prädikat ϕ entwickeln, welches als $\phi : \mathcal{W} \rightarrow \{TRUE, FALSE\}$ definiert ist. Dieses liefert $TRUE$ für die Punkte aus $\mathcal{W}$, die in $\mathcal{O}$ liegen, anderenfalls $FALSE$. Für eine gegebene Line $f(x, y) = 0$ liefert $e(x, y)$ $TRUE$, wenn $f(x, y) \leq 0$, ansonsten $FALSE$. Für ein konvexes Polygon ist mit

$$\alpha(x, y) = \bigwedge_{i=1}^{m} e_i(x, y) \tag{2.18}$$

bestimmbar, ob ein Punkt innerhalb des Polygons liegt. Besteht die $\mathcal{O}$ aus n konvexen Polygonen, so lautet die Formel für die Kollisionsprüfung:

$$\phi(x, y) = \bigvee_{i=1}^{n} \alpha_i(x, y) \tag{2.19}$$

Die Rechenzeit ist jediglich von der Anzahl der Hindernissprimitive linear abhängig.

2.4 Konfigurationsraum

Die Idee des Konfigurationsraumes ist es, das eigentliche Pfadplanungproblem zu abstrahieren und dieses somit allgemeiner lösbar zu machen. Dazu wird ein Raum konstruiert, in dem der Roboter nicht mehr als ausgedehntes Objekt, sondern als Punkt angenommen werden kann. Die Hindernisse werden in geeigneter Weise in diesen Raum transformiert. Die Pfadplanung reduziert sich somit auf die Planung eines Pfades für einen Punkt.

2.4.1 Definition für einen einfachen Roboter

Für eine starren Roboter, der keine beweglichen Teile, wie Greifarme, hat, gilt folgende Definition:

Definition 2.2. *Starrer Roboter*
Sei A ein starrer Roboter, der sich in einem physikalischen Arbeitsraum (der Welt) $\mathcal{W}$ bewegt. So ist $\mathcal{W}$ ein euklidischer Raum vom Typ $\mathbb{R}^2$ oder $\mathbb{R}^3$ mit einem festen Koordinatensystem, das mit $\mathcal{F}_{\mathcal{W}}$ bezeichnet wird. Der Roboter A besitzt ebenfalls ein Koordinatensystem $\mathcal{F}_{\mathcal{A}}$, so dass jeder Punkt des Roboters einen festen Punkt $\mathcal{F}_{\mathcal{A}}$ hat. Die Ursprünge von $\mathcal{F}_{\mathcal{W}}$ und $\mathcal{F}_{\mathcal{A}}$ werden als O_W und O_A bezeichnet. O_A ist der Referenzpunkt *des Roboters A. Es wird davon ausgegangen, dass A bezüglich des Referenzpunktes nicht symmetrisch und somit kein einzelner Punkt ist.*

Genau genommen ist $\mathcal{W}$ die *physikalische Umgebung* und $\mathbb{R}^n$ mit $n \in \mathcal{N}$ das mathematische Abbild.

Für einen starren Roboter, wie er beschrieben wurde, wird der Konfigurationsraum, wie folgt definiert:

Definition 2.3. *Konfigurationsraum*
Ein starrer Roboter aus Definition 2.2 kann sich auf der Ebene bewegen, somit kann die Translation durch $x_t, y_t \in \mathbb{R}$ beschrieben werden. Dieses ergibt die Mannigfaltigkeit $M_1 = \mathbb{R}^2$. Unabhängig von der Position kann der Roboter jede Orientierung $\theta \in [0, 2\pi]$ einnehmen. Dabei ist die Orientierung 0 gleich der Orientierung 2π, was der Mannigfaltigkeit $M_2 = \mathbb{S}^1$ entspricht. Der somit entstehende Konfigurationsraum C ist somit: $C = M_1 \times M_2 = \mathbb{R}^2 \times \mathbb{S}^1$.

Da $\mathbb{S}^1$ mehrfach verbunden ist, ist auch $\mathbb{R}^2 \times \mathbb{S}^1$ mehrfach verbunden. Eine Visualisierung ist jedoch nicht möglich, da es sich um eine 3D- Mannigfaltigkeit handelt. Mit der Identifikation kann C jedoch wie folgt interpretiert werden. Man nimmt einen offenen Kubus $(0, 1)^3 \in \mathbb{R}^3$ und fügt die Randpunkte $(x, y, 0)$ und $(x, y, 1)$ mit der Identifikation $(x, y, 0) \sim (x, y, 1)$ für alle $(x, y) \in \mathbb{R}^2$ hinzu. Bewegungen in x- und y-Richung sind durch das Intervall beschränkt. Die Bewegung in z-Richtung jedoch nicht, hierbei handelt es sich um einen Umlauf.

2.4.2 Hindernisse im Konfigurationsraum

In Kapitel 2.4.1 wurde der Konfigurationsraum C für einen einfachen starren Roboter definiert. Dieser enthält keine Hindernisse. In diesem Kapitel wird C in den Hindernisraum und den freien Konfigurationsraum aufgeteilt. Dabei beschreibt der Hindernisraum die Konfigurationen, die zu einer Kollision führen. Der überbleibende freie Konfigurationsraum ist der Raum, in dem sich ein Roboter bewegen kann.

Definition 2.4. *Hindernisraum*
Für einen starren Roboter A sei die Welt $\mathcal{W} = \mathbb{R}^2$ und die Hindernisregion $\mathcal{O} \in \mathcal{W}$. A und $\mathcal{O}$ sind, wie in Kapitel 2.3, als Polygone modelliert. $q \in C$ wird als Konfiguration *von A bezeichnet. Dabei ist $q = (x, y, \theta)$. Der Hindernisraum $C_{obs} \subseteq C$ ist als*

$$C_{obs} = \{q \in C \mid A(q) \cap \mathcal{O} \neq \emptyset\} \tag{2.20}$$

definiert. C_{obs} enthält also alle Konfigurationen q, bei denen es zwischen dem transformierten Roboter $A(q)$ und der Hindernisregion zu Kollisionen kommt. Der restliche Raum wird als freier Konfigurationsraum $C_{free} = C \setminus C_{obs}$ bezeichnet.

Explizite Modellierung von C_{obs} bei aussschließlicher Translation

Die Modellierung von C_{obs} ist keine triviale Aufgabe. Anhand eines Spezialfalles kann jedoch der grundsätzliche Aufbau beschrieben werden. Sei $C = \mathbb{R}^n$ mit $n = 1, 2$ oder 3 und A ein Roboter, dessen Bewegungen auf Translationen beschränkt sind, so lässt sich C_{obs} durch eine Faltungsoperation bestimmen. Für zwei beliebige Mengen X, Y sei die *Minkowski-Differenz* als

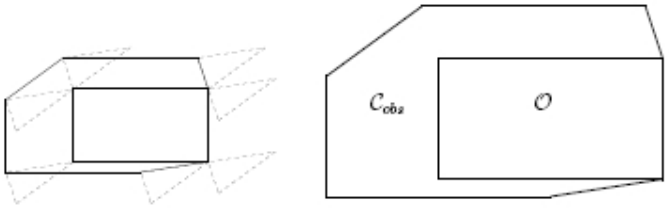
$$X \ominus Y = \{x - y \in \mathbb{R}^n \mid x \in X \wedge y \in Y\} \tag{2.21}$$

definiert, wobei $x - y$ die Vektorsubtraktion in $\mathbb{R}^n$ ist. Mit der Minkowski-Differenz kann der Hindernisraum als $C_{obs} = \mathcal{O} \ominus A(0)$ definiert werden.

Beispiel 2.5. *Minkowski-Differenz*

Für ein Beispiel zur Modellierung von C_{obs} bei ausschließlicher Translation wird A als ein dreieckiger Roboter (Abb. 2.8(a)) und $\mathcal{O}$ als ein Rechteck angenommen (Abb. 2.8(b)). Der schwarze Punkt in Abbildung 2.8(a) bezeichnet die Position von O_A, dem Urspung des Roboterkoordinatensystems. Für die Berechnung von C_{obs} wird A um $\mathcal{O}$ so herumgeschoben, dass immer ein Kontakt besteht. O_A beschreibt dabei den Umriss von C_{obs} (Abbildung 2.8(d)).

(a) dreieckiger Roboter (b) rechteckiges Hinderniss



(c) um den Roboter erweitertes Hinderniss (d) Resultierender C_{obs}

Abbildung 2.8: Beispiel für die Modellierung von C_{obs} bei ausschließlicher Translation $C_{obs} = \mathcal{O} \ominus A(0)$

Explizite Modellierung von C_{obs}

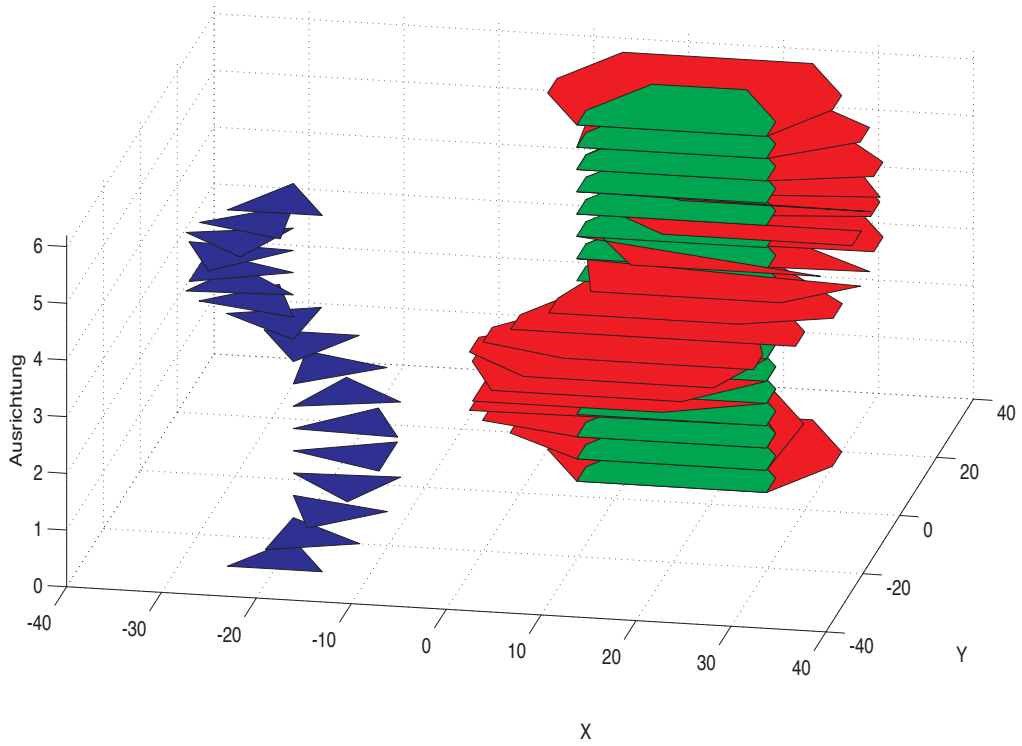


Abbildung 2.9: Ein Fahrzeug (blau), ein Hindernis (grün) und der Hindernisraum C_{obs} (blau) für die Fahrzeugausrichtungen $\theta \in [0, 2\pi]$.

In Kapitel 2.4.2 ist beschrieben, wie Hindernisse in den Konfigurationsraum, bei ausschließlicher Translation des Fahrzeuges, transformiert werden. Jedoch ist dieses nur selten der Fall. Die meisten Fahrzeuge können ihre Ausrichtung ändern, bzw. sind nicht kreisförmig, sodass eine Änderung der Ausrichtung eine Änderung von C_{obs} hervorruft.

Abbildung 2.9 zeigt den Hindernisraum für eine komplette Drehung eines dreieckigen Fahrzeuges. Berechnet wird dieser mit der Minkowski-Differenz aus Kapitel 2.4.2. Die Berechnung kann bei jedem Kollisionstest vorgenommen werden. Performanter ist es jedoch C_{obs} vorauszuberechnen und für Winkel, die zwischen den vorausberechneten Winkeln liegen, die Polynome zu interpolieren.

2.4.3 Verallgemeinerte Problemstellung Pfadsuche

Durch die Einführung des Konfigurationsraumes und der Modellierung von Hindernissen ist es nun möglich, das Pfadplanungsproblem als Problem 2.1 allgemeiner zu formulieren:

Problem 2.4. *Das* Piano Mover's *Problem*

1. *Es ist eine Welt $\mathcal{W}$ gegeben, hier $\mathcal{W} = \mathbb{R}^2$.*
2. *Es gibt eine semi-algebraische Hindernisregion $\mathcal{O} \subseteq \mathcal{W}$ in der Welt.*
3. *Es ist ein semi-algebraischer Roboter A in $\mathcal{W}$ definiert.*
4. *Der Konfigurationsraum C ist gegeben und aus diesem C_{free} und C_{obs} abgeleitet.*
5. *Eine Konfiguration $q_i \in C_{free}$ ist als Startkonfiguration festgelegt.*
6. *Eine Konfiguration $q_g \in C_{free}$ ist als Zielkonfiguration festgelegt.*
7. *Ein kompletter Algorithmus muss einen (kontinuierlichen) Pfad $\tau : [0,1] \rightarrow C_{free}$ berechnen, so dass $\tau(0) = q_i$ und $\tau(1) = q_g$ ist. Oder angibt, dass kein Pfad exisitiert.*

2.5 Ansätze für die Pfadplanung

Es gibt eine Vielzahl von Ansätzen für die Pfadplanung. Nach Latombe [Latombe (1991)] lassen sich diese alle in eine der drei folgenden Kategorien einteilen:

Wegkarten Ansätze: Der freie Konfigurationsraum C_{free} wird durch eindimensionale Kurven unterteilt. Diese Kurven beschreiben Pfade im Arbeitsraum und bilden einen Graphen. Die Pfadsuche ist auf die Verbindung der Start- und Zielkonfiguration im Graphen beschränkt.

Zellunterteilungs Ansätze: Der freie Konfigurationsraum C_{free} wird durch eine Menge von sich nicht überschneidenen Zellen zerlegt. Die Vereinigung aller Zellen ergibt genau C_{free}. Ein Verbindungspgraph wird erzeugt. In dem Verbindungsgraphen wird nach einem Pfad gesucht, der die Zellen verbindet, in denen die Start- und Zielkonfiguration befinden.

Potentialfeld Ansätze: Ein Roboter wird als Partikel in einem künstlichen Potentialfeld modelliert. Das Potentialfeld besteht aus einem anziehenden Potential, dass den Roboter zum Ziel zieht, und abstoßenden Potentialen, die den Roboter von Hindernissen fern hält. Die Summe aller Potentiale ergibt die Potentialfeldfunktion. Der Weg zum Ziel ist der negative Gradient der Potentialfeldfunktion.

Im folgenden sind Ansätze aus allen drei Kategorien beschrieben.

2.5.1 Sampling-Based Ansätze

Bei sampling based Ansätzen wird C_{obs} nicht explizit modelliert, da diese Berechnung mit großen Rechenaufwand verbunden ist. Im Gegensatz dazu verwenden sampling based Ansätze die Kollisionserkennung als *black box*. Die Art der Hindernisse und deren Modellierung sind somit nicht mehr relevant für diesen Ansatz. Die Grundidee besteht darin, den Konfigurationsraum abzutasten und aus den abgetasteten Punkten einen Suchgraphen zu erstellen. Dabei wird für jeden Abtastpunkt und den Pfad zu diesem getestet, ob dieser in C_{free} ist. Die meisten sampling based Ansätze folgen dem Schema:

Ansatz 2.1. *Sampling-Based Ansatz*

1. ***Initialisierung:*** *Sei $\mathcal{G}(V, E)$ ein ungerichteter* Suchgraph*, der mindestens einen Knoten, aber keine Kante enthält. Typischerweise sind q_i, q_g oder beide in V enthalten. Es können aber auch noch andere Punkte aus C_{free} enthalten sein.*
2. ***Knotenauswahl Strategie:*** *Ein Knoten $q_{curr} \in V$ wird gewählt.*

3. ***Lokale Pfadplanung:*** *Für eine beliebige Konfiguration $q_{new} \in C_{free}$, konstruiere eine Bahn $\tau_s : [0,1] \rightarrow C_{free}$, so dass $\tau_s(0) = q_{curr} \wedge \tau_s(1) = q_{new}$. Ist τ_s nicht hindernisfrei oder kann nicht generiert werden, gehe zu Schritt 2.*
4. ***Erweiterung des Graphen:*** *Füge τ_s als neue Kante von q_{curr} nach q_{new} zu E hinzu. Ist $q_{new} \notin V$ füge sie hinzu.*
5. ***Ergebnissprüfung:*** *Prüfe, ob $\mathcal{G}$ eine Lösung für das Pfadplanungsproblem enthält. Wenn dieses der Fall ist, verlasse den Algorithmus mit einer Erfolgsmeldung.*
6. ***Springe zu Schritt 2:*** *Iteriere weiter bis eine Lösung gefunden wurde oder eine Abbruchbedingung erfüllt ist. In diesem Fall verlasse den Algorithmus mit einem Fehler.*

Der Ansatz 2.1 ähnelt im Groben dem Problem 2.1. Der Unterschied liegt in der Zeile 3, anstatt von Aktionen u repräsentieren die Kanten Pfadsegmente τ_s. Zudem handelt es sich um einen ungerichteten Graphen. Die *lokale Pfadplanung* versucht ein möglichst einfaches Pfadsegment zwischen zwei Konfigurationen zu erzeugen. Dieser Schritt heißt *lokal*, da er nicht das gesamte Bahnplanungsproblem löst. Es wird davon ausgegangen, dass dieser Schritt sehr häufig fehlschlägt.

Sampling based Ansätze unterscheiden sich im wesentlichen in der Wahl eines nächsten Knoten und der lokalen Pfadplanung. Im weiteren werden zwei Verfahren vorgestellt, die sich in der Wahl der nächsten Knoten unterscheiden. Als lokaler Pfad wird die direkte Strecke zwischen den Konfigurationen genommen. Eine genauere Beschreibung der lokalen Pfadplanung wird unter der Berücksichtigung von mechanischen Zwangsbedingungen in Kapitel 2.8 vorgenommen.

Gitterbasierter Ansatz

Der einfachste Ansatz eines sampling based Ansatzes ist es, ein Gitter über C zu legen und eine diskrete Pfadsuche aus Kapitel 2.2.3 anzuwenden. Das daraus resultierende Planungsproblem ähnelt stark dem aus Beispiel 2.1. Die Kanten sind nun Pfade in C_{free}. Einige Kanten werden nicht vorhanden sein, da die Pfade Kollisionen aufweisen. Die Kanten werden während der Suche erzeugt, da eine explizite Konstruktion von C_{obs} zu aufwändig ist. Die Einführung eines Gitters ist eine Art der Diskretisierung. C wird in jede Dimension durch eine Auflösung $k_1, k_2, \ldots, k_n$, wobei k_i eine ganze positive Zahl ist, aufgeteilt. Somit kann die Auflösung für jede Dimension variieren. Es können nun die Nachbarschaften von q definiert werden. Dabei wird die 1-Nachbarschaft als

$$N_1(q) = \{q + \Delta q_1, \ldots q + \Delta q_n, q - \Delta q_1, \ldots, q - \Delta q_n\} \tag{2.22}$$

definiert. Dabei ist $\Delta q = [0 \ldots 1/k_i 0 \ldots 0]$, ein Vektor, in dem die ersten $i-1$ Elemente als auch die letzten $n-i$ Elemente 0 sind. Für einen n-Dimensionalen C gibt somit maximal $2n$ 1-Nachbarschaften. Für das Beispiel 2.1 würde dieses Bewegungen nach *unten, oben, links* und *rechts* entsprechen.

Bei diesem Ansatz ist zu beachten, dass die Auflösungen einen direkten Einfluss auf die Laufzeit und die Lösbarkeit des Problems haben. Ist die Auflösung zu hoch gewählt, so müssen sehr viele Konfigurationen besucht werden,um eine Lösung zu finden. Ist die Auflösung jedoch zu gering gewählt, so können vorhandene Pfade unentdeckt bleiben und somit das grundliegende Problem der Pfadsuche nicht gelöst werden.

Zufallsbasierter Ansatz

Dieser Ansatz ist frei von Parametern, die die Qualität oder die Rechenzeit des Algorithmus verändern. Mittels einer zufälligen Sequenz von Konfigurationen wird ein Suchbaum erstellt. Dieser Suchbaum wird als *rapidly exploring random tree (RRT)* bezeichnet. Je mehr Konfigurationen in dem Baum vorhanden sind desto höher ist die Dichte, daher wird dieser auch (unabhängig von der Knotenauswahl Strategie) als *rapidly exploring dense tree (RDT)* bezeichnet. Dabei ist wichtig, dass die Konfigurationen den gesamten Suchraum erfassen.

Definition 2.5. *Rapidly exploring dense tree*
Ein RDT ist ein topologischer Graph $\mathcal{G}(V, E)$. $S \subset C_{free}$ sind alle Punkte die in $\mathcal{G}$ erreicht werden können. Da alle $e \in E$ Pfade sind, kann S als

$$S = \bigcup_{e \in E} e([0,1]) \tag{2.23}$$

definiert werden. Dabei ist $e([0,1])$ das Bild des Pfades e.

Algorithm 2.3 Einfacher rapidly exploring dense tree(q_i)

```
G.insert_vertex(q_i)
for i = 1 to k do
    G.insert_vertex(α(i))
    q_n ← NEAREST(S, α(i))
    G.insert_edge((q_n, α(i)))
end for
```

In jedem Iterationsschritt des Algorithmus wird zufällig eine neue Konfiguration $\alpha(i)$ erzeugt. Anschließend wir der dichteste Punkt $q_n \in S$ zu $\alpha(i)$ gesucht. Ist q_n ein Knoten so wird q_n und die Kante $(q_n, \alpha(i))$ zu $\mathcal{G}$ hinzugefügt. Liegt q_n auf einer Kante, so wird die Kante an

Algorithm 2.4 Rapidly exploring dense tree(q_i) mit Hindernissen

```
G.insert_vertex(q_i)
for i = 1 to k do
    q_n ← NEAREST(S, α(i))
    q_s ← STOPPING_CONF(q_n, α(i))
    G.insert_vertex(q_s)
    G.insert_edge((q_n, q_s))
end for
```

q_n aufgeteilt und anschließend die Kante $(q_n, \alpha(i))$ eingefügt. In Algorithmus 2.3 und 2.4 geschieht dieses in der Methode $NEAREST$.

Bisher wurden keine Hindernisse betrachtet. Algorithmus 2.4 erweitert 2.3 um die Vermeidung von Hindernissen. Um keine Pfade in $\mathcal{G}$ aufzunehmen, die nicht in C_{free} sind, wird in der Methode $STOPPING\_CONF$ die Konfiguration berechnet, die zwischen q_n und $\alpha(i)$ und $q_s \notin C_{obs}$ ist. Dabei muss sich q_s auf der zu q_n zugewandten Seite des Hindernisses befinden. Befindet sich kein Hinderniss auf dem Pfad von q_n zu $\alpha(i)$, so gilt $q_s = q_n$.

Eine Lösung für die Pfadsuche ist gefunden, wenn sich q_g in dem Baum befindet. Hierzu kann zum Beispiel jedes 100ste $\alpha(i) = q_g$ sein. Wird ein Pfad gefunden, so ist q_i mit q_g verbunden und somit eine Lösung vorhanden.

2.5.2 Kombinatorische Ansätze

Im Gegensatz zu *sampling based* Ansätzen handelt es sich bei kombinatorischen Ansätzen um *exakte* und *vollständige* Algorithmen, das heißt, dass sie für jedes Problem eine Lösung finden, oder korrekt melden, wenn keine Lösung vorhanden ist. Jedoch sind die Ansätze häufig abhängig von der Modellierung von C_{obs}. So können einige Algorithmen nicht mehr verwendet werden, wenn sich die Repräsentation von C_{obs} verändert. Aus dem Konfigurationsraum C wird ein Streckennetz extrahiert, welches wie folgt definiert ist:

Definition 2.6. *Streckennetz (Roadmap)*
Sei $\mathcal{G}$ ein topologischer Graph, welcher C_{free} abbildet. Zudem sei $S \subset C_{free}$ die Menge aller Punkte die von $\mathcal{G}$ erreichbar sind (vgl. Gleichung 2.23). Der Graph $\mathcal{G}$ wird als Streckennetz bezeichnet, wenn für diesen folgende Bedingungen gelten:

1. Erreichbarkeit: *Für alle $q \in C_{free}$ ist ein einfacher Pfad $\tau : [0, 1] \to C_{free}$ mit $\tau(0) = q \wedge \tau(1) = s$ auf einfache Weise berechenbar. Dabei ist $s \in S$ und in der Regel der dichteste Punkt zu q, wenn C ein metrischer Raum ist.*

2. Konnektivitätserhaltung: *Unter der ersten Bedingung kann jedes q_i mit einem s_1 und jedes q_g mit einem s_2 verbunden werden. Wenn es einen Pfad $\tau : [0,1] \rightarrow C_{free}$ mit $\tau(0) = q_i \wedge \tau(1) = q_g$ gibt, so existiert auch ein Pfad $\tau' : [0,1] \rightarrow S$ mit $\tau'(0) = s_1 \wedge \tau'(1) = s_2$.*

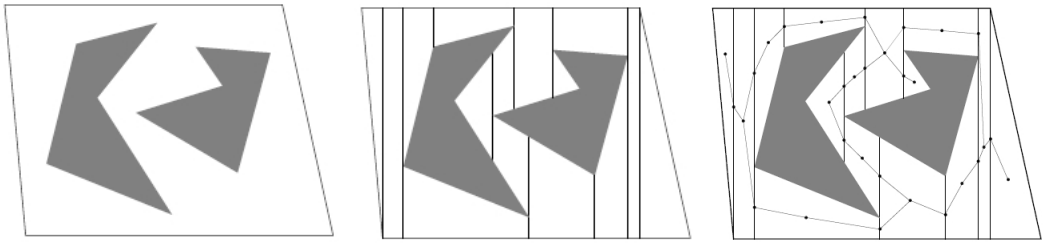
Die Definition 2.6 garantiert die *Vollständigkeit* der vorgestellten Algorithmen. Die erste Bedingung besagt, dass jede Suchanfrage in $\mathcal{G}$ gelöst werden kann. Die zweite Bedingung besagt, dass immer eine Lösung gefunden wird, wenn eine existiert.

Vertikale Zellzerlegung

Bei der Zellzerlegung wird C_{free} in eine endliche Menge von *Zellen* zerlegt. Dabei wird als k-Zelle eine k-dimensionale Zelle bezeichnet. Die Zellzerlegung muss dabei drei Bedingungen erfüllen.

1. Die Berechnung eines Pfades innerhalb einer Zelle muss trivial sein.
2. Nachbarschaftsinformationen für die Zellen müssen für die Erstellung des Streckennetztes verfügbar sein.
3. Für q_i, q_g soll einfach ermittelbar sein, in welcher Zelle sich diese befinden.

Wenn eine Zellzerlegung diese drei Bedingungen erfüllt, so beschränkt sich die Pfadsuche auf eine Suche in einem Graphen.



(a) C (b) C mit vertikaler Zerlegung (c) Das Streckennetz

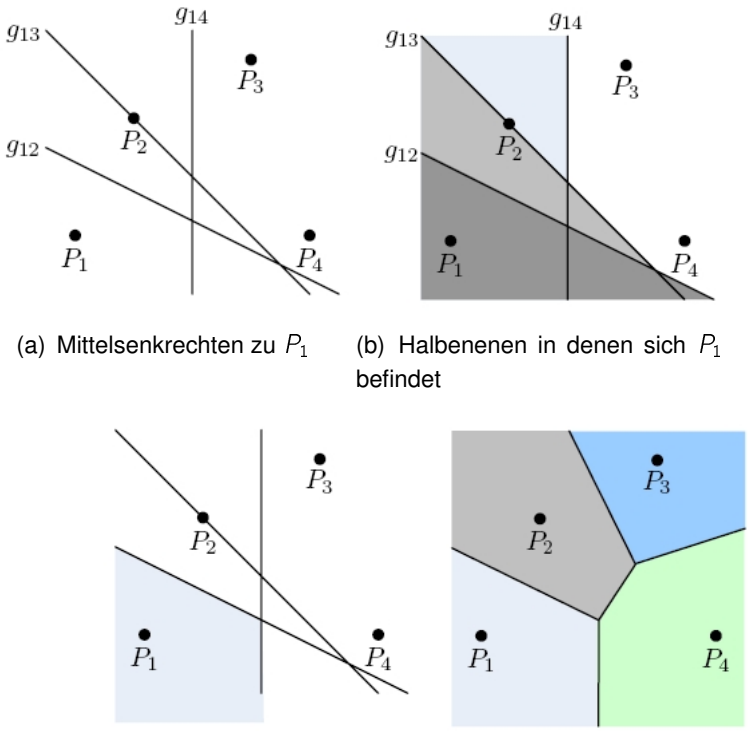
Abbildung 2.10: Zerlegung von C_{free} durch die vertikale Zellzerlegung (aus [LaValle (2006), S.255f])

Die vertikale Zellzerlegung partitioniert C_{free} in eine endliche Anzahl von 1-Zellen und 2-Zellen. 2-Zellen sind Trapeze mit vertikalen Seiten oder Dreiecken. Die Zerlegung ist wie folgendermaßen definiert: Sei P die Menge aller Knoten, die für die Beschreibung von C_{obs} nötig sind. Von allen $p \in P$ werden je zwei Strahlen konstruiert (Abbildung 2.10(b)). Einen nach oben, den anderen nach unten. Diese Strahlen werden nur durch Hindernisse oder

die Grenzen von C begrenzt. C_{free} wird durch diese Strahlen zerteilt. Das Zwischenraum zwischen den Strahlen sind die 2-Zellen. Die Strahlen selbst sind die 1-Zellen und dienen als Begrenzung zwischen zwei 2-Zellen.

Aus dem zerlegten C_{free} lässt sich ein Streckennetz generieren. Für jede Zelle C_i wird ein Abtastpunkt (sample point) q_i definiert, so dass $q_i \in C_i$. Als Abtastpunkt kann das Zentrum der Zelle verwendet werden, andere Punkte sind jedoch auch vorstellbar. Das Streckennetzt wird wie folgt erzeugt: Sei $\mathcal{G}(V, E)$ ein topologischer Graph. Für alle Zellen C_i wird ein Knoten $q_i \in C_i$ definiert. Dieser existiert für alle 1-Zellen und 2-Zellen. Für alle 2-Zellen werden Kanten erzeugt, von ihrem Abtastpunkt zu den Abtastpunkten auf den, die Zelle begrenzenden, 1-Zellen (Abbildung 2.10(c)).

Verallgemeinertes Voronoi- Diagramm



(a) Mittelsenkrechten zu P_1

(b) Halbenenen in denen sich P_1 befindet

(c) Schitt der Halbmengen ergibt $Vor(P_1)$

(d) Voronoizellen aller Punkte

Abbildung 2.11: Erstellung des Vornoi- Diagramms (aus [Schubert (2006), S.35])

Eine andere Zerlegung bietet das Voronoi- Diagramm. Dabei werden Strecken generiert, die jeweils die maximal mögliche Distanz zu den Hindernissen haben. Häufig wird dieses

Verfahren genutzt, wenn die Lokalisierung des Fahrzeuges ungenau ist und das Fahrzeug somit möglichst viel Abstand zu den Hindernissen halten soll. Das entstehende Streckennetz wird auch als *maximum-clearance roadmap* oder *reaction method* bezeichnet.

Es handelt sich dabei um eine Generalisierung des Voronoi- Diagrammes, da dieses nicht anhand von Punkten sondern mit Punkten und Polygonen erzeugt wird. Jeder Punkt auf einer Kante im Streckennetz ist von zwei Rändern eines Hindernisses in C_{obs} gleichweit entfernt. Jeder Knoten im Streckennetz entspricht einer Kreuzung von zwei oder mehreren Kanten und ist somit vom mindestens drei Hindernissgrenzen gleichweit entfernt. In Abbildung 2.11 ist die Erstellung des Voronoi- Diagramms für Punkte abgebildet.

Kürzeste Pfade Streckennetz

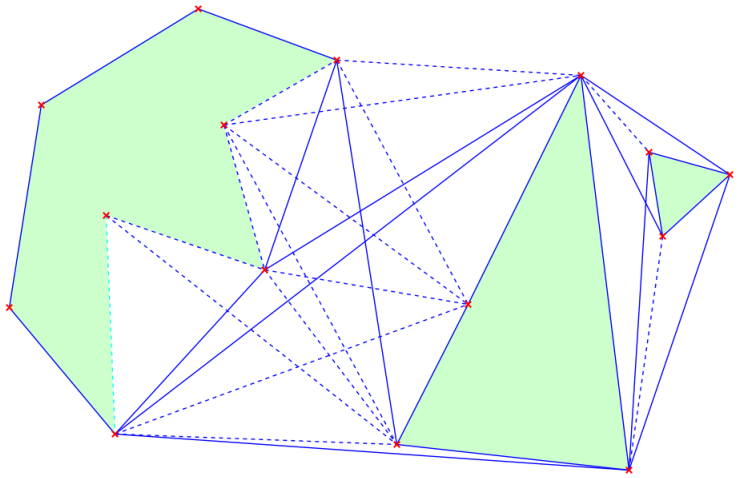


Abbildung 2.12: Sichtbarkeitsgraph. Die durch die Reduzierung entfallenden Kanten sind unterbrochen dargestellt.

Anstatt Pfade zu generieren, die möglichst viel Abstand zu den Hindernissen haben, können auch Pfade erstellt werden, deren Abstand zu den Hindernissen maximal gering ist. Dadurch entsteht das Streckennetz der kürzesten Pfade. Es handelt sich hierbei um einen Spezialfall des *Sichtbarkeitsgraphen*, bei dem alle Eckpunkte von C_{obs} miteinander mit Geraden verbunden werden, sofern diese Geraden das Innere von C_{obs} nicht schneiden.

Um den so entstehenden Sichtbarkeitsgraphen in der Anzahl der Kanten und Knoten zu verringern, werden alle Kanten entfernt, deren Verlängerung das Innere von C_{obs} schneiden würden. Abbildung 2.12 zeigt einen Sichtbarkeitsgraphen. Die unterbrochenen Kanten sind die Kanten, die durch die Reduzierung entfallen. Es ist nachweisbar, dass der reduzierte Sichtbarkeitsgraph immer den küzesten Weg im zweidimensionalen Fall enthält [Liu und Arimoto (1992)].

2.5.3 Planung mit Rückkopplung

Nicht immer ist es möglich, die exakte Position eines Roboters zu bestimmen, die jedoch für die Pfadplanung mit den bisher vorgestellten Methode notwendig ist, da diese immer von einer bekannten Startkonfiguration q_i ausgehen. Zudem wird davon ausgegangen, dass ein Roboter exakt auf dem geplanten Pfad fahren kann. Dieses ist in der Realität jedoch nur selten der Fall. Bei rückgekoppelten Ansätzen sind die Steuerungsparameter nicht von einem vorbestimmten Pfad, sondern von der aktuellen Konfiguration abhängig. Es findet somit eine Art *Regelung* statt. Dieses kann auf zwei Arten geschehen:

1. Die Unsicherheiten werden bei der Planung nicht berücksichtigt. Zudem wird während der Fahrt mittels Reglern die Abweichung von dem berechneten Pfad minimiert.
2. Die Unsicherheiten werden bei der Planung mitberücksichtigt. Es wird für jede mögliche Konfiguration ein Pfad zum Ziel berechnet.

Der erste Fall soll nicht weiter beschrieben werden, da davon auszugehen ist, dass jedem Pfadplanungsverfahren Regler nachgeschaltet sind. Der zweite Fall kann mit Hilfe der *Navigationsfunktion* gelöst werden.

Potentialfeldmethode

Die Idee der Potentialfeldmethode ist es, das Fahrzeug als ein geladenes Teilchen in einem virtuellen Potentialfeld zu modellieren. Das Potentialfeld hat in der Zielregion ein globales Minimum und jedes Hindernis stellt eine abstoßende Kraft dar. Das Teilchen folgt dabei dem negativen Gradienten des Feldes. Probleme, die dabei auftreten können, sind lokale Minima, in denen das Teilchen auf dem Weg zum Ziel hängen bleiben kann.

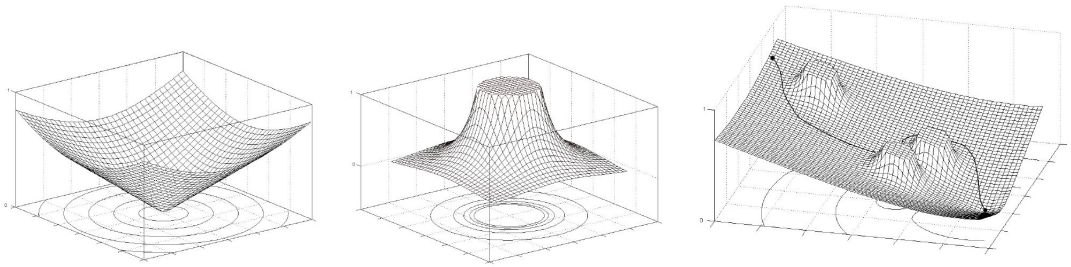
Definition 2.7. *Potentialfeldfunktion*
Die Potentialfeldfunktion ϕ ist eine Funktion $\phi : C_{free} \rightarrow \mathbb{R}$, die jeder Konfiguration eine reelle Zahl zuordnet. Mittels eines Operators ist es möglich, eine Aktion mittels ϕ zu berechnen, welches das Fahrzeug näher zum Ziel bewegt. Dieser Operator ist folgendermaßen

definiert:

$$u* = \arg\min_{u \in U(q)} \{\phi(q')\} \tag{2.24}$$

dabei ist q' die Konfiguration, die ensteht, wenn u auf q angewandt wird. Der Gradient $\nabla\phi$ ist wie folgt definiert:

$$\nabla\phi = \left[\frac{\partial\phi}{\partial x_1}\frac{\partial\phi}{\partial x_2}\cdots\frac{\partial\phi}{\partial x_n}\right] \tag{2.25}$$



(a) ϕ_{att} (b) ϕ_{abs} (c) ϕ

Abbildung 2.13: Potentialfelder (aus [Deutsch (2004)])

Bei der Potentialfeldmethode wird zwischen zwei Arten von Potentialen unterschieden: dem abstoßenden (ϕ_{abs}) und dem anziehenden (ϕ_{att}) Potential. Das Potential an der Stelle q ist die Summe beider Potentiale:

$$\phi(q) = q_{att}(q) + \phi_{abs}(q) \tag{2.26}$$

Das anziehende (attraktive) Potential ϕ_{att} hat die Aufgabe, den Roboter in die Richtung der Zielkonfiguration q_g zu bewegen. Dazu muss dieses Potential im gesamten C vorhanden sein und immer in Richtung q_g wirken. Eine mögliche Funktion für ϕ_{att} ist in der Gleichung 2.27 dargestellt.

$$\phi_{att}(q) = \xi p_g(q)^2 \tag{2.27}$$

ξ aus Gleichung 2.27 ist ein positiver Faktor für die Skalierung des Potentials und $p_g(q)$ gibt die euklidische Distanz zwischen einer Konfigutation und der Zielkonfiguration an:

$$p_g(q) = ||q - q_g|| \tag{2.28}$$

Die Funktion ist in Abbildung 2.13(a) dargestellt. Andere Funktionnen sind für das attraktive Potential ebenso denkbar.

Das abstoßende Potential ϕ_{abs} dient dazu, den Roboter während der Fahrt von Hindernissen fernzuhalten. Grundsätzlich ist ϕ_{abs} in der Nähe von Hindernissen sehr groß und nimmt stark

mit dem Abstand zum Hindernis ab, so dass weit entfernt Hindernisse keinen Einfluss haben. Aus diesem Grund wird ϕ_{abs} auch als *lokales* Potential bezeichnet. Für ϕ_{abs} gilt somit, dass die Wirkung nur in der nahen Umgebung von einem Hindernis Einfluss hat und vom Hindernis weggerichtet ist. Eine mögliche Funktion für ϕ_{abs} ist in Gleichung 2.29 dargestellt.

$$\phi_{abs} = \begin{cases} \eta \left(\frac{1}{p_o(q)} - \frac{1}{p_0} \right)^2 & \text{wenn } p_o(q) \leq p_0 \\ 0 & \text{wenn } p_o(q) > p_0 \end{cases} \tag{2.29}$$

p_o gibt dabei die jeweilige euklidische Distanz zum Hindernis an. η ist ein positiver Skalierungsfaktor für das Potential und p_0 definiert den Einflussbereich des Hindernispotentials. Für ein punktförmiges Hindernis ergibt sich ein Potentialverlauf wie in Abbildung 2.13(b) dargestellt. Da in der Regel mehrere Hindernisse in C_{obs} vorhanden sind, ist es nötig, alle Hindernisse zu berücksichtigen:

$$\phi(q) = \phi_{att}(q) + \sum_{q_o \in C_{obs}} \phi_{abs(q_o)}(q) \tag{2.30}$$

Dabei ist $\phi_{abs(q_o)}(q)$ das Potential bei q unter Betrachtung des Hindernisses q_o. Abbildung 2.13(c) zeigt eine Potentialfeldfunktion mit mehreren Hindernissen.

2.6 Planung in nicht statischen Welten

Bisher wurde in der Planung davon ausgegangen, dass die Welt, in der sich der Roboter bewegt, statisch ist und sich somit mit der Zeit nicht verändert. Jedoch ist dieses in einer realen Umwelt nur sehr selten gegeben. Ist es möglich, die Wege der Hindernisse vorherzusagen oder sind diese gegeben, so können diese mit in die Planung einfließen. Sind die Wege nicht weit genug vorherbestimmbar, so sind Ansätze, wie in Kapitel 2.5.3 zu verwenden. Dieser Fall soll jedoch hier nicht weiter betrachtet werden.

Um die zeitliche Abhängigkeit mit in der Planung zu berücksichtigen, wird ein *Zeitintervall* $\mathcal{T} \in \mathbb{R}$ eingeführt. Dieses kann beschränkt $\mathcal{T} = [0, t_f]$ oder unbeschränkt $\mathcal{T} = [0, \infty)$ sein. Im weiteren wird nur das unbeschränkte Intervall betrachtet. Es kann auch eine andere Zeit für die initiale Zeit 0 gewählt werden, jedoch wird dieses hier der Übersichtlichkeit halber nicht gemacht. Der Zustandsraum X für die Suche ist $X = C \times \mathcal{T}$, dabei ist C der Konfigurationsraum für den Roboter aus Kapitel 2.4. Ein Zustand x wird als $x = (q, t)$ dargestellt. Die Pfadsuche wird in X vorgenommen, wobei die Bedingung zu beachten ist, dass die Zeit in einem Pfad fortlaufend ist. Das Problem 2.4 lässt sich erweitern, so dass das folgende Problem entsteht:

Problem 2.5. *Zeitbasierte Planung*

1. *Eine Welt $\mathcal{W}$ ist gegeben, hier $\mathcal{W} = \mathbb{R}^2$.*
2. *Ein* Zeitintervall *$\mathcal{T} \subset \mathbb{R}$ ist gegeben, hier $\mathcal{T} = [0, \infty)$.*
3. *Eine semi-algebraische, zeitabhängige Hindernisregion $\mathcal{O}(t) \subset \mathcal{W}$ ist für alle $t \in \mathcal{T}$ gegeben.*
4. *Ein Roboter A und der dazugehörige Konfigurationsraum C sind gegeben.*
5. *Der Zustandsraum X wird aus dem kartesischen Produkt $X = C \times \mathcal{T}$ gebildet. Ein Zustand $x \in X$ wird als $x = (q, t)$ dargestellt und bezeichnet eine Konfiguration q und den Zeitpunkt t. Die Hindernisregion X_{obs} im Zustandsraum wird als*

$$X_{obs} = \{(q, t) \in X \mid A(q) \cap \mathcal{O}(t) \neq \emptyset\} \tag{2.31}$$

   definiert. $X_{free} = X \setminus X_{obs}$. Für einen Zeitpunkt $t \in \mathcal{T}$ können X_{obs} und X_{free} berechnet werden. Diese werden als C_{obs} und C_{free} mit

$$C_{obs}(t) = \{q \in C \mid A(q) \cap \mathcal{O}(t) \neq \emptyset\} \tag{2.32}$$

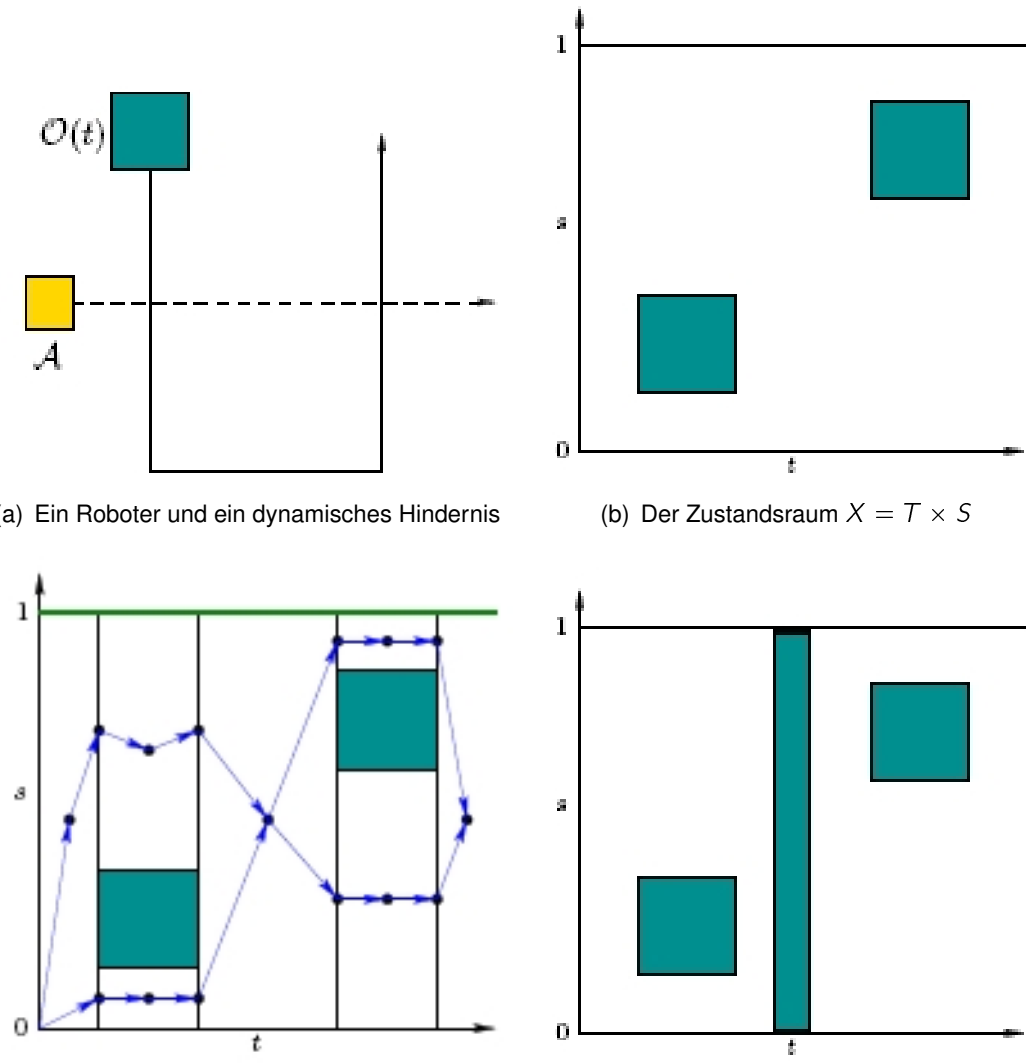
   und $C_{free} = C \setminus C_{obs}$ definiert.
6. *Ein Zustand $x_i \in X_{free}$ wird als Startzustand definiert, mit der Bedingung $x_i = (q_i, 0)$ und $q_i \in C_{free}$. Der Roboter kann sich im Startzustand nicht in einer Kollision befinden.*
7. *Eine Teilmenge $X_G \subset X$ ist die Zielregion. Diese wird mit $q_g \in C$ als $X_G = \{(q_g, t) \in X_{free} \mid t \in \mathcal{T}\}$ definiert. Die Zielregion ist somit fest.*
8. *Ein kompletter Algorithmus muss einen kontinuierlichen und zeitlich monotonen Pfad $\tau : [0, 1] \to X_{free}$ mit $\tau(0) = x_i$ und $\tau(1) \in X_G$ finden oder korrekt zurückmelden, dass keiner exisitiert. Zeitlich monoton impliziert, dass für alle $s_1, s_2 \in [0, 1]$ mit $s_1 < s_2$ gilt, dass $t_1 < t_2$ mit $(q_1, t_1) = \tau(s_1) \wedge (q_2, t_2) = \tau(s_2)$ gilt.*

2.6.1 Direkte Ansätze

Viele Ansätze, die der Klasse der sampling based Algorithmen aus Kapitel 2.5.1 angehören, können einfach von C auf X portiert werden. Die Zeitabhängigkeit muss lediglich bei der Überprüfung der Kollisionsfreiheit von Pfaden berücksichtigt werden. Kombinatorische Ansätze können auch zum Teil erweitert werden, wenn das Modell algebraisch beschrieben ist.

Zu beachten ist jedoch, dass Kanten in den Steckennetzen immer eine monoton steigende Zeit haben.

2.6.2 Geschwindigkeitanpassender Ansatz



(a) Ein Roboter und ein dynamisches Hindernis

(b) Der Zustandsraum $X = T \times S$

(c) Mögliche Zeitfunktionen, die durch vertikale Zerlegung entstehen

(d) Ein dynamisches Hinderniss blockiert den Pfad τ

Abbildung 2.14: (aus [LaValle (2006), S.318f])

Ein anderer Ansatz ist, das $C \times \mathcal{T}$ Problem zu lösen, die Zerlegung des Problems in die *Pfadplanung* und die *Bewegungszeitplanung*. Die Idee ist, *statische* und *dynamische* Hindernisse zu unterscheiden. Die Pfadplanung sucht einen Pfad $\tau : [0,1] \rightarrow C_{free}$ und berücksichtigt dabei nur statische Hindernisse. Die Pfadplanung kann somit mit einem Ansatz aus Kapitel 2.2 gelöst werden. Die Zeitplanung wird in einem zweiten Schritt durchgeführt. Dazu wird eine *Zeitfunktion (Zeitskalierungsfunktion)* $\sigma : \mathcal{T} \rightarrow [0,1]$ definiert, die zu jedem Zeitpunkt $t \in \mathcal{T}$ angibt, an welcher Position sich der Roboter entlang des Pfades τ befindet. Diese wird durch die Komposition $\phi = \tau \circ \sigma$ erreicht, welche $\mathcal{T}$ durch $[0,1]$ nach C_{free} abbildet. Somit ist $\phi : T \rightarrow C_{free}$. Die Berechnung der Zeitfunktion kann ebenso mit den Algorithmen aus Kapitel 2.2 geschehen, dazu wird der Zustandsraum $X = \mathcal{T} \times S$ definiert. S ist dabei der Definitionsbereich von τ. Ein Zustand $x \in X$ wird als $x = (t, s)$ bezeichnet und bezeichnet jeweils die Zeit $t \in \mathcal{T}$ und die Position entlang des Pfades $s \in S$. Die Hindernisregion in diesem Zustandsraum ist als

$$X_{obs} = \{(t,s) \in X \mid A(\tau(s)) \cap \mathcal{O}(t) \neq \emptyset\} \tag{2.33}$$

definiert. X_{free} ist wieder als $X_{free} = X \setminus X_{obs}$ definiert. Die Aufgabe ist es, jetzt einen Pfad $g : [0,1] \rightarrow X_{free}$ zu finden. Die Suche nach g ist mit den gegebenen Mitteln einfach, da es sich immer um einen zweidimensionalen Zustandsraum handelt. Zu beachten ist jedoch, dass dieser Ansatz nicht *komplett* ist. Ein vorhandener Pfad kann nicht immer gefunden werden. Dieses ist zum Beispiel der Fall, wenn ein dynamisches Hindernis den Pfad τ dauerhaft blockiert. Die Abbildung 2.14(a) zeigt einen Roboter, dessen Pfad geplant ist, welcher jedoch von einem dynamischen Hindernis gekreuzt wird, wodurch es zu Kollisionen kommen kann. In Abbildung 2.14(c) ist der Zustandsraum $X = \mathcal{T} \times S$ abgebildet. Es wird eine Funktion gesucht, die monoton steigend ist. Die Suche kann mittels der vertikalen Zellzerlegung vorgenommen werden, dieses wird in Abbildung 2.14(c) gezeigt. Die daraus resultierende Zeitfunktion ist jedoch weit entfernt von dem Optimum, welches durch die Funktion $s = 1$ gekennzeichnet ist. Ein dynamisches Hindernis, welches den Pfad blockiert, ist in Abbildung 2.14(d) dargestellt. Eine Lösung ist nicht berechenbar.

2.7 Fahrzeugemodell

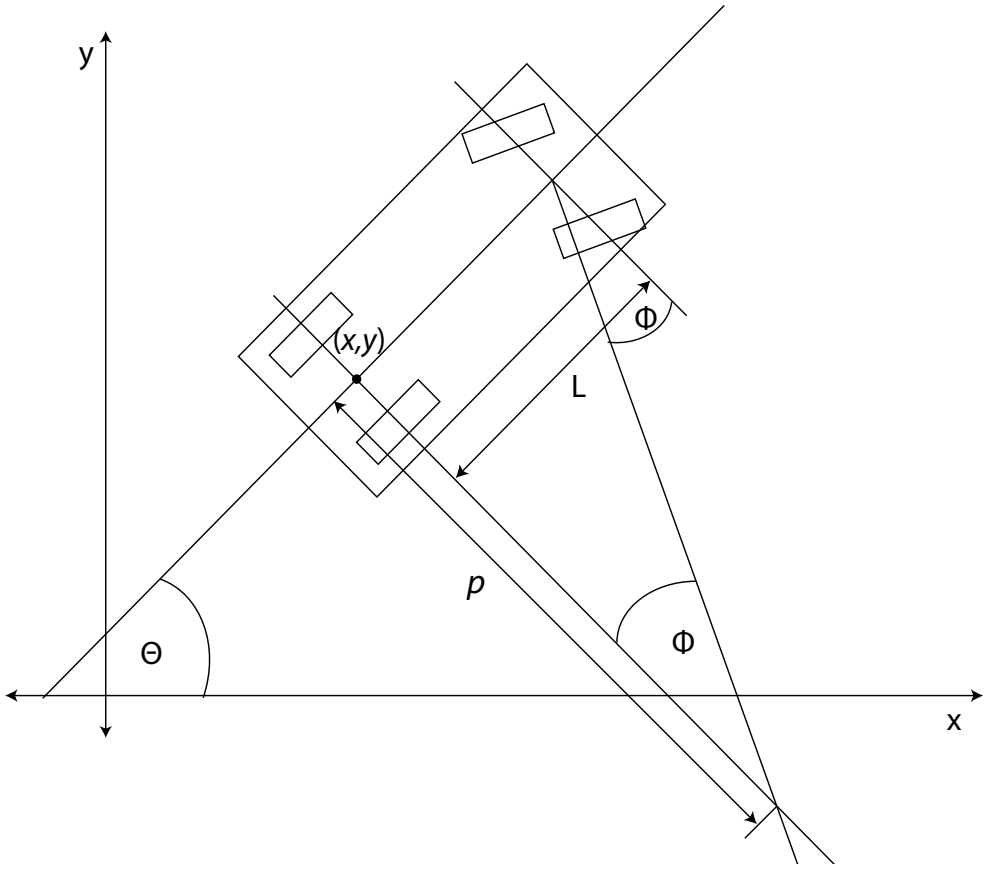


Abbildung 2.15: Fahrzeugmodell

Die Fahrzeuge, die für diese Arbeit genutzt werden, lassen sich in die Klasse der vierrädrigen Kraftfahrzeuge mit lenkbarer Vorder- und starrer Hinterachse einordnen. Die Abbildung 2.15 zeigt die Struktur eines solchen Fahrzeuges. In diesem Abschnitt wird die Kinematik solcher Fahrzeuge beschrieben, da diese für die Pfadplanung nötig ist. Der Konfigurationsraum C für ein solches einfaches Fahrzeug (carlike Robot) ist gemäß dem Abschnitt 2.3 vom Typ $\mathbb{R}^2 \times \mathcal{S}^2$, somit ist eine Konfiguration $q = (x, y, \theta)$ eindeutig bestimmt. Der Referenzpunkt O_A liegt mittig auf der Hinterachse. Da sich das Fahrzeug nur in die Richtung der Vorderräder bewegen kann, gilt für einen unendlich kleinen Zeitschritt folgende Gleichung

$$\tan\theta = \frac{\partial y}{\partial x} \tag{2.34}$$

Wird der Bruch mit ∂t erweitert und berücksichtig, dass $\tan = \frac{\sin}{\cos}$ ist, so entsteht die Gleichung

$$\frac{\sin\theta}{\cos\theta} = \frac{\frac{\partial y}{\partial t}}{\frac{\partial x}{\partial t}} = \frac{\dot{y}}{\dot{x}} \tag{2.35}$$

welche sich zu

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \tag{2.36}$$

umformen lässt. Diese Gleichung ist erfüllt, wenn $\dot{x} = \cos\theta$ und $\dot{y} = \sin\theta$ ist. Alle skalaren Multiplikatoren mit dieser Lösung führen ebenso zu einer Lösung. Als Skalar kann dabei die Geschwindigkeit v genutzt werden. Die allgemeine Lösung lautet somit:

$$\dot{x} = v * \cos\theta \tag{2.37}$$

$$\dot{y} = v * \sin\theta \tag{2.38}$$

Die Änderung der Orientierung θ des Fahrzeuges entsteht durch den Lenkeinschlag ϕ des Fahrzeuges. Bei gleichbleibenden ϕ bewegt sich das Fahrzeug auf einer Kreisbahn mit den Radius p

$$p = \frac{L}{\tan\theta} \tag{2.39}$$

dabei ist L der Radstand des Fahrzeuges. Unter Berücksichtigung, dass für einen unendlich kleinen Zeitschritt auf einer Kreisbahn für den zurückgelegten Weg gilt: $\partial s = \partial\theta p$, entsteht für die Änderung von θ die Gleichung:

$$\partial\theta = \partial s\frac{\tan\phi}{L} \tag{2.40}$$

Da für die Ableitung nach der Zeit für die zurückgelegte Strecke $\frac{\partial s}{\partial t} = v$ gilt, entsteht nach der Division mit ∂t die Gleichung:

$$\dot{\theta} = \frac{v}{L}\tan\phi \tag{2.41}$$

Für die oben genannten Fahrzeuge lässt sich das Fahrzeugmodell wie folgt zusammenfassen:

$$\dot{q} = \begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix} = v * \begin{pmatrix}\cos\theta\\ \sin\theta\\ \tan\frac{\phi}{L}\end{pmatrix} \tag{2.42}$$

Der Konfigurationsraum ist dreidimensional, jedoch sind nur zwei Steuergrößen vorhanden, was folgende Umstellung verdeutlicht:

$$\dot{q} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = v * \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} + \omega * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.43}$$

$$\omega = \frac{v}{L} \tan\phi \tag{2.44}$$

Roboter, wie das beschriebene Fahrzeug, unterliegen mechanischen Zwangsbedingungen. Zu erkennen ist dieses an der Gleichung 2.42. Die Anzahl der Steuerungsparameter ist geringer als die Dimension des Konfigurationsraumes. Das Fahrzeug kann sich somit nicht direkt in alle Richtungen bewegen und seine Ausrichtung ohne eine Bewegung in Längsrichtung ändern. Dieses hat essentielle Folgen auf die Pfadplanung, deshalb werden diese Zwangsbedingungen im weiteren definiert.

2.7.1 Holonome Beschränkungen

Roboter, deren Bewegungsfreiheit durch mechanische Zwangsbedingungen eingeschränkt und dabei unabhängig von der Geschwindigkeit sind, werden als *holonom* bezeichnet. Holonome Bedingungen lassen sich mathematisch durch eine Gleichung der Form

$$F(q, t) = F(q_1 \ldots, q_m, t) = 0 \tag{2.45}$$

beschreiben. Wobei F eine stetige Funktion mit Ableitungen ungleich 0 darstellt. Eine solche Gleichung definiert eine Menge von Konfigurationen, die ein Roboter einnehmen kann. Es gibt zwei Varianten von holonomen Bedingungen: in Gleichungsform reduzieren sie häufig die Dimension des Konfigurationsraumes, indem sie die Bewegung auf eine gekrümmte Fläche oder gar auf eine Linie reduzieren. In Ungleichungsform reduzieren sie die Größe des Konfigurationsraumes, indem sie z.B. die Bewegungsfreiheit einschränken.

Beispiel 2.6. *Straßenbahn*
Eine Straßenbahn auf einem weichenfreien Gleis ist ein Fahrzeug mit holonomen Beschränkungen. Das Fahrzeug bewegt sich in einer dreidimensionalen Welt, jedoch beschränkt sich der Konfigurationsraum auf nur eine Dimension, da sich die Straßenbahn auf dem Gleis nur vorwärts und rückwärts bewegen kann.

Beispiel 2.7. *Pendel*
Ein Pendel (eine Kugel an einem Seil aufgehängt) ist ein holonomes System. Das Seil hat immer die konstante Länge l. Aufgrund des Satzes von Pythagoras gilt die Zwangsbedingung $x^2 + y^2 = l^2 \Rightarrow x^2 + y^2 - l^2 = 0$

2.7.2 Nichtholonome Beschränkungen

Zu den holonomen Beschränkungen können auch Beschränkungen der Geschwindigkeiten eines Roboters auftreten. Diese Art von Bedingungen können in der Form

$$G(q, \dot{q}, t) = G(q_1, \ldots, q_m, \dot{q_1}, \ldots, \dot{q_m}, t) = 0 \tag{2.46}$$

beschrieben werden. Können nicht alle Ableitungen von q aus der Gleichung eliminiert werden, so handelt es sich um eine *nichtholonome* Zwangsbedingung. Solche Einschränkungen reduzieren nicht die Dimension des Konfigurationsraumen, sondern meistens die Anzahl der Steuergrößen. Daher werden solche Systeme auch als *untersteuert* bezeichnet.

Beispiel 2.8. *Untersteuert*
Ein Fahrzeug mit den kinematischen Eigenschaften aus Gleichung 2.42 hat einen dreidimensionalen Konfigurationsraum, jedoch nur zwei Steuergrößen. Die Positionsänderung ist abhängig von der Ausrichtungsänderung:

$$G_{car}(x, y, \theta, \dot{\theta}) = 0 \tag{2.47}$$

2.8 Planung unter Zwangsbedingungen

Die vorgestellen Pfadplanungsverfahren setzen allesamt voraus, dass zwei Konfigurationen mittels einer direkten Strecke verbunden werden können. Dieses bedeutet, dass ein Roboter jede Position von jeder beliebigen Position direkt anfahren kann. In Kapitel 2.7 wurde eine Beschreibung eines Fahrzeugemodells eingeführt. Sobald *nichtholonome* Beschränkungen vorhanden sind, ist es nicht mehr möglich, zwei Konfigurationen ohne weiteres zu verbinden.

Die Beschränkungen schränken die Menge der Konfigurationen ein, die von einer Konfiguration erreicht werden können. Es gibt für die Planung unter Zwangsbedingunen zwei Ansätze: die Zwangsbedingungen können *während* der Planung oder *nach* der Planung berücksichtigt werden. Im erst genannten Ansatz werden von dem Pfadplanungsalgorithmus nur Pfade berechnet, welche auch von dem Roboter bewältigt werden können. Bei dem zweiten Ansatz wird ein Pfad geplant, der die Zwangsbedingungen nicht berücksichtigt, und anschließend der Pfad mittels geometrischen Verfahren so verformt, dass er den Zwangsbedingungen genügt.

Nicht alle Pfadplanungverfahren sind für die Planung mit Zwangsbedingungen geeignet. Kombinatorische Ansätze genügen meist nicht der Bedingung eines Streckennetzes mit der Definition 2.6 genügt, da die Berechnung von Pfaden nicht mehr trivial ist.

2.8.1 Sampling-Based Ansatz

Der in Kapitel 2.5.1 vorgestellt Ansatz beruht darauf, dass es einfach möglich ist, einen Pfad zwischen zwei Konfigurationen zu berechnen. Da dieses unter nichtholonomen Zwangsbedingungen nicht mehr der Fall ist, ist der Ansatz nicht direkt erweiterbar. Eine Lösung des Problems ist es, nicht den Konfigurationsraum C, sondern den Aktionsraum U zu durchsuchen. Angenommen die Beschränkungen lassen sich durch eine Zustandsübergangsfunktion der Form $\dot{q} = f(q, u)$ darstellen. Ein Pfad lässt sich nicht mehr einfach durch die Folge von Aktionen, wie in Kapitel 2.2.1 darstellen. Anstelle dessen wird der Pfad durch das Integrieren über die Aktionsfolge erstellt.

Der Aktionsraum U ist eine beschränkte Menge von $\mathbb{R}^m$. Ziel ist es eine *Aktionstrajektorie* $\tilde{u} : [0, 1] \rightarrow U$ zu berechnen. Es wird wieder die aus Kapitel 2.2.3 bekannte Terminierungsaktion u_T eigeführt. Dieses führt dazu, dass der Definitionsbereich von $\tilde{u}$ auf $[0, \infty)$ erweitert werden kann. Die Aktionstrajektorie kann jetzt wie folgt berechnet werden:

$$x(t) = x(0) + \int_0^t f(x(t'), u(t'))dt' \tag{2.48}$$

Dabei ist $u(t)$ die zum Zeitpunkt t aktuelle Steueraktion.

Es gibt eine Reihe von verschiedenen Ansätzen um die Aktionstrajektorie zu berechnen. Grundsätzlich wird die Aktionstrajektorie diskretisiert. Bei dem *diskreten Zeitmodell* wird die Zeit $\mathcal{T}$ in feste Intervalle der Länge Δt eingeteilt. Der aktuelle Zeitpunkt kann somit als Anzahl k der vergangenen Zeitintervalle angegeben werden. Dabei heißt k, dass $(k-1)\Delta t$ Zeitintervalle verstrichen sind. Zudem wird eine endliche Menge an Aktionen als Aktionsraum U_d gewählt. Ist für jedes Zeitintervall Δt die Aktion $u(t) \in U_d$ konstant, so spricht man von der Verwendung des diskreten Zeitmodells.

Eine Trajektorie in U_d kann als einfache Sequenz von Aktionen $(u_1, u_2, \ldots, u_k)$ angegeben werden. Jedes $u_i \in U_d$ beschreibt dabei die Aktion in dem Zeit $(i-1)\Delta t$ bis $i\Delta t$. Nach dem k-ten Schritt wird die Terminierungsaktion angewandt. Für alle Startzustände kann eine erreichbare Menge $R(x_o, \mathcal{U})$ angegeben werden, $\mathcal{U}$ ist die Menge aller verfügbaren Aktionen im Zeitintervall $[0, \infty)$. Formal ist dieses wie folgt definiert:

$$R(x_0, \mathcal{U}) = \{x_1 \in X \mid \exists \tilde{u} \in \mathcal{U} \wedge \exists t \in [0, \infty) \wedge x(t) = x_1\} \tag{2.49}$$

Ein Erreichbarkeitsgraph $T_r(x_0, \mathcal{U}_d$ ist berechenbar. Dieser unterscheidet sich von $T_r(x_0, \mathcal{U})$ in der Menge der erreichbaren Zustände, da viele durch die Diskretisierung wegfallen. Die Menge aller Zustände, die durch T_r erreichbar sind (nicht nur die Knoten),ist S (vgl. Kapitel 2.5.1).

$$S(T_r) = \bigcup_{e \in E} \bigcup_{[0,\Delta t] x_e(t)} \tag{2.50}$$

Dabei ist E die Menge der Kanten von T_r, welche eine Aktion u über die Zeit Δt beschreibt.

Beispiel 2.9. *Erreichbarkeitsgraph eines* Dubins Fahrzeuges

Die Einschränkung des diskreten Zeitmodells, dass eine Aktion über ein Intervall konstant sein muss, ist nicht anwendbar. Ein Fahrzeug, zum Beispiel, kann nicht den Lenkwinkel oder die Geschwindigkeit schlagartig von einem Wert zu einem anderen ändern. Es wäre möglich, das Zeitintervall Δt so gering zu machen, dass reale Verläufe möglich sind, jedoch würde dieses zu einem großen Anstieg der Knoten und Kanten in T_r führen. Ein anderer Ansatz ist es, *Bewegungsprimitive* $\tilde{u}^p$ einzuführen. Das Zeitintervall Δt ist jetzt nicht mehr konstant, sondern von der Dauer eines Primitives $t_F(\tilde{u}^p)$ abhängig. Der Wert k bezeichnet jetzt nicht mehr die Anzahl der verstrichenden Zeitintervalle, sondern die Anzahl der Bewegungsprimitiven.

2.8.2 Entkoppelter Ansatz

Wie schon in Kapitel 2.6.2 kann auch an dieser Stelle die Planung unter Zwangsbedingungen in mehrere entkoppelte Schritte zerlegt werden. Der Ansatz kann wie folgt beschrieben werden:

Ansatz 2.2. *Entkoppelter Ansatz*

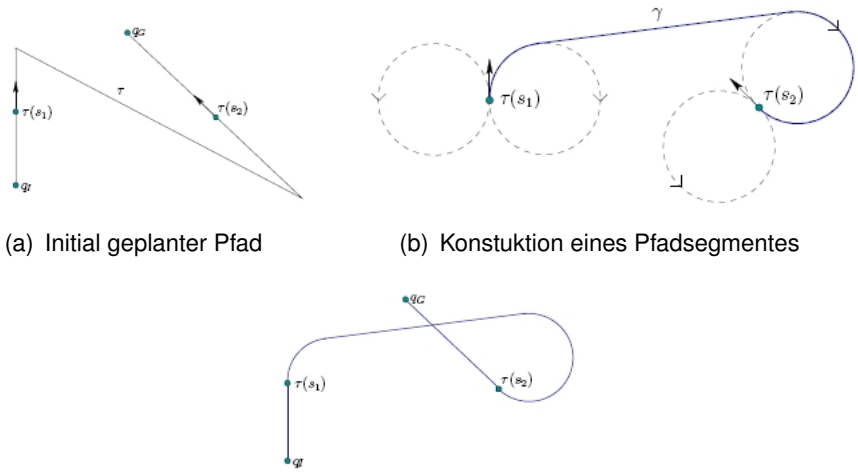
1. *Suche einen Pfad* $\tau : [0, 1] \to C_{free}$*, der keine Zwangsbedingungen beachtet.*
2. *Wähle* $s_1, s_2 \in [0, 1]$*, so dass* $s_1 < s_2$*, und ersetzte den Pfad von* $\tau(s1)$ *nach* $\tau(s_2)$ *durch einen neuen Pfad* γ*, der den Zwangsbedingungen genügt.*
3. *Wenn* τ *den Zwangsbedingungen im Intervall* $[0, 1]$ *genügt, so terminiere, ansonsten gehe zu Schritt 2.*

Die Punkte s_1 und s_2 können zufällig oder deterministisch gewählt werden. Es ist davon auszugehen, dass das Neuplanen eines Teilstückes häufig fehl schlägt. Dieses ist der Fall, wenn die beiden Konfigurationen nicht verbunden werden konnten und/oder dabei eine Kollision auftritt. Der Pfad τ' lässt sich anschließend mit der Formel

$$\tau'(s) = \begin{cases} \tau(s) & \text{wenn } s < s_1 \\ \gamma((s - s_1)/(s_2 - s_1)) & \text{wenn } s \in [s_1, s_2] \\ \tau(s) & \text{wenn } s > s_2 \end{cases} \tag{2.51}$$

berechnen.

Beispiel 2.10. *Entkoppeltes Planung für ein Dubins Fahrzeug*



(a) Initial geplanter Pfad (b) Konstuktion eines Pfadsegmentes

(c) Resultierender Pfad τ'

Abbildung 2.16: Entkoppelter Ansatz für die Planung mit Zwangsbedingungen (aus [LaValle (2006), S.844f])

Für das Beispiel wird angenommen, dass es sich bei dem Fahrzeug um ein Dubins Fahrzeug handelt, das nur vorwärts fahren kann. Abbildung 2.16(a) zeigt einen initial geplanten Pfad τ, der den Zwangsbedinungen nicht genügt. Die beiden „Spitzen“ können jedoch nicht von dem Fahrzeug befahren werden, da es sich hierzu auf der Stelle drehen können müsste. Es werden somit zwei Punkte s_1, s_2 gewählt und zwischen diesen ein neuer Pfad γ geplant. Hierzu wird, wie in Abbildung 2.16(b), eine Doppeltangente gesucht, die die Kreise an den Konfigurationen verbindet. Der Radius der Kreise entspricht dem minimalen Kurvenradius des Fahrzeuges. Der resultierende Pfad τ' in Abbildung 2.16(c) wird durch Anwenden der Formel 2.51 gebildet.

2.9 Umsetzung

Bisher wurde die Theorie und die Literatur vorgestellt, die sich mit der Pfadplanung beschäftigt. In diesem Kapitel wird die konkrete Umsetzung beschrieben. Diese ist in kleine Schritte unterteilt. Die Pfadplanung wird in einer statischen Welt beschrieben, in der sich nur feste Hindernisse befinden. Der gitterbasierte Ansatz löst das Pfadplanungsproblem für holonome Fahrzeuge durch die Diskretisierung des Arbeitsraumes durch ein Gitter. Ein weiterer Ansatz für nicht holonome Fahrzeuge wurde implementiert. Dieses ist ein *Sampling Based*

Ansatz, bei dem der Arbeitsraum nicht durch ein Gitter diskretisiert wird, sondern duch den Aktionsraum des Fahrzeuges.

2.9.1 Implementation für hononome Fahrzeuge

Gitterbasierter Ansatz

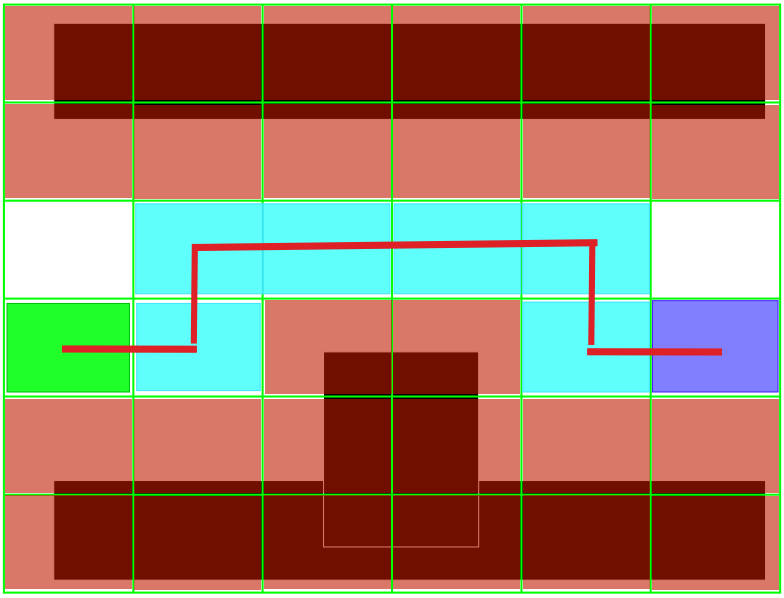


Abbildung 2.17: Pfadsuche für ein holonomes Fahrzeug mittels dem Gitterbasierten Ansatz. In Hindernisse (schwarz) sind um die Abmessungen des Fahrzeuges vergrößert. Die belegten Zellen sind rot markiert. Die von der Suche besuchten hellblau. Der Startpunkt ist grün und das Ziel blau markiert. In rot ist zudem der gefundene Pfad eingezeichnet.

Die Pfadplanung für ein holonomes Fahrzeug in einer statischen Welt ist mit einem gitterbasierten Ansatz gelöst worden. Dabei wird über den Arbeitsraum ein Gitter gelegt. Es handelt sich hierbei um einen kombinatorischen Ansatz, wie in Kapitel 2.5.2 beschrieben. Das Gitter wird verwendet, um den Arbeitsraum zu diskretisieren und damit die Anzahl möglicher Zustände des Fahrzeuges zu reduzieren. Eine Zelle des Gitters wird als belegt definiert, wenn die Zelle ein Hindernis oder Teile eines Hindernisses enthält. Andernsfalls ist die Zelle frei. Zur Überführung des Gitters in einen Konfigurationsraum, in dem das Fahrzeug als Punkt angenommen wird, ist zu beachten, dass der Rand einer Zelle einen Abstand zu einem Hindernis haben muss, der größer als der Radius des Fahrzeuges ist, da es sonst zu Kollisionen kommen kann. Gibt es einen Punkt auf dem Rand einer Zelle, der einen geringeren Abstand

zu einem Hindernis aufweist, so ist die Zelle belegt. Bei der Wahl der Gittergröße ist dieses zu beachten. Ist das Gitter zu grob gewählt, so können enge Passagen nicht passiert werden. Ist das Gitter jedoch zu fein gewählt, so ist der Rechenaufwand der Pfadplanung zu hoch, da zu viele Konfigurationen möglich sind.

Der Konfigurationsraum wird durch einen Graphen repräsentiert, dessen Knoten die freien Zellen des Gitters sind. Die Kanten sind die vierer- (oder achter-) Nachbarschaften der jeweiligen Zellen. In diesem Graphen wird mit dem A*- Algorithmus aus Kapitel 2.2.2 der kürzeste Weg ermittelt. Als Heuristik wird die euklidische Distanz zwischen zwei Konfigurationen genutzt. Die Abbildung 2.17 zeigt exemplarisch eine Lösung für diesen Ansatz.

RRT

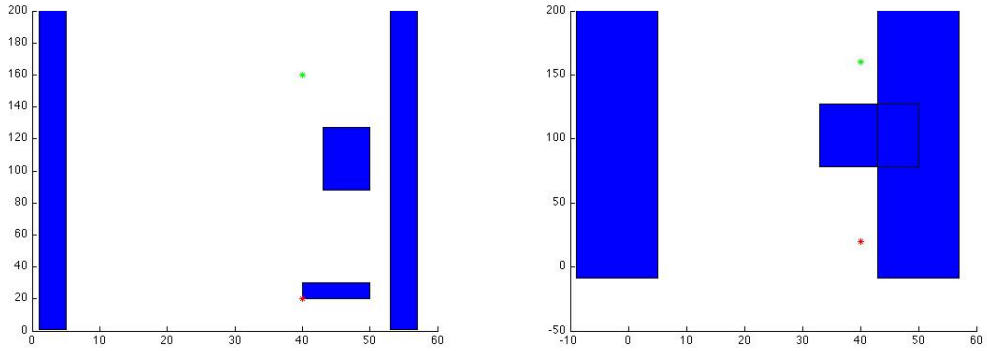
Der implementierte *Gitterbasierte Ansatz* hat den Nachteil, dass die Anzahl der Knoten, die zu durchsuchen sind, von der Gittergröße abhängig ist. Die Suchdauer erhöht sich somit, je kleiner die Gittergröße gewählt ist. Eine zu große Gittergröße kann jedoch mögliche Lösungen verhindern und somit den Algorithmus unbrauchbar machen. Die Alternative zu den kombinatorischen Ansätzen sind die *Sampling Based Ansatze* aus Kapitel 2.5.1. Für holonome Fahrzeuge wurde ein *rapidly exploring tree RRT* implementiert. Die theoretischen Grundlagen für diesen Ansatz sind in Kapitel 2.5.1 beschrieben.

Die Dimension des Konfigurationraumes konnte auf zwei reduziert werden, da durch die statische Umgebung keine Zeit modelliert und durch die holonomen Eigenschaften des Fahrzeuges keine Rotation betrachtet werden muss. Eine Konfiguration wird somit durch

$$q = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.52}$$

beschrieben. Der Konfigurationsraum C ist $C = \mathbb{R}^2$.

Transformation der Hindernisse in Konfigurationsraum Die Hindernisse, die aus dem Arbeitsraum $\mathcal{W}$ extrahiert wurden, müssen für die Pfadsuche in den Konfigurationsraum C_{obs} transformiert werden (vgl. Kapitel 2.4.2). Da ein Fahrzeug im Konfigurationsraum C nur einen einzelnen Punkt darstellt, werden die Hindernisse um die Ausmaße des Fahrzeuges erweitert. Hierzu wird die Minkowski Summe aus Gleichung 2.21 auf alle Hindernisse und das Polygon, das das Fahrzeug beschreibt, angewendet. Die so entstehende Menge von Vektoren beschreibt eine ersteinmal ungeordnete Menge von Start- bzw. Endpunkten von Strecken, die zusammen ein Polygon ergeben. Alle Vektoren, die innerhalb des neuen C_{obs} liegen, werden gelöscht. Die Randpunkte werden nach ihrem Winkel zum Ursprung des Hindernisses sortiert. Diese so enstehende Folge von Vektoren beschreibt das neue



(a) Arbeitsraum $\mathcal{W}$ (b) Konfigurationsraum C

Abbildung 2.18: Transformation von Hindernissen in den Konfigurationsraum

C_{obs}, das Hindernis wird transformiert in den Konfigurationsraum. Dieser Vorgang ist in der Abbildung 2.18 dargestellt. Abbildung 2.18(a) zeigt den Arbeitsraum und das Fahrzeug. In Abbildung 2.18(b) ist das Fahrzeug auf einen Punkt geschrumpft, dafür sind die Hindernisse (C_{obs}) vergrößert. Zu beachten ist, dass eine Rotation des Fahzeuges um seinen Ursprung (Ecke oben Links) nicht möglich ist.

Kollisionserkennung Die Pfade zwischen zwei Konfigurationen sind Geraden. Dieses erleichtert die Kollisionserkennung mit den festen Hindernissen. Die Hindernisse werden, wie in Kapitel 2.3 als konvexe Polygone beschrieben. Der Test, ob ein Pfad mit einem Hindernis kollidiert, ist gleich mit dem Test, ob eine Gerade eines der Hindernispolygone schneidet. Dafür wird ein Polygon nicht als Menge von Halbebenen definiert, sondern als Menge von Strecken. Eine Strecke ist immer durch zwei Punkte beschrieben.

Diese Punkte lassen sich aus der Polygonbeschreibung aus 2.15 ableiten. Dazu werden die Gerade G_i aus den Halbebenen H_i eines konvexen Polygons konstruiert, welche enstehen, wenn $f_i(x, y) = 0$ ist. Der Startpunkt einer Strecke S_i ist der Schnittpunkt der Geraden G_{i-1} und G_i. Das Ende der Stecke S_i ist der Schnittpunkt der Geraden G_i und $G_{(i+1) \mod |\mathcal{O}|}$. Für alle i in $[2, |\mathcal{O}| + 1]$. Ein Hindernispolygon ist somit als die Menge der Strecken, die das Hindernis begrenzen, beschrieben.

$$\mathcal{O} = S_2, S_3, \ldots, S_{m+1} \tag{2.53}$$

Verglichen mit der Gleichung 2.16 ist zu erkennen, dass es jetzt nicht mehr einfach zu erkennen ist, ob ein Punkt innerhalb eines Polygones liegt. Dieses wird jedoch derzeit nicht benötigt.

Da ein Pfad einer Strecke zwischen zwei Punkten entspricht, wurden die Gleichungen 2.18 und 2.19 angepasst, so dass geprüft werden kann, ob eine Strecke ein Polygon schneidet.

$$\alpha(\overline{P_1P_2}) = \bigvee_{i=1}^{m} e_i(\overline{P_1P_2}) \tag{2.54}$$

Dabei ist $e_i(\overline{P_1P_2})$ ein Prädikat, welches $TRUE$ liefert, wenn die Strecke S_i des Polygones einen Schnittpunkt mit der Strecke $\overline{P_1P_2}$ hat, anderenfalls liefert es $FALSE$.

Das Prädikat ϕ gibt an, ob eine Strecke $\overline{P_1P_2}$ mit einem Hindernis kollidiert.

$$\phi(\overline{P_1P_2}) = \bigvee_{i=1}^{n} \alpha_i(\overline{P_1P_2}) \tag{2.55}$$

In der Implementation ist dieser Schritt abgekürzt worden. Die Hindernisse wurden nicht als konvexe Polygone beschrieben, sondern direkt als allgemeine Polygone, die durch die Start und Endpunkte der begrenzenen Strecken definiert sind.

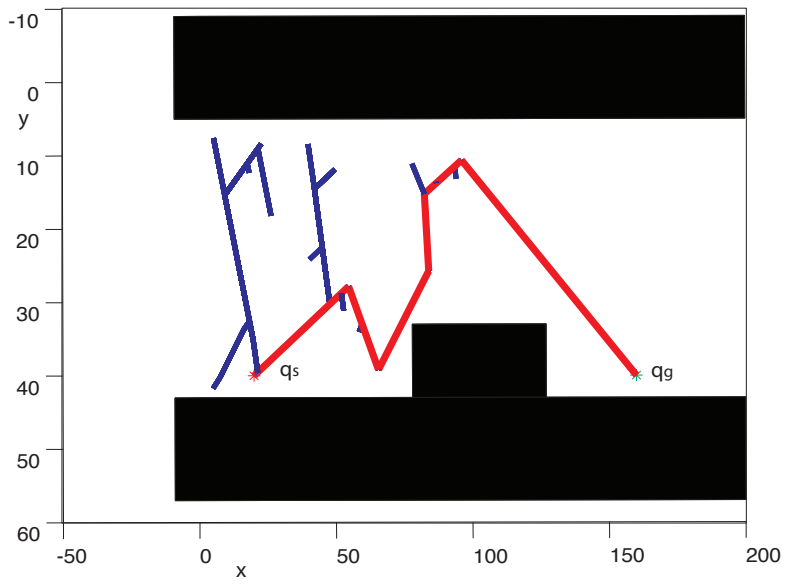
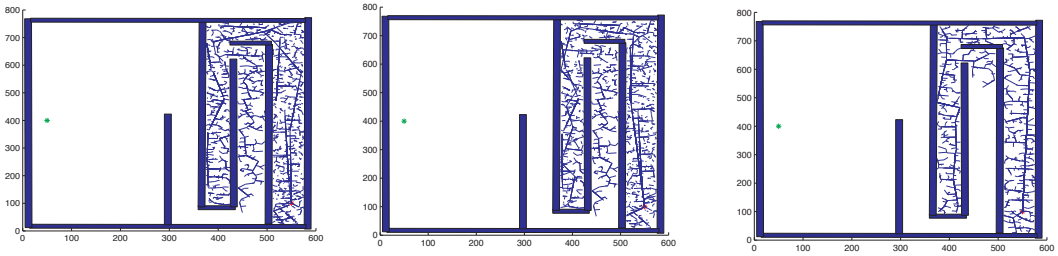


Abbildung 2.19: Pfadsuche für ein holonomes Fahrzeug mittels dem RRT Ansatz. In schwarz sind die Hindernisse, in blau die Pfade zu den zufälligen Konfigurationen und rot der gefundene Pfad dargestellt.

Die Abbildung 2.19 zeigt eine Lösung für das Pfadfindungsproblem für ein holonomes Fahrzeug.



(a) Nach 500 Iterationen (b) Nach 1000 Iterationen (c) Nach 5000 Iterationen

Abbildung 2.20: Versuche mit dem RRT- Ansatz in engen Umgebungen. Nach n Iterationen ist das Ziel nicht erreicht.

Das Problem bei RRT-Ansätzen ist die Strategie nach der zufällige Konfigurationen ausgewählt werden. In engen Passagen müssen mehere Konfigurationen untersucht werden, als im größen freien Räumen. Ist dieses nicht der Fall, so schlägt die Suche fehlt. In Abbildung 2.20 ist diese dargestellt. Nach 500, 1000 bzw. 5000 Iterationen konnte kein Pfad gefunden werden, da kein Pfad aus dem Labyrith gefunden worden ist.

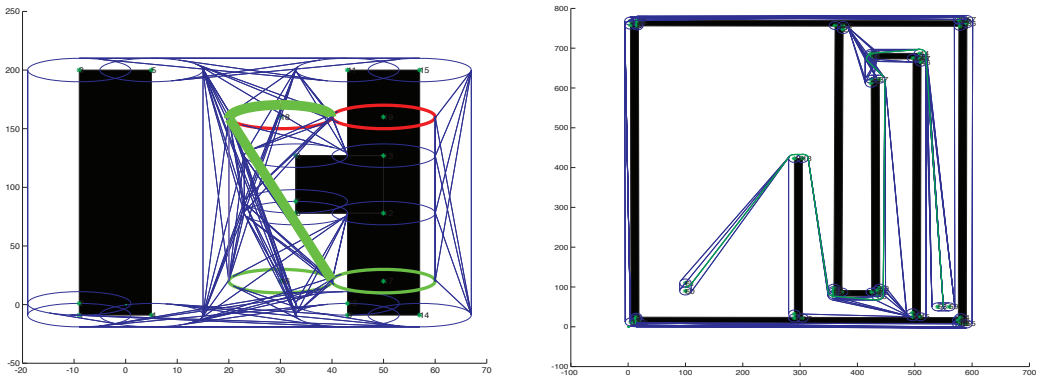
2.9.2 Implementation für nicht holonome Fahrzeuge

Dubins Kurven

Es gibt nicht viele kombinatorische Verfahren für die Pfadplanung von nicht holonomen Fahrzeugen. Jedoch einen, der auf der Dubins Metrik aus [Dubins (1957)] beruht. Die Dubins Metrik besagt, dass im hindernisfreien Raum der Pfad zwischen zwei Konfigurationen für ein Dubins Fahrzeug aus maximal drei Segmenten besteht. Diese Segmente sind entweder Kreise oder Geraden. Dabei bedeutet S eine geradeaus Fahrt L und R eine links bzw. rechts Fahrt mit jeweils minimalen Kurenradius.

$$LRL, RLR, LSL, LSR, RSL, RSR \tag{2.56}$$

In 2.56 sind alle möglichen Variantionen der möglichen Segmente dargestellt.



(a) Pfadsuche in einer Enge (b) Pfadsuche im Labyrinth

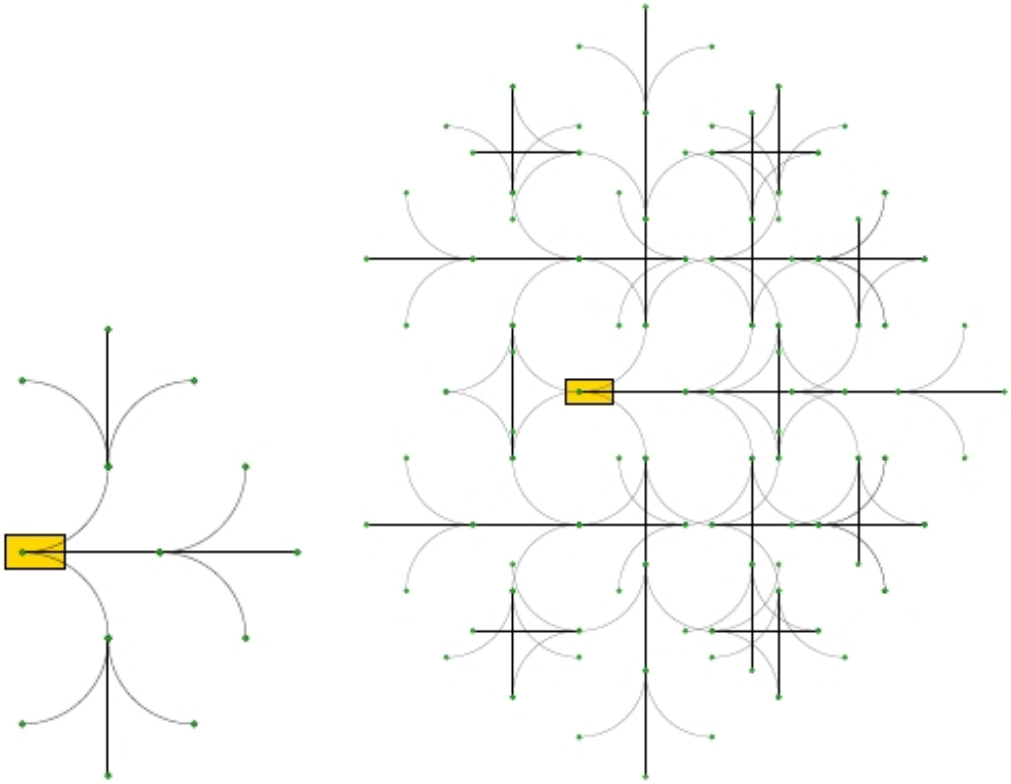
Abbildung 2.21: Pfadsuche für ein nicht holonomes Fahrzeug mittels Dubins Kurven. In schwarz sind die Hindernisse, in blau alle möglichen Pfade und grün der gefundene Pfad dargestellt. Die roten bzw. grünen Kreise sind für der Start- bzw. Zielkonfiguration hinzugefügt.

Der Sichtbarkeitsgraphen Ansatz aus Kapitel 2.5.2 lässt sich mittels der Dubins Metrik erweitern. Anstatt alle Eckpunkte miteinander zu verbinden, wird um jeden Eckpunkt ein Kreis mit dem minimalen Kurvenradius gelegt. Anschließend werden alle Tangenten berechnet, die an jeweils zwei Kreisen anliegen. Die Tangenten werden, um die reduziert, die durch Hindernisse verlaufen. Für die Pfadplanung werden zusätzlich jeweils zwei Kreise für die Startkonfiguration und die Endkonfiguration eingefügt, welche auch mit allen anderen Kreisen mit Tangenten verbunden werden. Die Kreise und die Tangenten bilden einen Suchgraphen, der den optimalen Pfad enthält. Die Pfade die auf diese Weise enstehen, entsprechen immer dem Schema der Dubins Metrik. Durch die Hindernisse werden jedoch weitere Segmente eingefügt. Jeder Pfad beginnt und endet jedoch mit einer Kurvenfahrt.

Sampling- Based Ansatz

Für Fahrzeuge mit nicht holonomen Zwangsbedingungen können die meisten kombinatorischen Ansätze nicht verwendet werden, da diese davon ausgehen, dass innerhalb einer Zelle immer ein einfacher Pfad gefunden werden kann. Der einfachste zu konstruierende Fall, in dem diese Bedingung fehl schlägt, ist, wenn sich die Start- und Zielkonfiguration in derselben Zelle befinden. In diesem Fall würde kein Pfad geplant werden und der *einfache* Pfad, der in 2.5.2 beschrieben ist, exsistiert möglicher Weise nicht.

Die Pfadplanung mit Dubinskurven 2.9.2 findet korrekt optimale Pfade, jedoch können nicht alle kinematischen und dynamischen Eigenschaften eines Fahrzeuges berücksichtigt wer-



(a) Nach zwei Schritten (b) Nach vier Schritten

Abbildung 2.22: Erreichbarkeitsgraph des Dubins Fahrzeuges (aus [LaValle (2006), S. 804])

den. Eine Kurve eines Fahrzeuges entpricht nicht genau einem Kreis, was daraus resultiert, dass die Lenkung nicht schlagartig geändert werden kann. Der Verlauf des Lenkwinkels ist also nicht stetig.

Anstatt, den Arbeitsraum durch ein Raster zu diskretisieren, geschieht dieses durch die möglichen Steueraktionen. Die Erstellung und das Durchsuchen des Konfigurationsraumes wurde in der Implementation zusammengeführt. Es wird zu Beginn nicht der komplette Konfigurationsraum aufgebaut und anschließend durchsucht. In Abbildung 2.22 sind die Konfigurationen gezeigt, die nach mehreren Iterationen erreichbar sind. Dabei besteht der Aktionsraum aus drei Aktionen vorwärts, links und rechts, bei jeweils gleicher Geschwindigkeit.

Von jeder gefundenen Konfiguration aus werden alle Aktionen angewandt. Die gefundenen Konfigurationen werden in die Liste aller Konfigurationen eingefügt. Durch die Verwendung des A-Stern Algorithmus ist die Liste Q nach einer Metrik $G(x)$ sortiert. Als Metrik kann dabei die euklidische Distanz genutzt werden, was jedoch den Nachteil hat, dass zwar die Zielposition erreicht wird, jedoch nicht die Zielausrichtung. Wird statt dessen die Ausrichtung

mit in die Metrik angenommen: $G(x) = sqrt((x_x - x_G)^2 + (y_x - y_G)^2 + (\theta_x - \theta_G)^2)$, so ist dieses eine gute Metrik für ein Fahrzeug, welches sich auf der Stelle drehen kann. Für ein nicht holonomes Fahrzeug kann diese Metrik jedoch nicht eingesetzt werden. Wird die Zielausrichtung nicht getroffen, so werden viele längere Pfade zum selben Ziel geplant, jedoch kann nicht garantiert werden, dass jemals ein Pfad die Zielausrichtung erreicht. Abhilfe schafft hier die Anwendung der Dubins Metrik aus 2.9.2. Werden die Hindernisse ignoriert, so lässt sich immer ein einfacher Pfad konstruieren, dessen Länge leicht berechenbar ist. Diese Länge stellt eine gute Metrik für den A-Stern Algorithmus dar.

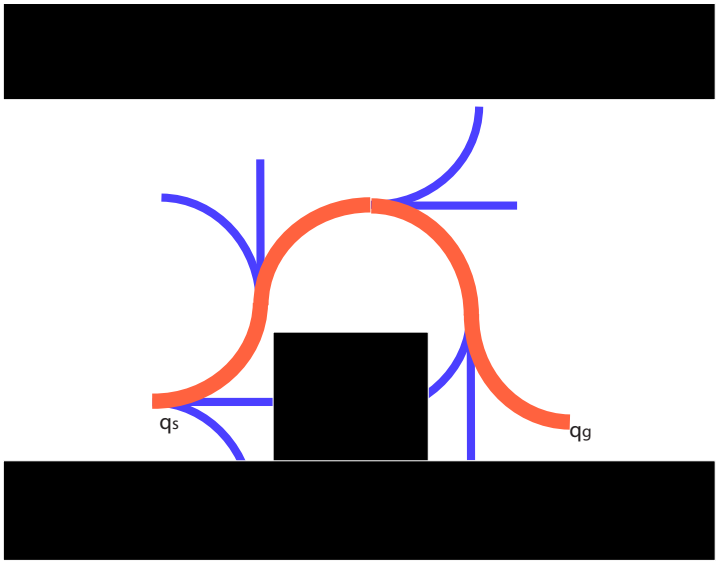


Abbildung 2.23: Pfadsuche für ein nicht holonomes Fahrzeug mittels dem *Sampling Based* Ansatz. In schwarz sind die Hindernisse, in blau untersuchte Teilpfade und in rot der gefundene Pfad dargestellt.

Ein durch das Anwenden einer Aktion entstehender Pfad muss auf Kollisionen mit der Umgebung getestet werden. Die Umsetzung dieses ist in 2.9.2 beschrieben.

Der in 2.5 beschriebene Algorithmus beschreibt das Vorgehen bei der Suche nach einem optimalen Pfad mittels des Sampling Based Ansatzes.

Fahrzeugmodell In Kapitel 2.7 ist der Herleitung eines Fahrzeugmodells beschrieben. Daraus resultierte das Fahrzeugmodell 2.42. Dieses Fahrzeugmodell ist jedoch noch sehr einfach gehalten und beschreibt nur die kinematischen Einschränkungen. Somit sind Geschwindigkeitssprünge durch eine unendliche Beschleunigung möglich. Das im weiteren verwendete Fahrzeugmodell nimmt die aktuelle Geschwindigkeit, den aktuellen Lenkwinkel und

Algorithm 2.5 Sampling Based und A-Star Algorithmen gekoppelt

```
1: Set C(x) = ∞ for all ∀x ∈ X \ {x_i}, and set C(x_i) = 0
2: Q.insert(x_i)
3: while Q ≠ ∅ do
4:    x ← Q.getFirst()
5:    for all u ∈ U(x) do
6:       x' = f(x, u)
7:       if C(x) + w(x, u) < min{C(x'), C(x_G)} then
8:          C(x') ← C(x) + w(x, u)
9:          if x' ≠ x_G then
10:            Q.insert(x', G(x'))
11:         end if
12:      end if
13:   end for
14: end while
```

die maximale Beschleunigung des Fahrzeuges und der Lenkung hinzu. Daraus ensteht folgendes Fahrzeugmodell:

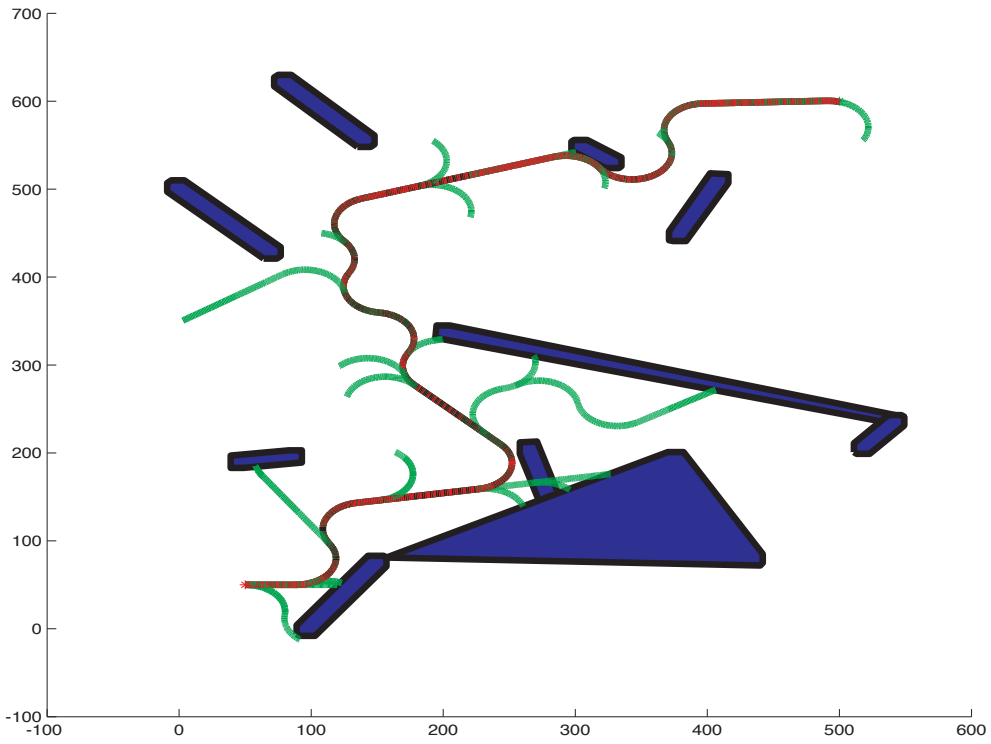
$$\dot{q} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \ddot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v * \cos\theta \\ v * \sin\theta \\ v * \tan\frac{\phi}{L} \\ \ddot{\phi} \\ \alpha \\ a \end{pmatrix} \tag{2.57}$$

Dabei ist $\alpha \in \{\alpha_{min}, 0, \alpha_{max}\}$ die Lenkwinkelbeschleunigung. Wobei $|\dot{\phi}|$ durch $\dot{\phi}_{max}$ die maxmiale Lenkwinkeländerungsgeschwindigkeit nach obenhin begrenzt ist. Die Beschleunigung des Fahrzeuges in Längsrichtung $\dot{v}$ ist ebenso durch die Maximalbeleunigung begrenzt. Es gilt: $|\dot{v}| \in [a_{min}, a_{max}]$, a_{min} ist eine negative Beschleunigung, die durch die maximale Bremskraft ensteht. Die Geschwindigkeit v ist ebenso limitiert $|v| < v_{max}$.

RRT

Die Performance des Sampline Based Ansatz aus 2.9.2 ist stark von der eingesetzten Metrik abhängig. Abhilfe schafft hier, wie auch bei holonomen Fahrzeugen in Kapitel 2.9.1, ein zufallsbasierter Ansatz. Im Gegensatz zu dem RRT Ansatz für holonome Fahrzeuge muss jetzt die Kinematik und oder Dynamik eines Fahrzeuges mitbetrachtet werden. Eine zufällig ausgewählte Konfiguration kann nicht mehr einfach mit einer Geraden, mit der dichtesten Konfiguration im Suchbaum verbunden werden. Statt dessen muss die Aktion gefunden werden, welche die zufällige Konfiguration am besten mit den Konfigurationen im Suchbaum

verbindet. Zudem ist die Suche nach der dichtesten Konfiguration im Suchbaum komplexer, da Konfigurationen durch Kurven verbunden sind. Als Fahrzeugmodell wird hier wieder das Fahrzeugmodell des *Sampling Based* Ansatztes aus Kapitel 2.9.2 verwendet. Ein weiteres Problem ist die Kollisionserkennung eines Pfades.



(a)

Abbildung 2.24: Erreichbarkeitsgraph des Dubins Fahrzeuges (grün) und ein gefundener Pfad (rot/schwarz).

Suche nach dem nächsten Punkt Für die Suche nach dem nächsten Punkt (der nächsten Konfiguration) im vorhandenen Suchbaum kann nicht mehr ein einfacher Algorithmus für die Abstandsbestimmung zwischen einer Geraden (einem Pfad) und einem Punkt oder zwischen zwei Punkten verwendet werden, da Pfade keine Geraden mehr sind. Eine mögliche Form der Approximation der Pfadsegmente ist es, das Pfadsegment in mehrere kleine Pfadsegmente zu unterteilen. Somit enstehen für ein ursprüngliches Pfadsegment im Baum mehrere kleine Pfadsegmente, die durch Kanten verbunden sind. In Abbildung 2.25 wird dieses verdeutlicht. Die Konfigurationen $q_1, q_2, \ldots, q_n$ sind die eigentlichen Endkonfigurationen, die durch Anwenden einer Aktion erreicht werden. Die Konfigurationen $q_{1_1}, q_{1_2}, \ldots, q_{1_6}$ approximieren den Verlauf des Pfades zwischen zwei Konfigurationen. Mithilfe dieser zusätzlichen

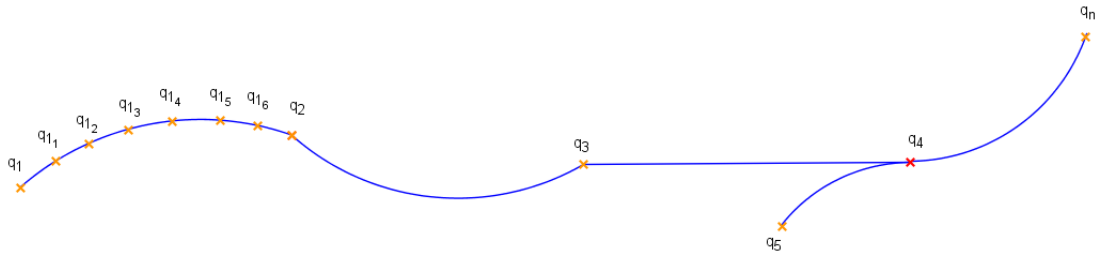
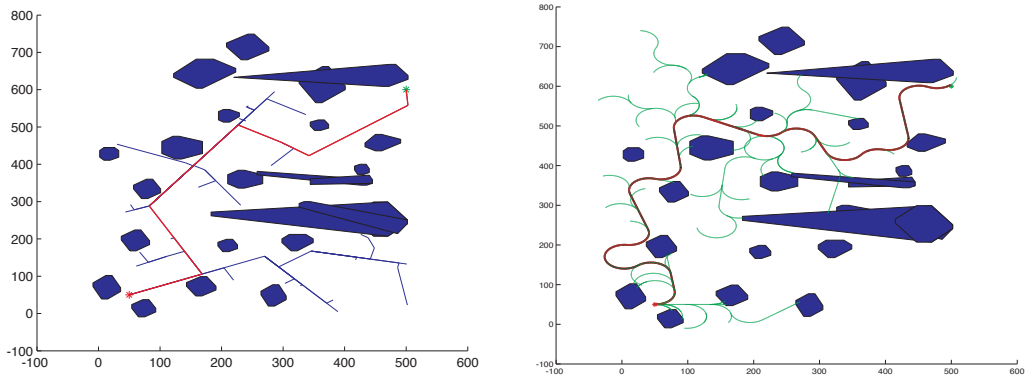


Abbildung 2.25: Ein Pfadsegment wird durch mehere Konfigurationen approximiert

Konfigurationen kann schnell eine nahe Konfiguration zu einer zufälligen neuen Konfiguration gefunden werden.

Kollisionerkennung Das Problem der Kollisionserkennung ist ähnlich, wie das der Suche nach dem nächsten Punkt. Der Ansatz aus 2.9.1 kann nicht genutzt werden, da es sich bei den Pfaden nicht um einfache Geraden handelt. Ein einfacher Ansatz ist es, die Pfadsegmente, wie bei der Suche nach dem nächsten Punkt, zu verwenden. Möglich wäre es, die einzelnen Zwischenkonfigurationen ($q_{1_1}, q_{1_2}, \ldots, q_{1_6}$ in Abbildung 2.25) zu testen, jedoch können so Hindernisse zwischen den Konfigurationen nicht erkannt werden. Ein anderer Ansatz ist es, die Pfadsegmente zwischen den Zwischenkonfigurationen auf Kollisionen zu testen. Dafür könnten die Pfadsegmente als Geraden angenommen werden. Diese Geraden werden mit den Algorithmen aus 2.9.1 auf Kollisionen getestet.



(a) Holonomer Pfad (b) Nicht holonomer Pfad

Abbildung 2.26: Der holonome RRT und der nicht holonome RRT im Vergleich. Der holonome RRT konnte nach 59 Iterationen einen Pfad finden. Der nicht holonome Pfad wurde nach 635 Iterationen gefunden.

In der Abbildung 2.26 sind die Ergebnisse von den beiden RRT Algorithmen im Vergleich

dargestellt. Während der holonome RRT bereits nach 59 Iterationen einen Pfad finden konnte, so dauerte es beim nicht holonomen RRT gut zehn mal so lange. Dieses liegt daran, dass das einfache Verbinden zweier Konfigurationen nicht möglich ist. Zudem spielt der Zufall eine große Rolle. Der holonome Pfad hat weniger Konfigurationen ausserhalb des Pfades besucht.

3 Koordination

Im vorherigen Kapitel wurde ausführlich beschrieben, wie ein Pfad für ein Fahrzeug in einer Welt geplant werden kann. Durch die Beschreibung des Pfadplanungsproblems im Konfigurationsraum konnte das spezielle Problem der Pfadsuche für einen Roboter abstrahiert werden. In diesem Kapitel werden die Probleme für die Pfadplanung beschrieben, die entstehen, wenn sich mehrere Roboter in einer gemeinsamen Welt bewegen. Kapitel 3.1 beschreibt allgemein die Probleme, die auftreten können, und welchen Bedingungen die Koordination genügen muss.

Es gibt eine Reihe von Konflikten, die zwischen zwei Fahrzeugen auftreten können. Der einfachste und häufigste Konflikt ist die Kollision. Dieses bedeutet, dass es einen Zeitpunkt gibt, an dem sich die Pfade zweier Fahrzeuge so nahe kommen, dass sie kollidieren werden. Da eine Kollision immer verhindert werden muss, enstehen neue Konflikte. Es kann zu *deadlocks* kommen. Diese enstehen zum Beispiel, wenn sich zwei Fahrzeuge in einer engen Gasse entgegenkommen und keines der Fahrzeuge die Möglichkeit hat, rückwärts aus der Gasse herauszufahren. Ein weiterer Konflikt ist die *congestion*. Diese tritt ein, wenn mehrere Fahrzeuge zum selben Zeitpunkt z.B. duch eine schmale Durchfahrt fahren wollen.

Definition 3.1. *Konflikte*
Ein Konflikt zwischen zwei Fahrzeugen besteht dann, wenn der Mindestabstand zwischen zwei Fahrzeugen zu einem Zeitpunkt unterschritten wird oder ein Fahrzeug aufgrund eines anderen seine Zielposition nicht erreichen kann.

3.1 Problemstellung

Zum Beispiel: es befinden sich mehrere Roboter in derselben Welt $\mathcal{W}$. So muss für jeden Roboter ein Pfad berechnet werden, der Kollisionen mit statischen Hindernissen und allen anderen Robotern vermeidet. Der Zustandsraum für die Suche nach Pfaden ist:

$$X = C^1 \times C^2 \times \ldots \times C^m \tag{3.1}$$

Dabei ist C^n der Konfigurationsraum des Roboters A^n. Ein Zustand $x \in X$ spezifiziert die Konfigurationen aller Roboter und wird als $x = (q^1, q^2, \ldots, q^m)$ dargestellt. Die Dimension von X ist N, welches die Summe $N = \sum_{i=1}^{m} \dim(C^i)$ ist.

Für jeden Roboter A^i ist ein Hindernis Raum X^i_{obs} gegeben:

$$X^i_{obs} = \{x \in X \mid A^i(q^i) \cap \mathcal{O} \neq \emptyset\} \tag{3.2}$$

Für jedes Paar von Robotern A^i und A^j kann ein weiterer Hindernis Raum definiert werden, der alle Zustände enthält, in denen die Roboter kollidieren:

$$X^{ij}_{obs} = \{x \in X \mid A^i(q^i) \cap A^j(q^j) \neq \emptyset\} \tag{3.3}$$

Somit kann das Problem allgemein wie folgt definiert werden:

Problem 3.1. *Pfadplanung für mehrere Roboter*

1. *Die Welt $\mathcal{W}$ und die Hindernis Region $\mathcal{O}$ sind wie in Problem 2.4 gegeben.*
2. *Eine Menge von Robotern, $A^1, A^2, \ldots, A^m$ ist gegeben.*
3. *Für jeden Roboter A^i, von $i = 1$ bis m, ist der Konfigurationsraum C^i spezifiziert.*
4. *Der Zustandsraum X ist das kartesische Produkt*

   $$X = C^1 \times C^2 \times \ldots \times C^m \tag{3.4}$$

   Der Hindernisraum ist

   $$X_{obs} = \left(\bigcup_{i=1}^{m} X^i_{obs}\right) \bigcup \left(\bigcup_{i,j,i\neq j} X^{ij}_{obs}\right) \tag{3.5}$$

5. *Ein Startzustand $x_i \in X$ ist gegeben, der die initialen Konfigurationen enthält $x_i = (q^1_i, q^2_i, \ldots, q^m_i)$.*
6. *Ein Zielzustand $x_g \in X$ ist gegeben $x_g = (q^1_g, q^2_g, \ldots, q^m_g)$.*
7. *Aufgabe ist es einen Pfad $\tau : [0, 1] \to X$ zu suchen, sodass $\tau(0) = x_i$ und $\tau(1) = x_g$.*

Die Koordination bzw. die Pfadplanung von mehreren Fahrzeugen kann nach Latombe [Latombe (1991)] grob in drei Kategorien eingeteilt werden: zentrales Planen, entkoppeltes Planen und priorisiertes Planen.

3.2 Ansätze zur Koordination

3.2.1 Zentrale Koordination

Der einfachste Ansatz ist es, die Pfade zentral zu berechnen. Dazu kann die Problemstellung 3.1 direkt umgesetzt werden. Ein Pfad kann ohne weiteres mit den Algorithmen aus dem Kapitel 2.2 gesucht werden. Jedoch ist zu beachten, dass die Dimension linear mit der Anzahl der Fahrzeuge steigt. Gilt $\dim(C^i) = 3$ für alle Roboter, so ist die Dimension des Zustandraumes $\dim(X) = 3m$, bei 10 Fahrzeugen ist die Dimension somit 30. Da die kompletten kombinatiorischen Ansätze aus Kapitel 2.5.2 eine Zeit für die Berechnung benötigen, die exponentiell mit der Anzahl der Dimension steigt, können diese hier nicht mehr angewandt werden. Sampling-Based Ansätze aus Kapitel 2.5.1 skalieren besser.

3.2.2 Entkoppelte Koordination

Anstatt in dem Zustandsraum aus Formel 3.1 nach einem Pfad zu suchen, kann auch ein *entkoppelter* Ansatz genutzt werden. Dabei wird für jeden Roboter A^i ein Pfad τ^i geplant. Andere Roboter werden in dem Pfad nicht betrachtet. Anschließend werden die Bewegungen (Aktionen) der Roboter so umgeplant, dass es keine Konflikte gibt. Der Ansatz ähnelt dem Ansatz aus Kapitel 2.6.2. Anstatt die Bewegungen (Aktionen) umzuplanen, kann auch ein neuer Pfad geplant werden, der jeweils andere Roboter als Hindernisse betrachtet.

3.2.3 Priorisierte Koordination

Um die Komplexität der Planung gering zu halten, ist es auch möglich, mittels Prioritäten eine Reihenfolge für die Pfadplanungen der einzelnen Roboter festzulegen. Es wird, beginnend bei dem Roboter mit der höchsten Priorität, für jeden Roboter ein Pfad geplant. Dabei betrachtet jeder Roboter alle Roboter mit höherer Priorität als Hindernisse. Somit können auf einfache Weise Pfade für eine große Anzahl von Fahrzeugen geplant werden. Jedoch verlagert sich das Problem der Pfadsuche auf die Suche nach einem Prioritätenschema, das es ermöglicht, jeden Roboter in seine Zielkonfiguration überführen zu lassen. Die Suche nach einem Prioritätenschema ist jedoch nicht trivial und lässt sich wiederum nur durch häufiges Neuplanen von Pfaden lösen.

3.2.4 Verteilte Koordination

Bei der verteilten Koordination wird die Komplexität des gesamten Pfadplanungsproblems auf die einzelnen Roboter verteilt. Anstatt zentral in einem hochdimensionalen Zustandsraum (vgl. Gleichung 3.1) einen Pfad zu suchen, sucht jeder Roboter für sich in seinem Zustandsraum einen Pfad. Diese Pfadsuche wird dabei in zwei Schritte eingeteilt. Im ersten Schritt wird ein Pfad unter der Berücksichtigung von statischen Hindernissen geplant. Andere Roboter werden in diesem Schritt explizit ignoriert. In einem zweiten Schritt wird überprüft, ob es Konflikte zwischen Robotern gibt. Bei der verteilten Koordination handelt es sich somit um *entkoppelte* Ansätze. Es besteht die Möglichkeit, den bereits geplanten Pfad zu modifizieren oder einen neuen Pfad zu planen. Werden neue Pfade geplant, so ensteht ein *priorisierendes* System.

Kommunikation

Durch die Verteilung des Pfadplanungsproblems auf eine Vielzahl von autonomen Robotern, ist eine Kommunikation nötig. Bei zentralen Ansätzen sind alle nötigen Informationen immer schnell und in der nötigen Qualität vorhanden. Bei dem verteilten Ansatz ist dieses nicht der Fall. Die Roboter können nicht ohne weiteres den geplanten Pfad eines anderen Roboters einsehen.

Wichtig für die Kommunikation ist, dass die Sende- und Empfangszeiten für eine Nachricht gering gehalten werden. Dieses ist nur dadurch möglich, dass die Nachrichten-Datenmenge klein gehalten wird.

Das Versenden von Statusnachrichten ist in der Regel kein Problem, da diese eine feste Länge haben. Jedoch ist das Versenden von geplanten Pfaden nicht trivial. Hier stellt sich, wie schon im Kapitel 2.9.2, die Frage nach der Auflösung. Handelt es sich um holonome Fahrzeuge, die sich auf Geraden bewegen, so reicht es, die Eckpunkte des Pfades zu übermitteln. Handelt es sich hingegen um nicht holonome Fahrzeuge, so bestehen die Pfade aus Kurven und Geraden. Bei den Kurven ist es nicht möglich, nur den Start und Endpunkt zu übermitteln, da die Informationen über die Steigung der Kurve verloren gehen würden.

Die Darstellung von Pfaden bzw. Trajektorien ist in Kapitel 3.3.1 weiter beschrieben.

Trajektorienkoordination

Ziel der Trajektorienkoordination ist es, die Trajektorien zweier Fahrzeuge so anzupassen, dass es keine Konflikte zwischen diesen gibt. Es gibt dabei zwei Arten von Ansätzen. Zum

einen kann die Geschwindigkeit über die Trajektorie modifiziert werden. Dieses hat zur Folge, dass Konfigurationen des Pfades zu einem früheren oder späteren Zeitpunkt erreicht werden.

Es gibt aber auch den Ansatz, die Trajektorie komplett neu zu planen und dabei die Trajektorie eines anderen Fahrzeuges als Hindernis zu berücksichtigen. Jeder dieser Ansätze birgt Vor- und Nachteile. So ist der geschwindigkeitsanpassende Ansatz nicht komplett, da nicht immer eine Lösung gefunden werden kann, obwohl eine Lösung exsistiert. Dieses ist der Fall, wenn ein Fahrzeug dauerhaft den Pfad eines anderen Fahrzeuges blockiert, da das Fahrzeug das stehende Fahrzeug nicht umfahren kann. Es kann mit sehr wenig Rechen- und Kommunikationsaufwand eine optimale Lösung berechnet werden. Ansätze, bei denen die Trajektorie neu geplant wird, haben den Vorteil, dass sie komplett sind, wenn der Pfadplanungsalgorithmus komplett ist. Jedoch ist die Suche nach einer alternativen Trajektorie aufwändig und benötigt daher mehr Zeit.

Beide Ansätze können sowohl entkoppelt als auch priorisierend umgesetzt werden. Die Grenzen zwischen entkoppelten und priorisierenden Ansätzen verschmelzen jedoch schnell. Bei vielen entkoppelten Ansätzen wird ein Priorisierungschema gesucht, nach dem die Fahrzeuge dann priorisiert werden. Durch eine Priorisierung wird immer ein Fahrzeug benachteiligt. In den meisten Ansätzen ist das Fahrzeug benachteiligt, welches einen besseren Alternativpfad findet. Dieser ist jedoch immer weniger optimal als der ursprüngliche Pfad.

In dem Ansatz für die Koordination in dieser Arbeit, der in Kapitel 3.3 beschrieben ist, können beide Fahrzeuge ihre jeweilige Trajektorie anpassen. Es wird dabei kein Fahrzeug priorisiert. Die Koordination ist bezüglich einer Gesamtbewertungfunktion optimal und nicht nur für ein Fahrzeug optimal.

3.3 Umsetzung

Für die Koordination ist ein Protokoll entwickelt worden, mit dessen Hilfe sich Fahrzeuge koordinieren können. Im Grunde basiert der Ansatz auf dem zweistufigen Ansatz aus Kapitel 2.6.2, bei dem die Geschwindigkeiten der einzelnen Fahrzeuge so angepasst werden, dass es zu keiner Kollision kommt.

Der in 2.6.2 beschriebene Ansatz beschreibt, wie für ein Fahrzeug die Geschwindkeit so angepasst werden kann, dass dieses mit keinem weiteren Fahrzeug kollidiert. Dabei wird ein Zustandsraum aufgebaut, in dem die Kollisionen als Hindernisse vorhanden sind. Der Suchraum wird anschließend mittels den Algorithmen aus 2.2.2 durchsucht. Dabei wird keinerlei Rücksicht auf die Fahrzeugkinematik und -dynamik genommen.

Für den implementierten Ansatz ist vorausgesetzt worden, dass für jedes Fahrzeug ein Pfad mit den Algorithmen aus 2.9 geplant worden ist. Mittels einer Konflikterkennung ist festgestellt worden, dass es bei dem Abfahren der Pfade zu Konflikten kommen wird. In vielen in der Literatur beschriebenen Ansätzen wird der Pfad eines Fahrzeuges so angepasst, dass die Konflikte gelöst werden. In dem implementierten Ansatz passen alle Fahrzeuge ihre Geschwindigkeiten an. Dieses ist der große Unterschied des neuen Ansatzes zu den anderen. Wird nur die Geschwindigkeit eines Fahrzeuges angepasst, so impliziert dieses eine Priorisierung eines Fahrzeuges. Die Koordination der Fahrzeuge soll möglichst optimal bezüglich einer verteilten Bewertungsfunktion sein. Dieses ist in dem Fall der Priorisierung eines Fahrzeuges jedoch nicht möglich, da sich nur ein Fahrzeug optimal verhält, die Gesamtheit der Fahrzeuge insgesamt jedoch nicht.

In der Problemdefinition 2.4 ist ein Pfad τ durch $\tau : [0,1] \rightarrow C_{free}$ definiert worden. In Kapitel 2.6 ist ein Pfad um die Zeit auf $\tau : [0,1] \rightarrow C \times \mathcal{T}$ erweitert. Der erweiterte Pfad wird als Trakjektorie bezeichnet, da er abhängig von der Zeit ist. Im weiteren wird der Definitionsbereich $[0,1]$ von τ in dieser Form nicht weiter benötigt. Anstelle dessen wird eine Trajektorie Tr wie folgt definiert:

Definition 3.2. *Trajektorie*

$$Tr : [0,t] \rightarrow C_{free} \tag{3.6}$$
$$\tau(1) = (q_g, t) \tag{3.7}$$

Dabei ist t der Zeitpunkt, an dem das Fahrzeug die Zielkonfiguration q_g erreicht. Dieses ist der Fall bei $\tau(1)$. Im Unterschied zum Pfad ist bei der Trajektorie der Definitionsbereich das Zeitintervall, über das sich das Fahrzeug bewegt. Die Zeit ist nicht mehr Bestandteil der Zielmenge von Tr.

Der neue verteilte Ansatz ist folgendermaßen definiert:

Ansatz 3.1. *Verteilte Geschwindkeitsanpassung zur Koordination*

1. *Es ist eine Menge von Fahrzeugen $A^1, A^2, \ldots, A^m$ gegeben.*
2. *Für jedes Fahrzeug ist eine Trajektorie $Tr^1, Tr^2, \ldots, Tr^m$ gegeben. Die Trajektorien führen jedes Fahrzeug von der Startkonfiguration q_i zu der Zielkonfiguration q_g ohne Kollisionen mit der Hindernisregion $\mathcal{O}$. Die anderen Fahrzeuge werden ignoriert.*
3. *Für jedes Fahrzeug ist eine Bewertungsfunktion $b : (Tr, Tr_m) \rightarrow [0,\infty]$ definiert. Diese bewertet die veränderte Trajektorie Tr_m bezüglich der ursprünglichen Trajektorie Tr.*

4. *Die Fahrzeuge werden so koordiniert, dass die Summe der Bewertungen b_g möglichst gering ist.*

$$b_g(A^i, \ldots, A^j) = \min \sum_{k_i}^{j} b^k(Tr^k, Tr_m^k) \tag{3.8}$$

Im Folgenden wird die Umsetzung der wichtigsten Komponenten der Koordination beschrieben.

3.3.1 Trajektorienbeschreibung

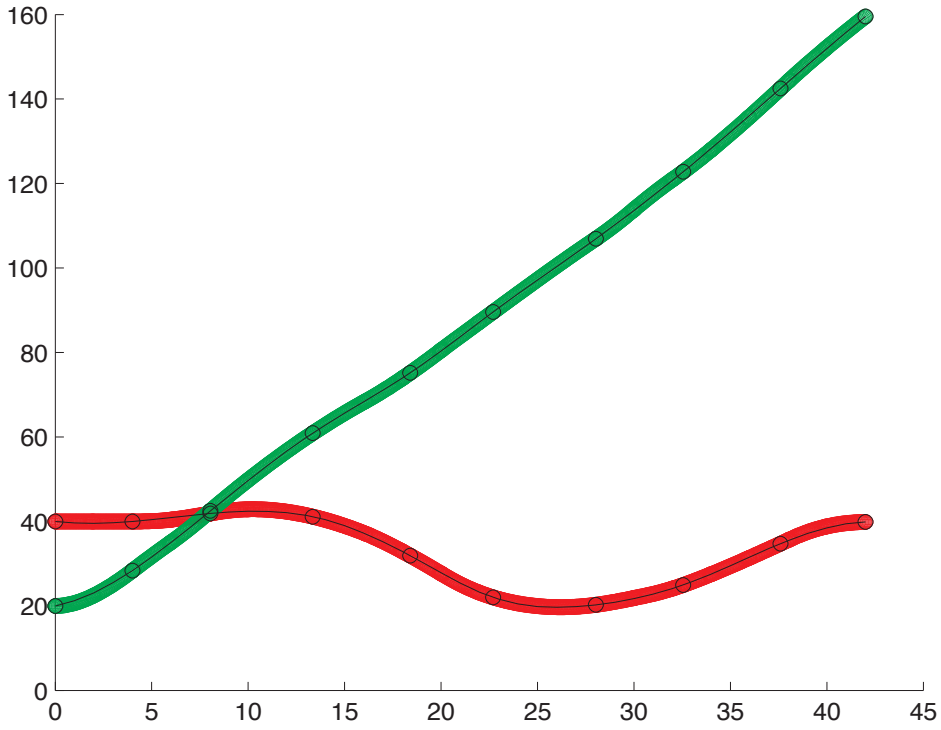


Abbildung 3.1: Der Verlauf der X- (rot) und Y- (grün) Koordinate über die Zeit. Der schwarze Strich ist die Approximation mit Splines, die Kreise sind die übertragenen Konfigurationen

Jedes Fahrzeug erhält durch die Pfadplanung eine Pfadbeschreibung, die das Fahrzeug von der Start- in die Zielkonfiguration überführt. Die Sampling Based, zu denen auch die RRT Ansätze gehören, liefern dabei für nicht holonome Fahrzeuge eine Folge von Aktionen, die über eine in der Aktion festgelegten Zeit angewandt werden. Diese Folge ist die Aktionstrajektorie

$\tilde{u} : [0, 1] \rightarrow U$ aus Kapitel 2.8.1. Die Aktionen geben jedoch noch keine Aussage darüber, an welcher Stelle sich das Fahrzeug zu einem Zeitpunkt befindet. Dieses wird erst durch die Lösung der Zustandsübergangsfunktion möglich. Die Zustandsübergangsfunktion ist jedoch nur dem jeweiligen Fahrzeug bekannt. Ein beliebiges Fahrzeug kann somit mit der Folge der Aktionen eines anderen Fahrzeuges keine Trajektorie berechnen. Für die Konflikterkennung ist es jedoch nötig, dass ein Fahrzeug die Trajektorie der anderen Fahrzeuge kennt. Es ist also ein einheitliches Format für die Trajektorienbeschreibung notwendig.

Ein Ansatz ist es, die bei der Pfadplanung genutzten Konfigurationen (inkl. der Zwischenkonfigurationen) als Sequenz von Punkten als Trajektorienbeschreibung zu nutzen (vgl. Kapitel 2.9.2 und 2.9.2). Dieses kommt der Approximation einer Trajektorie durch ein Polygon gleich. Da die Trajektorienbeschreibung zwischen den Fahrzeugen ausgetauscht werden muss, ist diese Art der Beschreibung jedoch ungeeignet, da sehr viele Daten auszutauschen sind. Möglich ist es, nur eine geringe Anzahl von Konfigurationen zu übertragen und diese anschließend durch Splines zu approximieren. Die so enstehenden Trajektorien sind sehr nah an den realen Trajektorien, die Menge der zu übertragenden Daten ist jedoch deutlich geringer. Eine durch wenige Punkte diskretisierte Trajektorie ist wie folgt definiert:

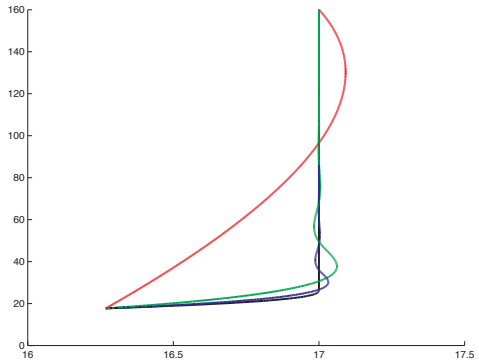
$$Tr_d = ((q_1, t_1), (q_2, t_2), \ldots, (q_m, t_m)) \tag{3.9}$$

Dabei sind $q_1, q_2, \ldots, q_3$ die Konfigurationen (Positionen) eines Fahrzeuges und $t_1, t_2, \ldots, t_m$ die dazugehörigen Zeitpunkte. Aus der Folge von gezeiteten Konfigurationen werden mittels der Splineinterpolation für jede Dimension der Konfigurationen ein Spline S erstellt. Eine Trajektorie besteht somit aus meheren Splines bzw. einem mehrdimensionalen Spline:

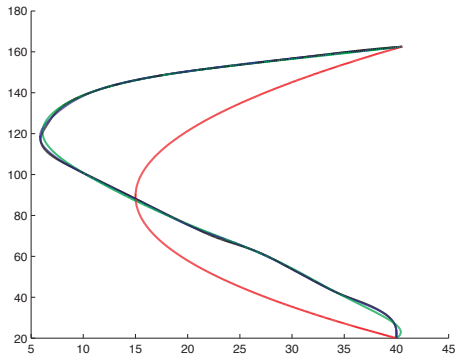
$$Tr(t) = \begin{pmatrix} S_x(t) \\ S_y(t) \end{pmatrix} \tag{3.10}$$

Die Ausrichtung (dritte Dimension) wird im Folgenden nicht betrachtet und kann daher ignoriert werden.

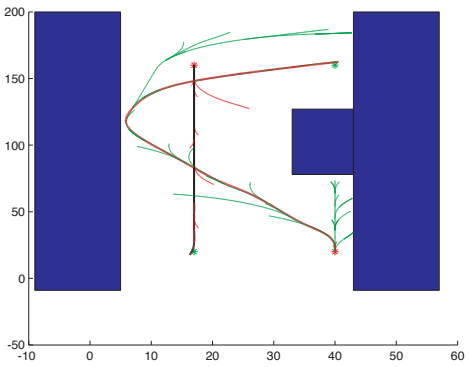
Die Abbildung 3.1 zeigt eine Trajektorie, aufgeteilt in die X- und Y- Koordinate in Abhängingkeit von der Zeit. Die schwarzen Kreise sind die übertragenen Daten. In diesem Fall wurden 10 Konfigurationen übertragen. Mittels Splines wurden die Konfigurationen approximiert. Somit ist die in schwarz dargestellte Trajektorie enstanden.



(a) Trajektorie des ersten Fahrzeuges



(b) Trajektorie des zweiten Fahrzeuges



(c) Das Result im Arbeitsraum

Abbildung 3.2: Interpolation zweier Trajektorien (schwarz) durch Spilnes mit 3(rot), 10(grün) und 15(blau) Stützstellen

In Abbildung 3.2(a) und 3.2(b) sind verschiedene Pfadapproximationen dargestellt. In schwarz ist jeweils der real geplante Pfad dargestellt. In rot wurde der Pfad jeweils durch nur 3 Stützstellen übertragen. Der daraus resultierende Pfad hat nur noch wenig mit dem ursprünglichen Pfad zu tun. Der grüne Pfad wurde durch die Übertragung von 10 Stützstellen berechnet. Der blaue durch 15. Beide Pfade approximieren den urprünglichen mit einer ausreichenden Genauigkeit. Abbildung 3.2 zeigt beide Trajektorien im Arbeitsraum.

3.3.2 Konflikterkennung

Die Konflikterkennung hat die Aufgabe, alle Konflikte, die in Definition 3.1 beschrieben sind, zu erkennen. Ein Konflikt ist immer vorhanden, wenn es einen Zeitpunkt gibt, an dem sich zwei Fahrzeuge berühren. Dieses ist der Fall, wenn für einen Zeitpunkt für zwei Fahrzeuge A^i und A^j gilt:

$$A^i(q^i) \cap A^j(q^j) \neq \emptyset \tag{3.11}$$

Dieses ist mit der Bedingung in Gleichnung 3.3 äquivalent. Zur Vereinfachung wird in der Implementierung ein Fahrzeug für die Kollisionsprüfung als kreisförmig angenommen. Die Orientierung wird somit nicht betrachtet. Dadurch ist es möglich, die Gleichung 3.11 zu vereinfachen:

$$\sqrt{(q_x^i - q_x^j)^2 + (q_y^i - q_y^j)^2} \leq r^i + r^j \tag{3.12}$$

Gilt diese Gleichung für alle Zeitpunkte, so besteht kein Konflikt zwischen den zwei Fahrzeugen A^i und A^j. r^j und r^j sind die jeweiligen Radien der Fahrzeuge.

Für beide Fahrzeuge sind die Trajektorien Tr^i und Tr^j bekannt. Die Trajektorien sind, wie in Kapitel 3.3.1 beschrieben, als Splines vorhanden. Für die Berechnung der Konfliktzeitpunktes sind nach 3.12 die Differenztrajektorie bzw. die Differenzsplines nötig.

$$\Delta Tr = Tr^i(t) - Tr^j(t) \Rightarrow \tag{3.13}$$

$$\Delta S_x(t) = S_x^i(t) - S_x^j(t) \tag{3.14}$$

$$\Delta S_y(t) = S_y^i(t) - S_y^j(t) \tag{3.15}$$

S_x^i gibt dabei den Spline des Fahrzeuges A^i an, der die Position entlang der X-Achse über die Zeit beschreibt. Entprechend auch für die anderen Fahrzeuge und Achsen.

Um die Konfliktzeitpunkte zu finden, sind die Nullstellen R zu berechnen.

$$R = \{t \in \mathbb{R} \mid \Delta S_x(t)^2 + \Delta S_y(t)^2 - (r^i + r^j)^2 = 0\} \tag{3.16}$$

Der Verlauf ist in Abbildung 3.3 in rot dargestellt.

Für die Koordination ist zunächst nur der erste Zeitpunkt nötig, in dem ein Konflikt auftritt, da sich durch Veränderungen an den Pfaden alle weiteren Konflikte verschieben oder aufheben. Der erste Konfliktzeitpunkt t_c ist somit:

$$t_c = \min\{R\} \tag{3.17}$$

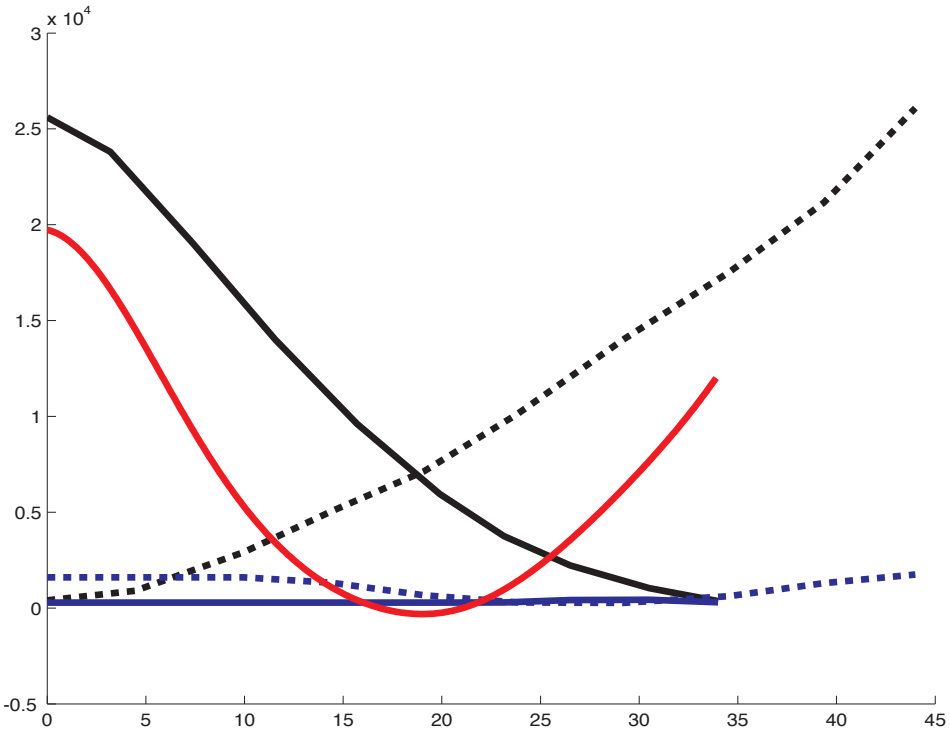


Abbildung 3.3: Berechnung der Konfliktzeitpunkte. In schwarz sind die Verläufe der X-, in blau der Y- Koordinaten zweier Fahrzeuge dargestellt. Der rote Graph ist der Verlauf von $\Delta S_x(t)^2 + \Delta S_y(t)^2 - (r^i + r^j)^2$. Nullstellen dieses Graphen sind die Begrenzungen von Konfliktintervallen.

3.3.3 Trajektorienmodifikation

Für die verteilte Trajektorienmodifikation ist es wichtig, dass diese auf allen Fahrzeugen, ohne Wissen über die dynamischen oder kinematischen Beschränkungen, durchführbar ist. Ansonsten würden sehr viele Trajektorien und damit eine Menge von Daten ausgetauscht werden. Wie im Ansatz 3.1 beschrieben, ist es den Fahrzeugen nur möglich, die Geschwindigkeiten zu verändern. Das Verändern des Pfades an sich ist nicht erlaubt. Eine Geschwindigkeitsänderung führt in der Trajektorie zu einer Dehnung oder Stauchung in der Zeit. Diese Veränderung ist in allen Dimensionen gleich. In Abbildung 3.4 ist dieser Zusammenhang verdeutlicht. Der Verlauf der X- Koordinate über die Zeit ist hier in schwarz in der urprünglichen Form dargestellt. In rot ist der Verlauf dargestellt, der ensteht, wenn sich das Fahrzeug mit der vierfachen Geschwindigkeit bewegt. Sei s der Faktor, um den die Geschwindigkeit erhöht

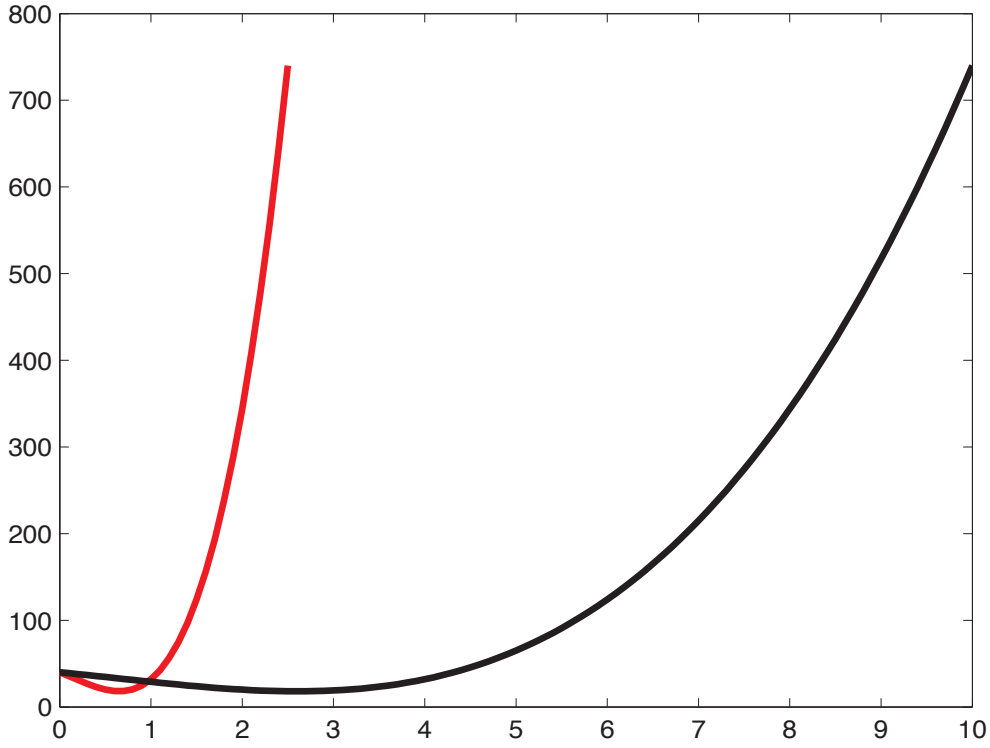


Abbildung 3.4: $S_x(t)$ einer Trajektorie. In schwarz der urprüngliche Verlauf. In rot derselbe Verlauf, jedoch um den Faktor 4 beschleunigt.

wird, so gilt für die modifizierte Trajektorie Tr_m:

$$Tr_m(t) = Tr(t * s) \tag{3.18}$$

Es entsteht somit eine Trajektorie, die in $1/s$ der Zeit der ursprünglichen Trajektorie zurückgelegt wird. Auf diese Weise kann eine Trajektorie erstellt werden, die von einem Fahrzeug mit einer um den Faktor s veränderten Geschwindigkeit abgefahren werden kann. Jedoch soll in der Regel für die Koordination nicht die gesamte Trajektorie verändert werden, sondern bestimmte Bereiche. Erzielt werden kann dieses durch die Einführung einer Zeitskalierungsfunktion σ, die bereits in Kapitel 2.6.2 beschrieben wurde. Da jedoch die Trajektorie in Form von Splines vorliegt, kann die Geschwindigkeitsänderung direkt in die Trajektorie eingebunden werden.

Eine Geschwindigkeitsänderung darf für die Koordination nur in Teilintervallen (zwischen den Stützstellen) der Splines vorgenommen werden. Somit kann jedes Fahrzeug die modifizierte Trajektorie eines anderen Fahrzeuges, ohne Wissen über dieses, berechnen.

Für den Fall, dass eine Trajektorie nicht durch Splines, sondern durch ein Polynom beschrieben ist, ist die Berechnung einer um den Faktor s beschleunigten/abgebremsten Trajektorie sehr einfach möglich. Es wird die Operation $\otimes$ eingeführt, die eine Trajektorie skaliert.

$$Tr(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0 \tag{3.19}$$
$$Tr_m(t) = Tr(t * s) \Rightarrow \tag{3.20}$$
$$Tr_m = Tr \otimes s \Rightarrow \tag{3.21}$$
$$Tr_m(t) = s^n a_n t^n + s^{n-1} a_{n-1} t^{n-1} + \ldots + s a_1 t + a_0 \tag{3.22}$$

In der Abbildung 3.4 ist die Skalierung für das Polynom $S_x(t) = 1t^3 - 2t^2 - 1t + 40$ gezeigt. Durch die Skalierung mit dem Faktor $s = 4$ ensteht das Polynom $S_{mx}(t) = 64t^3 - 32t^2 - 40t + 40$

Für die Splinedarstellung von Trajektorien kann dieses ebenfalls genutzt werden. Ein Spline besteht aus Teilintervallen, die jeweils durch ein Polynom interpoliert werden. Ein Spline S besteht somit aus einer Folge von Zeiten $breaks(S) = (t_0, t_1, \ldots, t_n)$, die die Teilintervalle begrenzen, und einer Folge von Polynomen $coefs(S) = (p_0, p_1, \ldots, p_{n-1})$, die den Verlauf über ein Teilintervall $[t_0, t_1]$, $[t_1, t_2]$, $\ldots$, $[t_{n-}, t_n]$ angeben.

Die modifizierte Trajektorie Tr_m wird somit wie folgt berechnet. Sei s der Geschwindigkeitsfaktor, der im m-ten Intervall der Trajektorie Tr angewandt werden soll, gilt für die Polynome der Splines S bzw. S_m in Tr bzw Tr_m:

$$p_m \in coefs(S) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0 \tag{3.23}$$
$$pm = p_m \otimes s = s^n a_n t^n + s^{n-1} a_{n-1} t^{n-1} + \ldots + s a_1 t + a_0 \tag{3.24}$$
$$coefs(S_m) = (p_0, \ldots, p_{m-1}, pm, p_{m+1}, \ldots, p_{n-1}),\ p_i \in coefs(S), i \in [0, n-1] \tag{3.25}$$

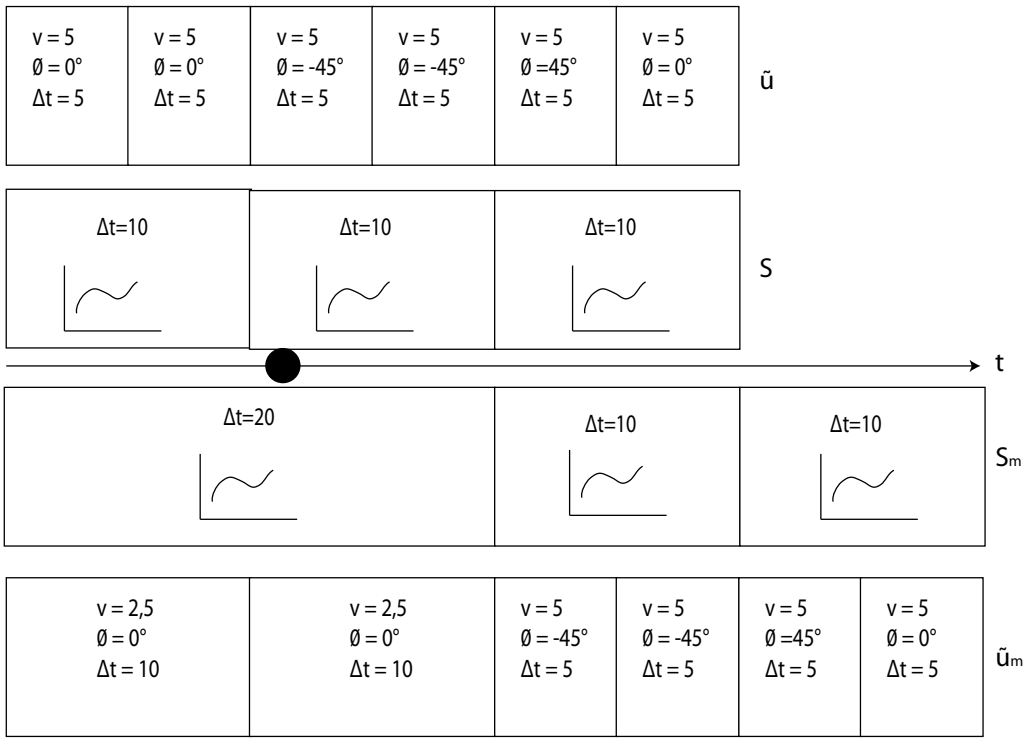
Für die Stützstellen bedeutet die Anwendung:

$$d = (t_{m+1} - t_m)/s - t_m \tag{3.26}$$
$$breaks(S_m) = (t_0, \ldots, t_m + d, t_{m+1} + d, \ldots, t_n + d),\ t_i \in breaks(S), i \in [0, n] \tag{3.27}$$

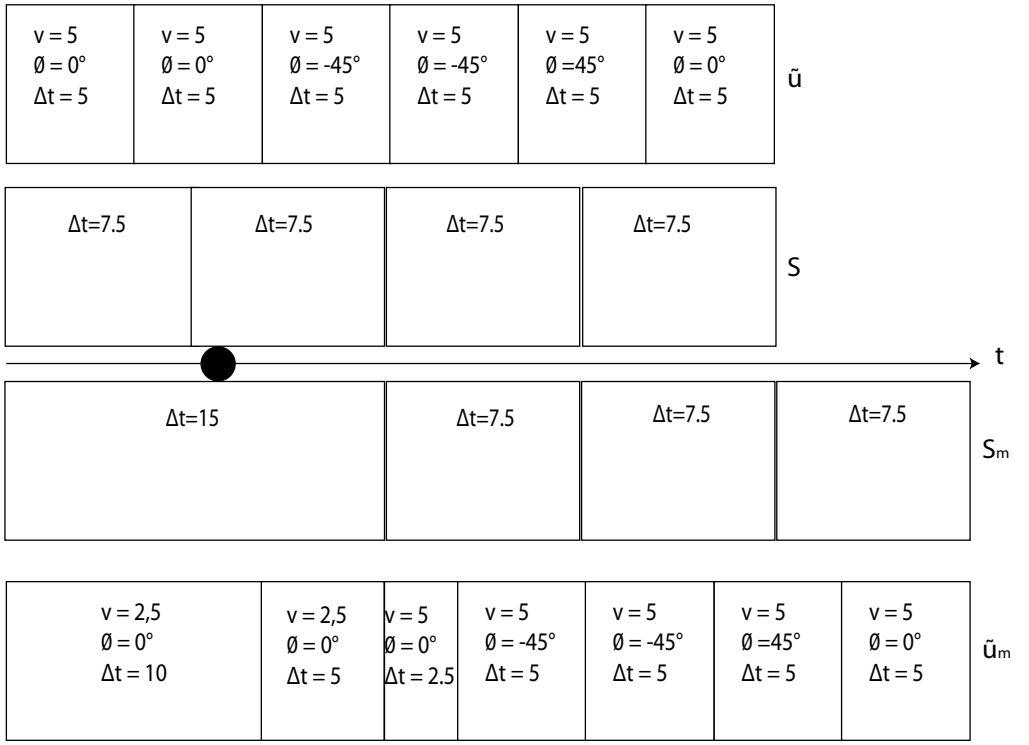
Die Zeitdifferenz d, die durch die Anwendung des Faktors s auf die Trajektorie ensteht, wird auf alle Zeiten der nachfolgenden Stützstellen addiert. Der Definitionsbereich von $D(Tr) = [t_0, t_n]$. Der Defintionsbereich von Tr_m ist um $(t_{m+1} - t_m)/s$ verändert $D(Tr_m) = [t_0, t_n + d]$. Was bewirkt, dass das Fahrzeug seine Zielposition bei $s > 1$ eher und bei $s < 1$ später erreicht.

In Kapitel 3.3.1 ist beschrieben, wie eine Aktionstrajektorie $\tilde{u}$ als Splines beschrieben wird. Dieses Kapitel beschreibt, wie Trajektorien modifiziert werden können. Der umgekehrte Weg

von einer als Spline beschriebenden modifizierten Trajektorie zu einer Aktionstrajektorie $\tilde{u}_m$ ist jedoch auch nötig. Ist eine Trajektorie, wie beschrieben, modifiziert worden, so sind diese Änderungen auch an der Aktionstrajektorie vorzunehmen. Da es sich bei den Modifikationen an der Trajektorie nur um Änderungen der Geschwindigkeit handelt, ist auch nur die Geschwindigkeitskomponente der Aktionstrajektorie anzupassen. Da die Stützstellen der Splineintervalle nicht mit den Aktionsgrenzen übereinstimmen müssen, werden neue Aktionen in die Aktionstrajektorie eingefügt. Dieses geschieht für die Zeitpunkte, für die es Splinestützstellen gibt, die an modifizierten Splineintervallen grenzen.



(a) Konfliktzeitpunkt $t_c = 11$, angewandter Faktor $s = .5$



(b) Konfliktzeitpunkt $t_c = 9$, angewandter Faktor $s = .5$

Abbildung 3.5: Die Aktionstrajektorie $\tilde{u}$ als Splines S und die nach der Modifikation enstandenden Splines S_m mit der angepassten Aktionstrajektorie $\tilde{u}_m$.

3.3.4 Bewertungsfunktion

Für die Koordination ist eine Bewertungsfunktion nötig, die Änderungen an Trajektorien bewertet. Dieses ist nötig, damit entschieden werden kann, wie ein Fahrzeug seine Trajektorie verändert. Im Gegensatz zu vielen anderen Ansätzen passen bei dem in dieser Arbeit beschriebenen Ansatz beide Fahrzeuge die Trajektorien an.

Die Bewertungsfunktion wird mit b bezeichnet und hat als Definitionsbereich (Tr, Tr_m):

$$b : (Tr, Tr_m) \rightarrow [0, \infty] \tag{3.28}$$

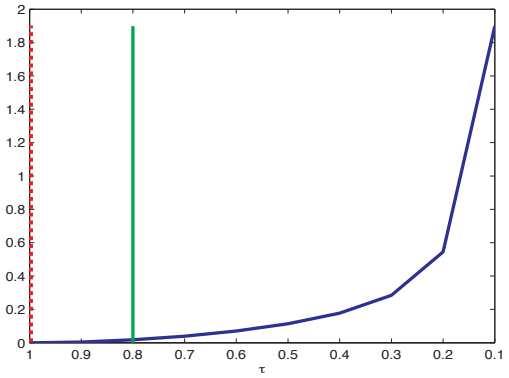
Die Funktion muss folgende Bedingung erfüllen:

$$b(Tr, Tr) = 0 \tag{3.29}$$

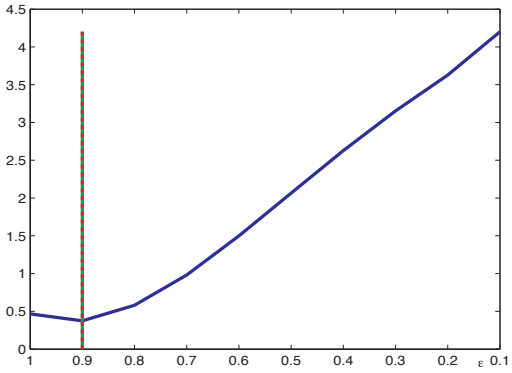
Sind beide Trajektorien gleich, so liefert die Funktion den Wert 0. Die Lösungsmenge von b ist auf $[0, \infty]$ begrenzt, da es keine negativen Bewertungen geben kann, da davon ausgegangen wird, dass die Trajektorien, die von der Pfadplanung erstellt worden sind, optimal sind. Dieses schließt ein, dass die optimale Geschwindigkeit zu jedem Zeitpunkt geplant wurde.

Faktoren, die in die Bewertungsfunktion einfließen können, sind z.B. Beschleunigungsänderungen, Energieverluste, die durch das Beschleunigungen oder Abbremsen entstehen, Veränderungen der Gesamtfahrtdauer und oder die Zunahme von Querbeschleunigungen. Die Bewertung kann zusätzlich von dem aktuellen Zustand des Fahrzeuges abhängig gemacht werden. Als Zustand kann hier z.B. gewertet werden, ob das Fahrzeug beladen ist oder ob es weitere Aufträge zu bearbeiten hat.

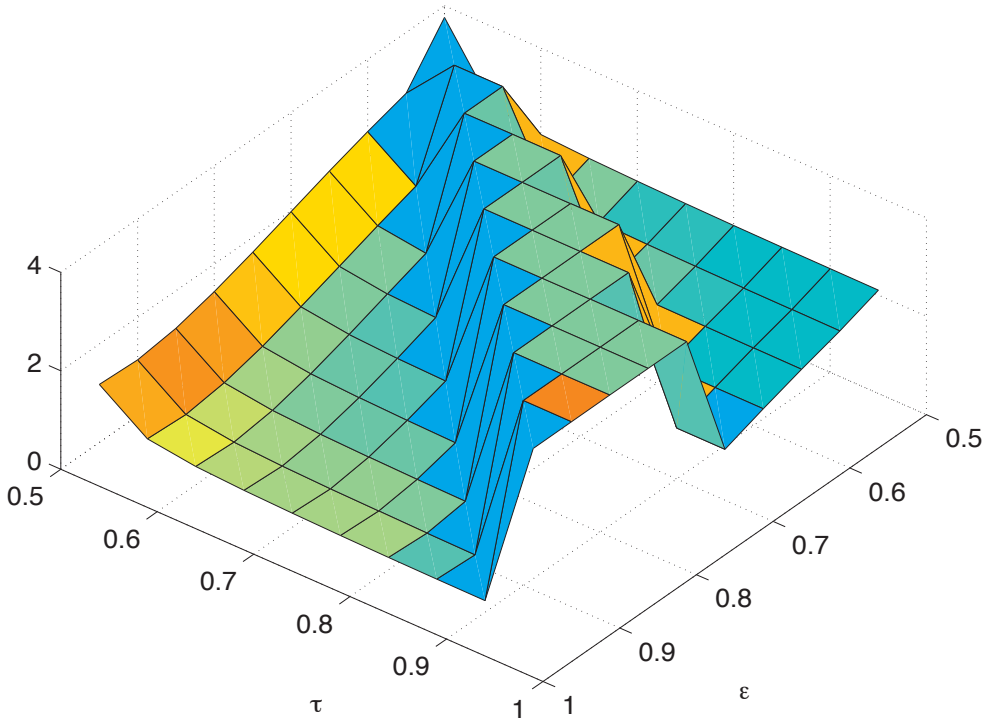
Wichtig bezüglich der Bewertungsfunktion ist, dass diese für jedes Fahrzeug individuell ist. Ein Fahrzeug hat somit nicht die Möglichkeit, Bewertungen für andere Fahrzeuge vorzunehmen.



(a) Bewertung $b^1(Tr^1, Tr^1 \otimes \tau)$ für das erste Fahrzeug



(b) Bewertung $b^2(Tr^2, Tr^2 \otimes \epsilon)$ für ein weiteres Fahrzeug



(c) Gesamtbewertung $b^1(Tr^1, Tr^1 \otimes \tau) + b^2(Tr^2, Tr^2 \otimes \epsilon)$

Abbildung 3.6: Ergebnisse verschiedener Bewertungsfunktionen und die Gesamtbewertungsfunktion, bei der alle Werte von 4 bedeuten, dass es zu einer Kollision kommt. τ und ϵ sind die Faktoren, die jeweils auf (einen Teil) der Trajektorie angewandt wurden.

In den Abbildungen 3.6(a) und 3.6(b) sind jeweils die Bewertungsfunktionen zweier Fahrzeuge dargestellt. Die X-Achse ist jeweils der Faktor s aus Kapitel 3.3.3. Für das erste Fahrzeug ist dieser mit τ, für das zweite Fahrzeug mit ϵ bezeichnet. Zudem sind in rot die optimalen Faktoren für das einzelne Fahrzeug und in grün die gemäß der Gesamtbewertung optimalen Faktoren eingezeichnet. Abbildung 3.6(c) zeigt die Summe der beiden Bewertungsfunktionen. Dabei sind alle Kombinationen von Faktoren, bei denen es zu einer Kollision kommt, mit dem Maximalwert versehen. Für die Koordination ist es nötig, das Wertepaar (τ, ϵ) zu finden, so dass die Gesamtbewertung möglichst optimal (gering) ist. In dem Fall, der in Abbildung 3.6 dargestellt ist, ist das optimale Faktorenpaar $(\tau, \epsilon) = (0.8, 0.9)$. Das erste Fahrzeug würde somit den Faktor 0.8 auf seine Trajektorie und das weitere Fahrzeug den Faktor 0.9 anwenden. Somit ist der Konflikt der Fahrzeuge gelöst.

Suche nach dem optimalen Faktorenpaar

Nach dem Ansatz 3.1 ist es nötig, das Faktorenpaar (τ, ϵ) zu finden, welches, angewandt auf die jeweiligen Trajektorien, die geringste Summe der Bewertungsfunktionen hervorruft.

Die in Abbildung 3.6(c) dargestellte Bewertungsfunktion ist für die Fahrzeuge nicht berechenbar, da die Bewertungen des jeweils anderen Fahrzeuges nicht bekannt sind. Durch eine Diskretisierung der Schritte, in der die Änderung der Trajektorie bewertet wird, kann die Anzahl der Daten, die übertragen werden müssen, gering gehalten werden, jedoch geht möglicherweise eine optimalere Lösung verloren. In Abbildung 3.6(c) sind die Gesamtbewertungen für die Faktorenpaare $(\tau, \epsilon) = \{(.1, .1), (.2, .1), \ldots (1, 1)\}$ dargestellt. Es kann somit nur Lösungen aus dieser Menge geben. Jedoch liegt die optimale Lösung nicht genau auf einem Wertepaar dieser Menge.

Es ist möglich, den Verlauf der Bewertungsfunktion mit einem Polynom zu interpolieren. Jedes Fahrzeug berechnet sich anhand mehrerer Faktoren den Verlauf der Bewertungsfunktion und interpoliert diesen durch ein Polynom dritten Grades. Das Resultat der Bewertungsfunktion kann somit mit sechs Werten beschrieben werden. Vier für das Polynom und zwei für den Definitionsbereich. Diese sechs Werte können einfach an andere Fahrzeuge gesendet werden.

Für die Berechnung des optimalen Faktorenpaares stehen somit jedem Fahrzeug der eigene und der Verlauf der Bewertungsfunkion des anderen Fahrzeuges zur Verfügung. Durch die Summenbildung ensteht auf beiden Fahrzeugen ein Graph, der dem aus Abbildung 3.6(c) gleich kommmt. Mit einem Algorithmus, der auf allen Fahrzeugen zum selben Ergebnis kommen muss, wird nun das optimale Faktorenpaar gewählt. Für die Suche ist es wichtig, dass nicht der komplette Bereich, in denen es zu Konflikten kommt, berechnet werden muss, da dieses zu einem großen Rechenaufwand führt.

Das Problem ist die Suche nach dem Minimum der Gesamtbewertungsfunktion b_g. Der Bereich, in dem es zu Konflikten kommt, bildet ein abgeschlossenes Intervall, da sich zwei Fahrzeuge unendlich nah kommen können, ohne dass ein Konflikt auftritt. Daher kann eine Lösung für den optimalen Faktor unendlich nah an dem Konfliktintervall liegen. Einfach zu berechnen lässt sich das optimale Faktorenpaar durch eine neue Diskretisierung. Wird der Bereich von τ und ϵ in jeweils zehn Schritte unterteilt, so sind 100 Faktorenpaare auf Konflikte zu prüfen und in den verbleibenden Faktorenpaaren das Faktorenpaar zu suchen, welches den geringsten Bewertungswert hat. Dieses Verfahren findet immer das optimale Wertepaar bezüglich der Auflösung der Faktoren.

Dieses Verfahren kann sogar noch beschleunigt werden. Die Berechnung des optimalen Faktors s_o ist einfach:

$$s_o = \min_s b(Tr, Tr \otimes s) \tag{3.30}$$

s_o ist also der Faktor, der die Bewertungsfunktion für ein einzelnes Fahrzeug minimiert.

Als Startpunkt für die Suche nach dem optimalen Faktorenpaar wird $x_i = (\tau_o, \epsilon_o)$ genommen. Bei der Anwendung dieser Initialfaktoren wird es mit sehr großer Wahrscheinlichkeit zu einem Konflikt kommen. Gesucht ist das Faktorenpaar, welches sich in der Nähe der Initialfaktoren befindet und keinen Konflikt mehr verursacht. Für die Suche nach diesem Faktorenpaar kann eine Vereinfachung des Dijkstra Algorithmus (Kapitel 2.2.2) genutzt werden. Die Kostenfunktion liefert in diesem Fall das Ergebniss von b_g der Gesamtbewertungsfunktion. Die Suche wird abgebrochen, wenn ein Faktorenpaar gefunden worden ist, welches keinen Konflikt verursacht. Die Anzahl der auf Konflikte zu testenden Faktorenpaare, kann somit minimiert werden.

3.3.5 Protokoll

In den bisherigen Kapiteln ist beschrieben, wie eine Trajektorie dargestellt wird (Kapitel 3.3.1), wie Konflikte zwischen zwei Trajektorien ermittelt werden können (Kapitel 3.3.2), wie eine Trajektorie modifiziert werden kann (Kapitel 3.3.3) und wie Änderungen an Trajektorien bewertet werden (Kapitel 3.3.4). Mit diesen Bestandteilen alleine ist das Problem der Koordination nicht lösbar. Die fehlende Komponente ist der Zusammenschluss aller Teile. Diese ist in diesem Kapitel beschrieben.

Es wird zugrundegelegt, dass alle Fahrzeuge miteinander kommunizieren können. Beispielsweise über *Wireless LAN*. Die in diesem Kapitel beschriebenen Nachrichten werden über das W-LAN von Fahrzeugen versendet und empfangen. Es ist jedoch auch möglich die Daten über ein anderes Kommunikationssystem, welches einen Broadcast ermöglicht, zu versenden.

Die Vergabe von Fahraufträgen an die Fahrzeuge ist nicht Bestandteil dieser Arbeit. Es wird davon ausgegangen, dass jedes Fahrzeug auf eine nicht näher beschriebende Weise Fahraufträge zugewiesen bekommt.

Für die Koordination von Fahrzeugen ist es notwendig, dass die Fahrzeuge, die möglicherweise im Konflikt zueinander stehen, voneinander wissen. Kennt ein Fahrzeug ein weiteres Fahrzeug nicht und es besteht ein Konflikt zwischen den Fahrzeugen, so ist die Lösung des Konfliktes nicht möglich, da das Fahrzeug den Konflikt nicht erkennt. Somit ist es nötig, dass die Fahrzeuge über das Umfeld und damit über alle Fahrzeuge in der nahen Arbeitswelt informiert sind. Für die Koordination sind folgende Daten über alle anderen Fahrzeuge nötig: aktuelle Position, aktuell geplante Trajektorie und die Dimension des Fahrzeuges.

Der Abgleich der Daten erfolgt über Nachrichten. Eine Übersicht über alle Nachrichten ist in der Tabelle 3.1 gegeben.

Nachricht	Inhalt
I	ID,Position, Trajektorienhash
II	ID
III	ID,Trajektorie, Fahrzeugdaten
IV	ID,Konfliktzeitpunkt
V	ID,Bewertungspolynom,Faktorintervall
VI	ID,Status

Tabelle 3.1: Übersicht über die Nachrichten, die zwischen Fahrzeugen versendet werden.

Die aktuelle Position jedes Fahrzeuges wird allen Fahrzeugen durch das periodische Aussenden der Nachricht I mitgeteilt. Diese Nachricht enthält zusätzlich einen Hash der aktuellen Trajektorie. Somit kann festgestellt werden, ob ein Fahrzeug eine neue Trajektorie geplant hat oder ob es seiner alten folgt.

Empfängt ein Fahrzeug eine Nachricht I, so wird geprüft, ob das Fahrzeug, welches die Nachricht gesendet hat, bereits bekannt ist, und ob der Hash der Trajektorie zu der dem Fahrzeug bekannten Trajektorie des anderen Fahrzeuges passt. Ist dieses nicht der Fall, so fordert das Fahrzeug mittels der Nachricht II die aktuelle Trajektorie des anderen Fahrzeuges an.

Empfängt ein Fahrzeug eine Nachricht II und ist die Anfrage darin an das Fahrzeug selbst gerichtet, so antwortet es mit der Nachricht III und sendet die aktuelle Trajektorie und die Fahrzeugdimensionen. Die Fahrzeuge, die die Nachricht III empfangen, aktualisieren, wenn nötig, die Daten über das Fahrzeug, dass die Nachricht gesendet hatte. Hat sich die Trajektorie eines Fahrzeuges geändert, so führen alle Fahrzeuge die Konflikterkennung aus Kapitel 3.3.2 durch.

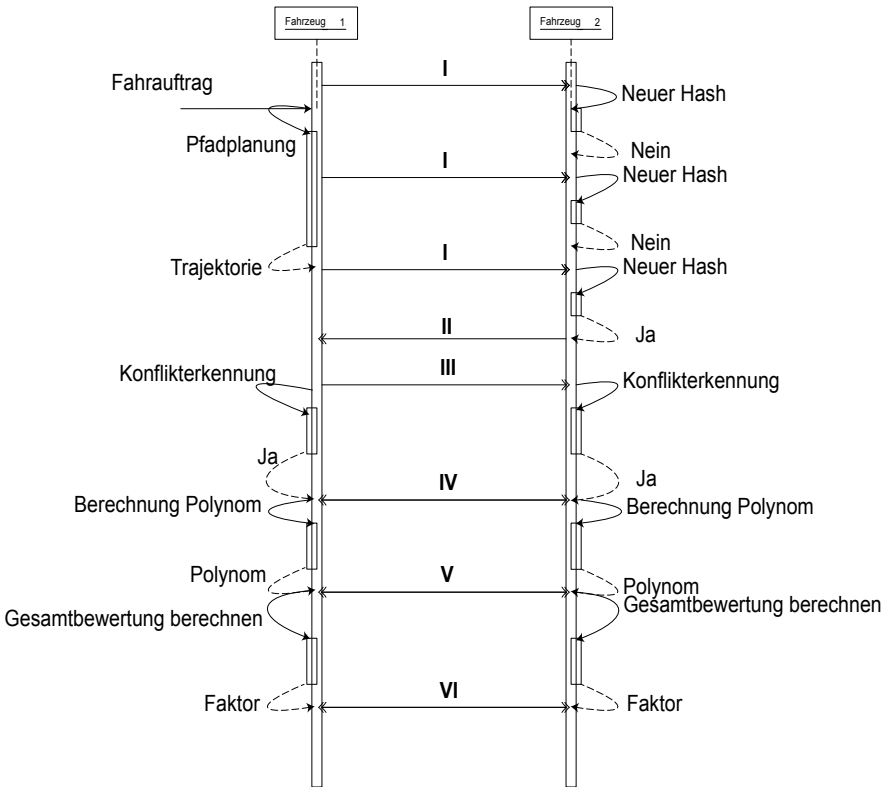


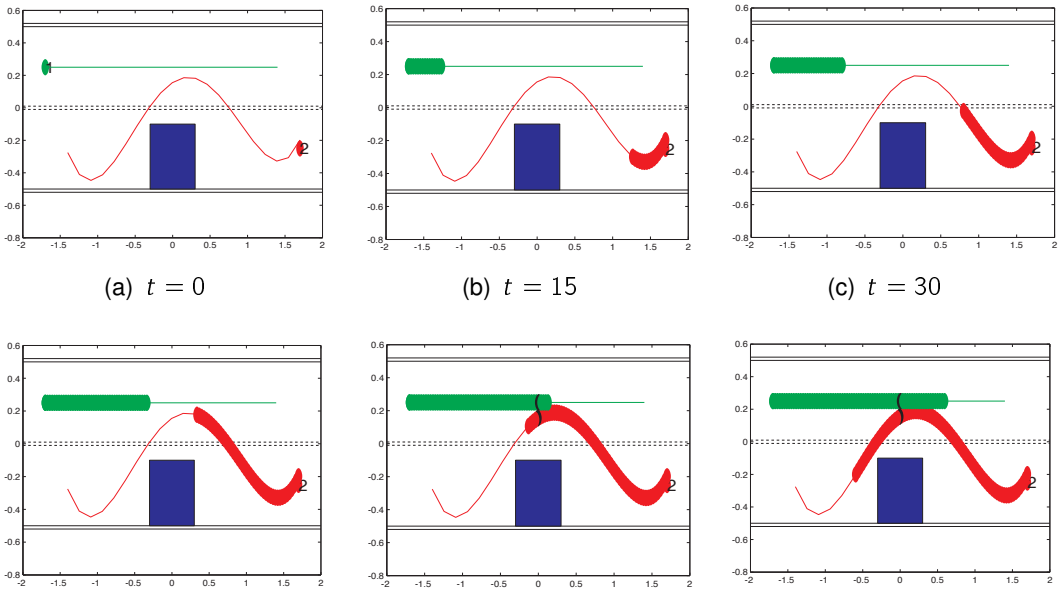
Abbildung 3.7: Das Protokoll als Sequenzdiagramm. Die Nachrichten I wurden für die Übersichtlichkeit nach 3 Übertragnungen entfernt.

Wird ein Konflikt erkannt, so ist die Suche nach dem optimale Faktorenpaar (Kapitel 3.3.4) nötig. Ist ein Konflikt bei einem Fahrzeug erkannt worden, so teilt das Fahrzeug dem anderen Fahrzeug mit der Nachricht IV den Konfliktzeitpunkt mit. Da auch das andere Fahrzeug diesen Konflikt feststellen wird, wird auch das andere Fahrzeug die Nachricht IV versenden.

Hat ein Fahrzeug den Konflikt nicht festgestellt, so wird dieses durch die Nachricht IV über einen Konflikt benachrichtigt. Empfängt ein Fahrzeug die Nachricht IV, so testet dieses, ob der Konflikt bereits bekannt ist und ob der Lösungsvorgang bereits begonnen hat. Ist dieses nicht der Fall, wird der Lösungsvorgang eingeleitet. In einem ersten Schritt werden durch die Trajektorienmodifikation (Kapitel 3.3.3) eine Menge von Trajektorien durch die Anwendung von Faktoren berechnet. Diese Trajektorien werden durch die Bewertungsfunktion (Kapitel 3.3.4) bewertet und die Resultate der Bewertung als Polynom interpoliert. Dieses Polynom wird dem anderen Fahrzeug mit der Nachricht V mitgeteilt.

Auf diese Weise sind beiden Fahrzeugen die Ergebnisse der Bewertungsfunktionen bekannt. Somit kann die Gesamtbewertungsfunktion b_g berechnet werden und das optimale Faktorenpaar nach dem Verfahren aus Kapitel 3.3.4 bestimmt werden.

Da alle Fahrzeuge denselben Algorithmus zur Berechnung der Faktoren nutzen, kommt es auch bei allen Fahrzeugen zu demselben Ergebnis, so dass dieses nicht mehr ausgetauscht werden muss. Lediglich wird mit der Nachricht VI bestätigt, dass ein Faktor gefunden wurde und die entsprechende modifizierte Trajektorie angewandt wird. Die Abbildung 3.7 zeigt das Protokoll als Sequenzdiagramm.

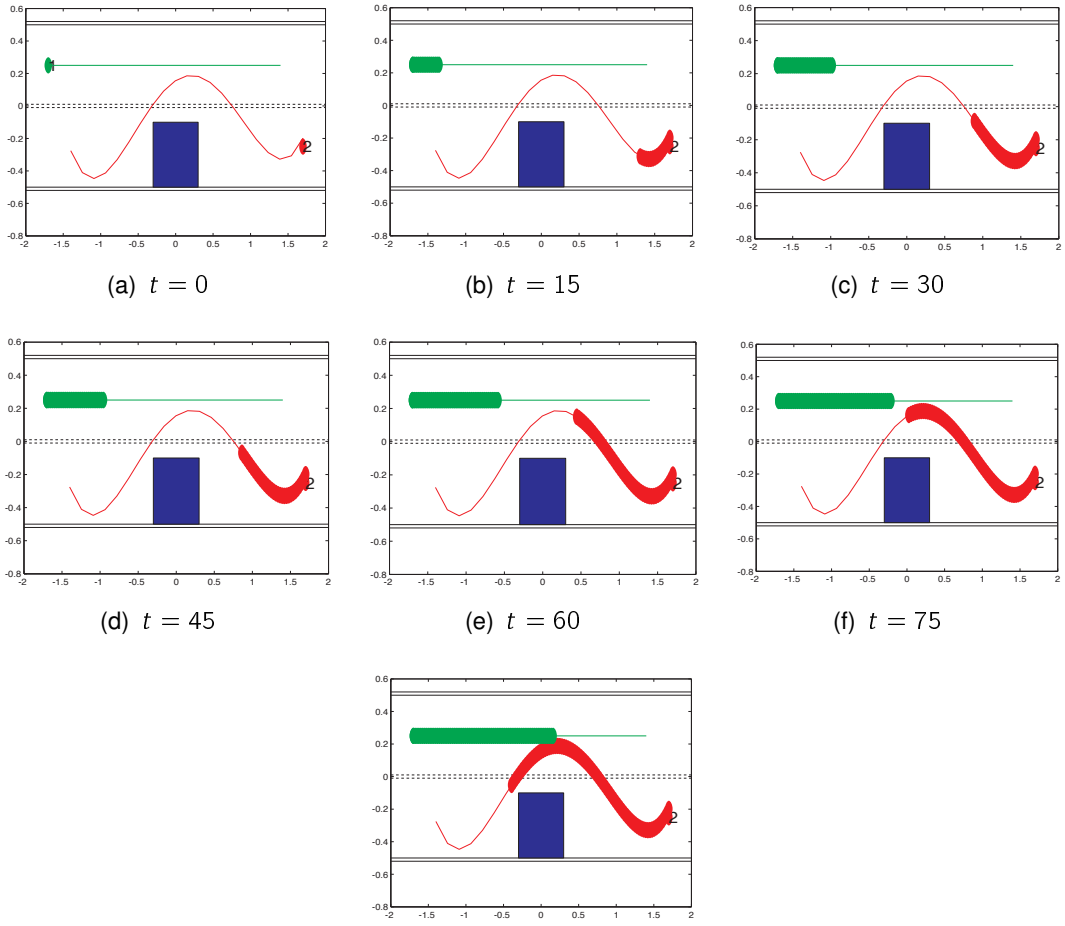


(a) $t = 0$ (b) $t = 15$ (c) $t = 30$

(d) $t = 45$ (e) $t = 60$ (f) $t = 75$

Abbildung 3.8: Die Fahrzeugpositionen ohne Koordination

Die Abbildung 3.8 zeigt den Verlauf von zwei Fahrzeugen ohne Koordination. Zum Zeitpunkt $t = 53$ kommt es zu einer Kollision. Nach der Bewertung der möglichen Modifikationen entsteht die Gesamtbewertungsfunktion aus Abbildung 3.10. Die Suche nach dem optimalen Faktorenpaar liefert $(0.8, 0.9)$. Durch die Anwendung der Faktoren auf die jeweilige Trajektoie, enstehen die Verläufe in Abbildung 3.9.



(a) $t = 0$ (b) $t = 15$ (c) $t = 30$

(d) $t = 45$ (e) $t = 60$ (f) $t = 75$

(g) $t = 100$

Abbildung 3.9: Die Fahrzeugpositionen nach der Anwendung der optimalen Faktoren aus der Gesamtbewertungsfunktion aus Abbildung 3.10

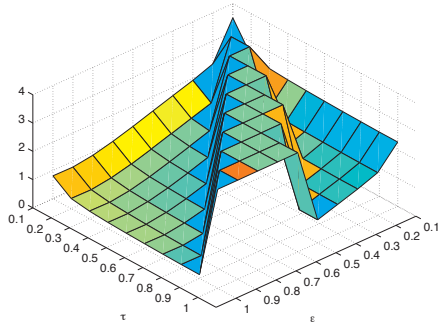


Abbildung 3.10: Die Bewertungsfunktion für die Trajektorien aus Abbildung 3.8.

4 Zusammenfassung und Ausblick

Im folgenden wird die Arbeit kurz zusammengefasst und ein Ausblick gegeben.

4.1 Zusammenfassung

In dieser Arbeit wurden aktuelle Pfadplanungsalgorithmen untersucht und simuliert. Dabei wurden die Hindernisse in der Umwelt als Polygone modelliert. Das Fahrzeug wurde mit seinen kinematischen und dynamischen Beschränkungen als Differentialgleichung dargestellt. Somit ist die Pfadplanung für diverse Fahrzeuge ermöglicht worden. Die Ansätze, die sich nicht zufällig verhalten, sind gemäß einer Heuristik und der gewählten Auflösung optimal. Es war zu sehen, dass viele der Verfahren nicht ohne weiteres in jede Umgebung portiert werden können. Die RRT Ansätze kommen in engen Passagen zu keinen Lösungen, da hier die Samplingstrategie nicht passend ist. Der kombinatorische Ansatz findet hier zwar schnell eine Lösung, jedoch können nicht alle kinemaischen und dynamischen Zwangsbedinugen berücksichtigt werden.

Im zweiten Teil der Arbeit ist ein verteilter Koordinationsalgorithmus entwickelt worden. Dieser koordiniert die Fahrzeuge bezüglich einer verteilgen Bewertungsfunktion optimal. Die Fahrzeuge können dabei die Pfadänderungen individuell und abhängig von dem aktuellen Zustand bewerten. Mittels wenig Kommunikation wird zwischen den Fahrzeugen entschieden, welches Fahrzeug seinen Pfad auf welche Weise zu modifizieren hat, sodass es nicht zu Konflikten kommt. Für die verteilte Konflikterkennung ist eine schlanke Trajektorienbeschreibung durch Splines nötig gewesen. Mittels dieser ist es auch einfach und performant möglich Konflikte zwischen Trajektorien festzustellen.

4.2 Ausblick

Die in der Arbeit beschriebenden Ansätze sind bereits gut für viele Umgebungen geeignet. In engen Passagen jedoch liefern RRT Ansätze z.B. keine guten Ergebnisse. Es gibt noch eine Reihe von Verbesserungsideen, die als Ansätze für weiterführende Arbeiten sein können:

- Die Darstellung des gezeiteten Pfades (Trajektorie) durch Splines ist bezüglich der Datenmenge, die für die Übertragung essentiell ist, nicht sehr effektiv. Gerade, wenn ein Pfad viele Kurven enthält, so sind viele Splines zu übertragen. Eine effizientere Darstellung wäre wünschenswert.
- Die Modellierung der Hindernisse als Polygone führt zu einer großen Menge von Kanten, die bei der Kollisionsprüfung geprüft werden müssen. Eine andere Modellierung oder eine hierarische Kollisionprüfung würden hier die Pfadplanung beschleunigen.
- Die optimale Auslastung der Hardware ist derzeit nicht gewähleistet. Die Pfadplanung könnte essentiell beschleunigt werden durch die Ausnutzung paralleler Struckturen vom Rechnern.
- Die Suche nach der nächsten Konfiguration in RRTs, ist derzeit nicht optimal umgesetzt. Die Verwendung eines kd-Baumes, würde hier große Performancesteigerungen mit sich bringen.

Durch die Anwendung der vorgestellten Verbesserungen können die Algorithmen und Ansätze weiter verbessert werden. Auch die Koordination kann bezüglich des Kommunikationsaufwandes und Ausfallsicherzeit verbessert werden.

Literaturverzeichnis

Akella und Hutchinson 2002 AKELLA, S. ; HUTCHINSON, S.: Coordinating the motions of multiple robots with specified trajectories, 2002, S. 624 – 631 vol.1

Azarm und Schmidt 1997 AZARM, K. ; SCHMIDT, G.: Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In: *Robotics and Automation* Bd. 4, Apr 1997, S. 3526–3533 vol.4

Bennewitz 2004 BENNEWITZ, Maren: *Mobile robot navigation in dynamic environments*, Uni Freiburg, Dissertation, 2004. – URL `http://www.freidok.uni-freiburg.de/volltexte/1362/`. – Zugriffsdatum: 26.02.2010

van den Berg und Overmars 2005 BERG, J.P. van den ; OVERMARS, M.H.: Prioritized motion planning for multiple robots, aug. 2005, S. 430 – 435

Berns und Luksch 2008 BERNS, Karsten (Hrsg.) ; LUKSCH, Tobias (Hrsg.): *Autonome Mobile Systeme 2007*. Springer Berlin Heidelberg, 2008. (Informatik aktuell). – URL `http://www.springerlink.com/content/978-3-540-74763-5`. – ISBN 978-3-540-74763-5 (Print) 978-3-540-74764-2 (Online)

Bobyr und Lumelsky 1999 BOBYR, S. ; LUMELSKY, V.: Control of dynamics and sensor based motion planning for a differential drive robot, 1999, S. 157 –162 vol.1

Cormen u. a. 2001 CORMEN, T. H. ; LEISERSON, C. E. ; RIVEST, R. L. ; STEIN, C.: *Introduction to Algorithms*. MIT Press, 2001

Cup 2010 CUP, Carolo: *Carolo Cup*. Website. 2010. – URL `http://carolocup.de/`. – Zugriffsdatum: 26.02.2010

DARPA DARPA: *DARPA Grand Challenge*. Website. – URL `http://www.darpa.mil/grandchallenge/index.asp`. – Zugriffsdatum: 20.10.2010

Deutsch 2004 DEUTSCH, Christian: *Pfadplanung für einen autonomen mobilen Roboter*, TECHNISCHE UNIVERSITAT GRAZ, Diplomarbeit, 2004. – URL `http://www.robocup.tugraz.at/documents/deutsch2004.pdf`

Dijkstra 1959 DIJKSTRA, E. W.: *Numerische Mathematik 1*. Kap. A Note on Two Problems in Connexion with Graphs, S. 269–271, Springer Berlin / Heidelberg, 1959. – URL http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf

Dubins 1957 DUBINS, L. E.: On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. In: *American Journal of Mathematics*, The Johns Hopkins University Press, July 1957, S. 497–516. – URL http://www.jstor.org/pss/2372560. – Zugriffsdatum: 04.08.2010

Erdmann und Lozano-Perez 1986 ERDMANN, M. ; LOZANO-PEREZ, T.: On multiple moving objects. In: *Robotics and Automation. Proceedings. 1986 IEEE International Conference on* Bd. 3, apr 1986, S. 1419 – 1424

Ersson und Hu 2001 ERSSON, T. ; HU, Xiaoming: Path planning and navigation of mobile robots in unknown environments, 2001, S. 858 –864 vol.2

FAUST HAW Hamburg FAUST HAW HAMBURG: *FAUST*. – URL www.informatik.haw-hamburg.de/faust.html. – Zugriffsdatum: 22.10.2010

Ferrari u. a. 1995 FERRARI, C. ; PAGELLO, E. ; OTA, J. ; ARAI, T.: Planning multiple autonomous robots motion in space and time, aug. 1995, S. 253 –259 vol.2

FTS 2009 FTS, Forum: *FTS beim Automobilhersteller VW*. Website. 2009. – URL http://www.forum-fts.de/index.php/fts-anwendungen/fts-beim-vw

Gutmann 2000 GUTMANN, Jens-Steffen: *Robuste Navigation autonomer mobiler Systeme*, Fachhochschule Konstanz, Dissertation, 2000. – URL http://www-home.fh-konstanz.de/~bittel/robo/gutmann2000thesis.ps.gz. – Zugriffsdatum: 26.02.2010

Hamburger Hafen und Logisik AG 2010 HAMBURGER HAFEN UND LOGISIK AG: *Hamburger Hafen und Logisik AG*. 2010. – URL http://www.hhla.de/Umschlag.147.0.html. – Zugriffsdatum: 26.02.2010

Jacob 2004 JACOB, Dr. D.: Roboter in der Automobilindustie / KUKA Roboter GmbH. URL http://www.muenchner-wissenschaftstage.de/mwt2004/content/e160/e707/e728/e751/filetitle/Jacob_ger.pdf. – Zugriffsdatum: 23.08.2010, 2004. – Forschungsbericht

Koditschek 1987 KODITSCHEK, D.: Exact robot navigation by means of potential functions: Some topological considerations, mar. 1987, S. 1 – 6

Koenig und Likhachev 2002 KOENIG, S. ; LIKHACHEV, M.: Improved fast replanning for robot navigation in unknown terrain, 2002, S. 968 – 975 vol.1

Kuffner und LaValle 2000 KUFFNER, Jr. ; LAVALLE, S.M.: RRT-connect: An efficient approach to single-query path planning, 2000, S. 995 –1001 vol.2

Latombe 1991 LATOMBE, Jean-Claude: *Robot Motion Planning*. Norwell, MA, USA : Kluwer Academic Publishers, 1991. – ISBN 079239206X

Latombe 2008 LATOMBE, Jean-Claude: Non-Holonomic Motion Planning / Stanford University. URL `http://robotics.stanford.edu/~latombe/cs326/2009/class12/class12.htm`, 2008. – Class

Laumond 1998 LAUMOND, Jean-Paul P.: *Robot Motion Planning and Control*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1998. – ISBN 3540762191

LaValle 2006 LAVALLE, S. M.: *Planning Algorithms*. Cambridge, U.K. : Cambridge University Press, 2006. – Available at http://planning.cs.uiuc.edu/

Liu und Arimoto 1992 LIU, Yun-Hui ; ARIMOTO, Suguru: Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. In: *Int. J. Rob. Res.* 11 (1992), Nr. 4, S. 376–382. – ISSN 0278-3649

Lozano-Perez 1983 LOZANO-PEREZ, T.: Spatial Planning: A Configuration Space Approach. In: *Computers, IEEE Transactions on* C-32 (1983), feb., Nr. 2, S. 108 –120. – ISSN 0018-9340

Lozano-Perez 1981 LOZANO-PEREZ, Tomas: Automatic Planning of Manipulator Transfer Movements. In: *Systems, Man and Cybernetics, IEEE Transactions on* 11 (1981), oct., Nr. 10, S. 681 –698. – ISSN 0018-9472

Lumelsky und Harinarayan 1997 LUMELSKY, V.J. ; HARINARAYAN, K.R.: Decentralized Motion Planning for Multiple Mobile Robots: The Cocktail Party Model. In: *Autonomous Robots* 4 (1997), March, Nr. 1, S. 121–135. – URL `http://www.springerlink.com/content/l62854230676w201/`. – ISSN 0929-5593 (Print) 1573-7527 (Online)

Melchior und Simmons 2007 MELCHIOR, N.A. ; SIMMONS, R.: Particle RRT for Path Planning with Uncertainty, apr. 2007, S. 1617 –1624. – ISSN 1050-4729

Mojaev 2001 MOJAEV, Alexander: *Umgebungswahrnehmung, Selbstlokalisation und Navigation mit einem mobilen Roboter*, Uni Thübingen, Dissertation, 2001. – URL `http://www.ra.cs.uni-tuebingen.de/mitarb/mojaev/documents/thesis.ps.gz`. – Zugriffsdatum: 26.02.2010

Mutambara und Durrant-Whyte 1993 MUTAMBARA, A.G.O. ; DURRANT-WHYTE, H.F.: Modular Decentralized Robot Control, jul. 1993, S. 101 –106

Naumann u. a. 1998 NAUMANN, R. ; RASCHE, R. ; TACKEN, J.: *Managing autonomous vehicles at intersections*. May/Jun 1998

Naumann u. a. 1997 NAUMANN, R. ; RASCHE, R. ; TACKEN, J. ; TAHEDI, C.: *Validation and simulation of a decentralized intersection collision avoidance algorithm*. Nov 1997. – 818–823 S

Pallottino u. a. 2007 PALLOTTINO, L. ; SCORDIO, V.G. ; BICCHI, A. ; FRAZZOLI, E.: Decentralized Cooperative Policy for Conflict Resolution in Multivehicle Systems. In: *Robotics, IEEE Transactions on* 23 (2007), Dec., Nr. 6, S. 1170–1183. – ISSN 1552-3098

Peasgood u. a. 2008 PEASGOOD, M. ; CLARK, C.M. ; MCPHEE, J.: A Complete and Scalable Strategy for Coordinating Multiple Robots Within Roadmaps. In: *Robotics, IEEE Transactions on* 24 (2008), April, Nr. 2, S. 283–292. – ISSN 1552-3098

Rebai u. a. 2007 REBAI, K. ; AZOUAOUI, O. ; BENMAMI, M. ; LARABI, A.: Car-like robot navigation at high speed. (2007), dec., S. 2053 –2057

Rimon und Koditschek 1992 RIMON, E. ; KODITSCHEK, D.E.: Exact robot navigation using artificial potential functions. In: *Robotics and Automation, IEEE Transactions on* 8 (1992), oct., Nr. 5, S. 501 –518. – ISSN 1042-296X

RoboCup ROBOCUP: *RoboCup*. – URL `www.robocup.org`. – Zugriffsdatum: 22.10.2010

Roszkowska 2008 ROSZKOWSKA, E.: Decentralized motion-coordination policy for cooperative mobile robots, may. 2008, S. 364 –369

Rull 2010 RULL, Andrej: *Fahrspur- und odometriebasierte Selbstlokalisierung und Kartierung*, HAW Hamburg, Masterarbeit, 2010. – forthcoming

Samuel und Keerthi 1993 SAMUEL, S. ; KEERTHI, S.S.: Numerical determination of optimal non-holonomic paths in the presence of obstacles, may. 1993, S. 826 –831 vol.1

Schubert 2006 SCHUBERT, Dipl.-Ing. R.: *Automatische Bahnplanung und Hindernisumfahrung für ein autonom navigierendes Fahrzeug*, TU Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Diplomarbeit, 2006. – URL `http://archiv.tu-chemnitz.de/pub/2006/0152`

Shan u. a. 2009 SHAN, Enzhong ; DAI, Bin ; SONG, Jinze ; SUN, Zhenping: A Dynamic RRT Path Planning Algorithm Based on B-Spline, dec. 2009, S. 25 –29

STILL GmbH 2009 STILL GMBH: *STILL GmbH*. Website. 2009. – URL `http://www.still.de`. – Zugriffsdatum: 22.10.2010

Svestka und Overmars 1995 SVESTKA, P. ; OVERMARS, M.H.: Coordinated motion planning for multiple car-like robots using probabilistic roadmaps, may. 1995, S. 1631 –1636 vol.2. – ISSN 1050-4729

Takahashi und Ohnishi 2003 TAKAHASHI, H. ; OHNISHI, K.: Autonomous decentralized control for formation of multiple mobile-robots considering ability of robot, nov. 2003, S. 2041 – 2046 Vol.3

Volkswagen AG VOLKSWAGEN AG: *Gläserne Manufaktur VW AG*. – URL `http://www.glaesernemanufaktur.de/`. – Zugriffsdatum: 26.10.2010

Weisser u. a. 1999 WEISSER, H. ; SCHULENBERG, P.J. ; GOLLINGER, H. ; MICHLER, T.: Autonomous driving on vehicle test tracks: overview, implementation and vehicle diagnosis, 1999, S. 62 –67

Zulli u. a. 1995 ZULLI, R. ; FIERRO, R. ; CONTE, G. ; LEWIS, F.L.: Motion planning and control for non-holonomic mobile robots, aug. 1995, S. 551 –557

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach APSO-TI-BM §16(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 30. August 2010	
Ort, Datum	Unterschrift