Universität Heidelberg
Institut für Technische Informatik
Lehrstuhl für Automation

Bachelor-Arbeit

Sampling-based Path Planning for Human Steering Evaluation

Name:	Arne Hegel
Matrikelnummer:	4016777
Betreuer:	Prof. em. Dr. sc. techn. E. Badreddin
Mitbetreuer:	MSc. H. Dieterich
Datum der Abgabe:	30. März 2021

Ich versichere, dass ich diese Bachelor-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe und die Grundsätze und Empfehlungen "Verantwortung in der Wissenschaft" der Universität Heidelberg beachtet wurden.

Abgabedatum: 30. März 2021

Zusammenfassung

Die Aufrechterhaltung der Selbstständigkeit einer immer älter werdenden Gesellschaft gewinnt zunehmend an Relevanz. Ein elektrischer Rollstuhl mit Assistenzfunktionen kann dies im Bereich der Mobilität ermöglichen. So können besonders motorisch eingeschränkte Menschen durch solch einen Rollstuhl unterstützt werden. Um den Rollstuhlfahrer in Situationen zu unterstützen, in denen er diese benötigt, muss diese Situation zunächst erkannt werden. Eine Möglichkeit, um dies zu bewerkstelligen, ist es, das Steuerverhalten des Rollstuhlfahrers fortlaufend zu bewerten. Diese Bewertung muss jedoch die nicht-holonomen Zwangsbeschränkungen des Rollstuhls beachten.

Das Problem ist somit zweiteilig. Das erste Teilproblem besteht darin, auf einer gegebenen Karte möglichst schnell einen Weg aus Zwischenzielen zu einem vorher bestimmten Ziel zu berechnen. Dieser Weg muss dabei schnell, intuitiv und navigierbar sein. Das zweite Teilproblem besteht darin, die Distanz entlang des zuvor berechneten Weges fortlaufen zu berechnen, um die Entwicklung der Distanz zu erfassen. Die Distanzfunktion muss dabei die nicht-holonomen Zwangsbeschränkungen des Rollstuhls widerspiegeln. Mit Hilfe der Distanzen soll daraufhin eine Bewertung des Steuerverhaltens vorgenommen werden können.

Um dies zu realisieren, wird zunächst mit Hilfe eines stichprobenbasierten Algorithmus ein Graph auf dem navigierbaren Bereich generiert, dessen Kanten mit einem Rollstuhl abgefahren werden können. Auf diesem Graphen wird daraufhin unter Verwendung des A* Algorithmus ein optimaler Weg zu einem Ziel berechnet, welches der Fahrer zuvor ausgewählt hat. Während der Fahrt wird entlang des Weges fortlaufend die Distanz zum Ziel berechnet. Die berechneten Distanzen können mit den vorherigen Berechnungen verglichen werden und bieten somit eine Möglichkeit, das Steuerverhalten des Fahrers zu bewerten.

In einer experimentellen Evaluation wurde gezeigt, dass mit Hilfe dieser Bewertungsmethode das Steuerverhalten in den meisten Fällen sehr gut bewertet werden kann. Da der Graph jedoch aus zufällig ausgewählten Knoten besteht, kann es unter Umständen zu Fehlinterpretationen des Steuerverhaltens kommen.

Abstract

Supporting the independence of an increasingly aging society is becoming more and more relevant. An electric wheelchair with assistance functions can make this possible in the area of mobility. Especially motorically impaired people can be supported by such a wheelchair. In order to support the wheelchair user in situations where he needs it, this situation must first be identified. One way to accomplish this is to continuously evaluating the wheelchair user's steering behavior. However, this evaluation must take into account the non-holonomic constraints of the wheelchair.

In this way, the problem is two-fold. The first subproblem is to compute a path from intermediate goals to a previously determined goal as short as possible on a given map. This path must be fast, intuitive and navigable. The second subproblem is to compute the distance along the previously computed path in order to measure the evolution of the distance. The distance function must reflect the non-holonomic constraints of the wheelchair. It should then be possible to evaluate the steering behavior with the help of the distances.

In order to realize this, first a graph is generated on the navigable space using a sampling-based algorithm, whose edges can be traversed with a wheelchair. On this graph, an optimal path to a goal previously selected by the driver is then calculated using the A* algorithm. During the drive, the distance to the goal is continuously calculated along the path. The calculated distances can be compared with the previous calculations and thus provide a way to evaluate the driver's steering behavior.

In an experimental evaluation it was shown that with the help of this evaluation method of the steering behavior can be evaluated very well in most cases. However, since the graph consists of randomly selected nodes, misinterpretation of the steering behavior may occur.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Aufgrund von verbessertem sozialen Umfeld und voranschreitender Medizin wird unsere Gesellschaft immer älter. Daher ist es notwendig die Selbstständigkeit dieser Gruppe, welche häufig unter eingeschränkter Mobilität leidet, zu erhalten. Elektrische Rollstühle werden in vielen Bereichen von dieser Gruppe genutzt. Da diese jedoch eine präzise Kontrolle des Fahrers benötigen, welche der Fahrer unter Umständen nicht leisten kann, sollen ihn Assistenzfunktionen unterstützen. [Choi and Choi, 2017]

Üblicherweise werden Steuereingaben in Form von Linear- und Rotationsgeschwindigkeit übermittelt, wie beispielsweise bei elektrischen Rollstühlen über einen Joystick. Um die automatisierten Hilfsfunktionen an den Fahrer anzupassen, muss sein Fahrverhalten während der Fahrt analysiert und ausgewertet werden. Zur Auswertung werden bei existierenden Ansätzen neben den vom Fahrer initiierten Geschwindigkeiten auch lokale Maße wie die Position des Fahrzeugs relativ zu einem Ziel oder zu Hindernissen verwendet [Poncela et al., 2009]. Neben den genannten lokalen Maßen sind auch globale Maße (wie beispielsweise der Fortschritt zum Ziel) geeignet, um die Eingangssignale des Fahrers zu bewerten. Soll auch die Distanz zum Ziel evaluiert werden, muss diese durch eine Wegplanung fortlaufend ermittelt werden. Bei der Berechnung der Distanz zum Ziel muss jedoch berücksichtigt werden, dass herkömmliche Rollstühle sich nicht entlang der Querachse der Antriebsräder bewegen können.

1.2 Ziele der Arbeit

Im Labor des Instituts für Technische Informatik der Universität Heidelberg wurde ein elektronischer Rollstuhl mit Sensoren ausgestattet, sodass dessen 3D-Position (x, y, φ) beim Fahren ermittelt werden kann. Zusätzlich können durch Laserscanner und Ultraschall - Sensoren die Abstände zu Objekten in der Umgebung gemessen werden. Die Variablen (x, y) beschreiben hierbei den Schnittpunkt der Koordinaten, an dem sich der Rollstuhl befindet und φ dessen Orientierung. Linear- und Rotationsgeschwindigkeit können in Form von (v, ω) über einen Joystick vom Nutzer eingegeben werden. Die

Sensor- und Eingabedaten können während des Fahrens sowohl online als auch offline verarbeitet und gespeichert werden. Mit Hilfe der Sensordaten kann so eine Umgebung abgefahren, erfasst und gespeichert werden, wobei hierbei nur statische Objekte erfasst werden sollten, um später mit der Karte in dieser statischen Umgebung navigieren zu können.

Während der Rollstuhl zu einer Zielposition gesteuert wird, kann also seine 3D-Position ermittelt werden. Ist das Ziel bekannt, können die Fortschritte des Fahrers fortlaufend evaluiert werden.

Ziel der Arbeit ist es, unter Verwendung einer Start- und einer Zielposition automatisch einen Weg mit Zwischenzielen zu berechnen, welche dann alle nacheinander auf einem kollisionsfreien Pfad abgefahren werden. Die Pfade zwischen den Zwischenzielen sollen zudem unter der Annahme berechnet werden, dass der Rollstuhl, welcher den Pfad abfahren soll, ein nicht-holonomes System ist. Es ist dem Rollstuhl also beispielsweise nicht möglich seitlich zu fahren.

Aufgrund der Eigenschaften eines nicht-holonomen Systems (z.B. Unfähigkeit von Rollstühlen, seitwärts zu fahren) muss ein Distanzmaß verwendet werden, welches diese Eigenschaften berücksichtigt und die Distanz des zuvor erwähnten berechneten Pfades widerspiegelt. Da sich der Rollstuhl kontinuierlich fortbewegt, und sich somit die Navigationsaufgabe mit jeder neuen Position fortlaufend ändert, muss die verbleibende Distanz zur Zielposition kontinuierlich aktualisiert werden. Die Distanz von der aktuellen Position des Rollstuhls bis zur Zielposition berechnet sich hierbei so, dass die Distanz von der Rollstuhlposition zum nächsten Zwischenziel berechnet wird und auf diese Distanz die Distanzen zwischen den folgenden Zwischenzielen bis zum endgültigen Ziel aufaddiert werden.

Zur Bewertung des Fahrverhaltens wird das nicht-holonome Distanzmaß verwendet, welches sich bei korrektem Fahrverhalten mit konstanter Geschwindigkeit linear bis zu einer Distanz von null verringert. Bei unpassenden Eingaben bezüglich des Navigationsfortschritts soll das System eine verringerte Reduzierung bis zu einem Anstieg der Restdistanz erkennen.

1.3 Problemstellung

Das Problem kann in zwei Teile unterteilt werden.

1. Zunächst muss, unter Zuhilfenahme einer *occupancy grid map*[1] innerhalb einer statischen Umgebung, ein Weg von einer gegebenen Startposition zu einer gegebe-

[1] Dies stellt eine 2-D-Rasterkarte dar, in der jede Zelle die Belegungswahrscheinlichkeit darstellt.

nen Zielposition berechnet werden. Dieser Weg soll aus Zwischenzielen bestehen, welche nacheinander auf kurzen und intuitiven Pfaden erreicht werden können. Die Bezeichnung "kurz"bezieht sich hier auf ein nicht-holonomes Distanzmaß zwischen zwei 3D-Positionen (x, y, φ), welches in Gleichung (2.19) beschrieben ist [Park and Kuipers, 2015].

2. Der zweite Teil des Problems besteht darin, unter Verwendung des zuvor berechneten Weges mit Zwischenzielen, eine Methode zu erarbeiten, die es ermöglicht, das Fahrverhalten eines Rollstuhlfahrers auf diesem Weg zu evaluieren. Die Bewertung kann mit Hilfe einer nicht-holonomen Distanzfunktion geschehen, welche die Distanz der aktuellen Position zum nächsten Zwischenziel und zwischen allen folgenden Zwischenzielen berechnen kann. Die Summe dieser Teildistanzen ergibt eine Gesamtdistanz, die zur Bewertung verwendet werden kann.

1.4 Lösungsansatz

Um einen Weg zwischen einer Start- und einer Zielposition zu berechnen, wird zunächst ein Wegfindungsalgorithmus benötigt. Diverse Wegfindungsalgorithmen, welche eine Distanz über einen navigierbaren Bereich mit nicht-holonomen Navigationsbeschränkungen berechnen, wurden schon früher untersucht [Konolige, 2000], [Park and Kuipers, 2015]. Da für unser Distanzmaß Zwischenziele benötigt werden, bietet sich insbesondere hier ein stichprobenbasierter Wegfindungsalgorithmus an. Der optimierte stichprobenbasierte Wegfindungsalgorithmus des *Rapidly Exploring Random Tree RRT** ist hierfür besonders gut geeignet, da er sich mit steigender Anzahl der Stichproben asymptotisch dem optimalen Pfad annähert [Park and Kuipers, 2015]. Zudem haben [Park and Kuipers, 2015] eine Möglichkeit gefunden, Zwischenziele mit nicht-holonomen Pfaden in einen RRT* einzufügen, indem sie eine Distanzfunktion für zwei 3D-Positionen (x, y, φ) definierten, welche die Distanz unter nicht-holonomen Zwangsbedingungen korrekt wiedergibt. Ein Beispielergebnis des RRT* von [Park and Kuipers, 2015] ist in Abbildung 1.1 dargestellt. Auf Grundlage dieses Graphens ist eine wiederholte Neuplanung für alle Positionen möglich, um die Gesamtdistanz zu aktualisieren. Indem die berechneten Distanzwerte nach jeder Berechnung gespeichert und verglichen werden, ist eine Bewertung des Fahrtfortschritts möglich.

Der Pfad von der Startposition zur Zielposition wird als eine Liste von Zwischenzielen ausgegeben. Die Distanzfunktion von [Park and Kuipers, 2015] stellt einen Übergang zwischen zwei Zwischenzielen dar, welcher von einem nicht-holonomen System, wie dem Rollstuhl, in glatten Bewegungen abgefahren werden kann. Zusätzlich können die Kosten

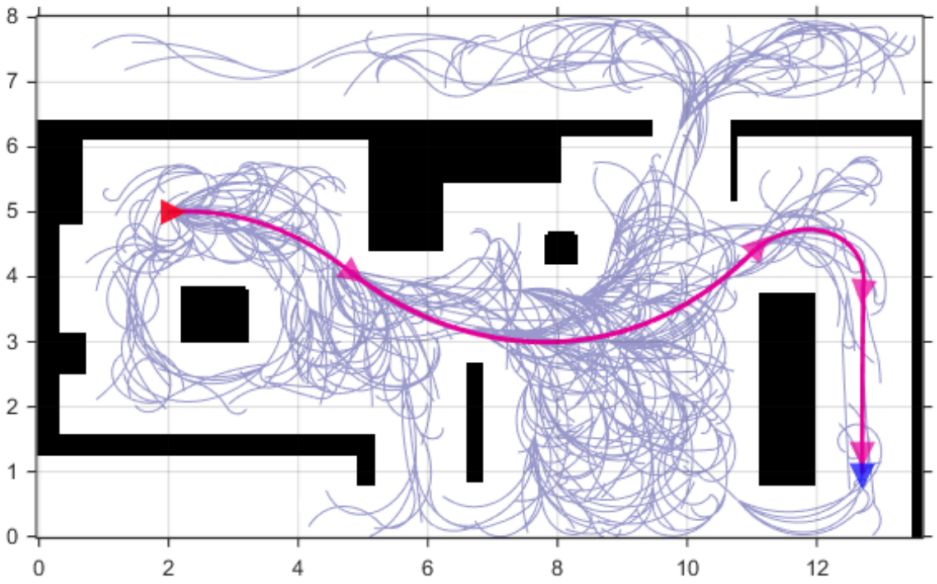

Abbildung 1.1: Nicht-holonomer Pfad mit minimaler Distanz durch RRT* berechnet. [Park and Kuipers, 2015]
(Start: rotes Dreieck, Ziel: blaues Dreieck, Berechneter Pfad mit Zwischenzielen: magenta, Weitere berechnete Pfade: grau)

für lineare und Winkelbeschleunigung berücksichtigt werden, um die Bewegungsqualität zu sichern.

1.5 Aufbau der Arbeit

Für Programme mit der Aufgabe, das Fahrverhalten von Rollstuhlfahrern mit einem Distanzmaß zu bewerten, ist es notwendig, einen navigierbaren Weg zu berechnen, welcher einem sicheren, schnellen und intuitiven Fahrverhalten entspricht.

In dieser Arbeit wird gezeigt, dass ein Weg mit entsprechenden Eigenschaften berechnet werden kann und auf dessen Grundlage eine Bewertung des Fahrverhaltens stattfinden kann. Zudem wird gezeigt, dass diese Bewertung während der Fahrt stattfinden kann.

Aufbau der Arbeit:

- Grundlagen und verwandte Arbeiten (Kapitel 2)
  Die Eigenschaften und der Aufbau der Wegfindungsalgorithmen RRT* und A* werden vorgestellt. Des weiteren werden eine Positionsregelung und eine Distanzfunktion beschrieben, welche nicht-holonomen Zwangsbeschränkungen unterliegen.

- Implementierung (Kapitel 3)
  In diesem Kapitel werden Implementierungen und Designentscheidungen beschrieben und erläutert, welche zur Lösung der Problemstellung verwendet wurden. Die beiden Teilprobleme der Problemstellung werden in zwei getrennten Kapiteln behandelt.
  - Nicht-holonomen RRT* (Kapitel 3.1)
    Es wird die Implementierung eines RRT* Algorithmus mit nicht-holonomer Distanzfunktion beschrieben, welcher einen Pfad für nicht-holonome Systeme berechnet. Dabei wird der Pfad kollisionsfrei, schnell, intuitiv und navigierbar sein [Park, 2016]. Dass der Pfad kollisionsfrei ist, wird hierbei sichergestellt, indem der berechnete Pfad an bestimmten Punkten in regelmäßigen Abständen auf eine Mindestentfernung zu allen Objekten überprüft wird.
  - Bewertung menschlichen Steuerverhaltens (Kapitel 3.2)
    Das menschliche Steuerverhalten wird mithilfe der Distanz zu einem vorher definierten Ziel bewertet. Die Distanz berechnet man hier mit dem in Kapitel 3.1 entworfenen Algorithmus und dessen ausgegebenen Pfad berechnet. Nach Aktualisierungen der Position des Rollstuhls wird die Distanz auf Grundlage des zuvor schon berechneten Pfades aktualisiert und gespeichert. Daraufhin ist ein Vergleich mit zuvor gemessenen Distanzen möglich, der eine Auskunft über das Steuerverhalten des Fahrers gibt.

- Experimentelle Evaluation (Kapitel 4)
  Es wird sowohl das menschliche Steuerverhalten an einem ferngesteuerten Roboter als auch am zuvor erwähnten elektrischen Rollstuhl bewertet. Bei beiden Experimenten werden sowohl die Distanz zum Ziel als auch die Position des Gefährts kontinuierlich aufgezeichnet und anschließend analysiert. Dies soll eine Auskunft darüber geben, wie gut eine Bewertung des Steuerverhaltens auf Grundlage der Distanz zum Ziel funktioniert.

- Ausblick (Kapitel 5)
  In diesem Kapitel werden die wichtigsten Ergebnisse und Erkenntnisse der Arbeit kurz zusammengefasst und mögliche Verbesserungsvorschläge für weiterführende Arbeiten vorgestellt.

2 Grundlagen und verwandte Arbeiten

Es existieren schon zahlreiche Arbeiten zur Bewegungsplanung. In diesem Kapitel wird ein grober Überblick über einige dieser Arbeiten gegeben und auch spezifischer auf Arbeiten eingegangen, welche für den weiteren Verlauf dieser Arbeit besonders wichtig sind.

Als *Agent* wird im Folgenden eine Person oder ein Roboter bezeichnet, welcher sich mit oder ohne Zuhilfenahme von Hilfsmitteln fortbewegt.

Graphen, auf denen Suchoperationen durchgeführt werden, bestehen aus *Knoten* und *Kanten*. *Knoten* besitzen eine *Position* im Raum, sowie *Nachbarknoten*. *Kanten* sind Verbindungen zwischen *Knoten*, welche durch zwei *Knoten* und *Kosten* zwischen den beiden *Knoten* repräsentiert werden.

2.1 Wegfindungsalgorithmen

Die Wegfindung ist definiert als das Problem, einen kollisionsfreien Weg mit Start- und Endpunkt zu finden, den ein Agent mit möglichst geringen Kosten abfahren kann. Zur Berechnung des Weges ist im Regelfall die komplette geometrische Struktur der Umgebung in Form einer Karte gegeben. Die Kosten berechnen sich je nachdem, welchen Beschränkungen und Bedingungen, wie beispielsweise der minimalen Distanz oder der kürzesten Zeit, der Weg unterliegen soll. Es können jedoch auch Anforderungen an die Berechnungszeit oder Komplexität des Algorithmus gesetzt werden. So können beispielsweise zu lange Berechnungszeiten ein wiederholtes Anwenden des Algorithmus erschweren. In klassischen Anwendungen werden die kinematischen Zwangsbeschränkungen des Agenten meist vernachlässigt, weshalb zum Abfahren des Weges eine separate Steuerung benötigt wird. Dieses Problem wird in Kapitel 2.2 weiter behandelt. Im Folgenden werden zwei Arten von Wegfindungsalgorithmen beschrieben. [Park, 2016] [Korkmaz and Durdu, 2018]

2.1.1 Wegfindung auf einem gegebenen Graphen

Die Wegfindungsalgorithmen von Dijkstra [Dijkstra, 1959] und der A* Algorithmus [Hart et al., 1968] sind gut etabliert und finden auf einem gegebenen Graphen immer den optimalen Weg. Dabei gehören sie zu der Gruppe der Suchalgorithmen. Der A* Algorithmus geht nach dem Prinzip vor, immer das vielversprechendste Teilziel weiter zu erkunden. Dadurch ist er beim Durchsuchen von großen Graphen besonders schnell. Anzumerken ist hierbei, dass der A* Algorithmus eine Erweiterung des Algorithmus von Dijkstra ist, welcher versucht, die Anzahl der Knoten, die auf dem Graph untersucht werden müssen, zu reduzieren. Dies geschieht, indem der Algorithmus eine heuristische Schätzung der Kosten einbezieht, um von einem gegebenen Knoten zum Ziel zu gelangen. [LaValle, 2006] [Park, 2016]

Der A* Algorithmus, welcher in Algorithmus 1 zu sehen ist, arbeitet auf einem Graphen G mit Knoten $v \in V$ und Kanten $e \in E$. Dabei sind die Kanten mit nicht negativen Kosten $cost(e)$ verbunden. Die Kosten einer Kante können auch durch $cost(v_1, v_2)$ beschrieben werden, wobei $v_1 \in V$ den Startknoten und $v_2 \in V$ den Zielknoten einer direkten Verbindung beschreibt. Die *Gesamtkosten* des Weges von einem Startknoten zu einem Zielknoten sind die aufsummierten Kosten über alle abgegangenen Knoten. [LaValle, 2006]

Die folgende Beschreibung des A* Algorithmus orientiert sich hauptsächlich an den Arbeiten von [LaValle, 2006] und [Kevin M. Lynch and Frank C. Park, 2018].

Algorithm 1 $foundTrace \leftarrow$ A*$(start, goal)$

```
openlist ← []
closedlist ← []
start.g ← 0
start.f ← cost(start, goal)
openlist.add(start)
for all Nodes do
  node.h ← cost(node, goal)
end for
while not openlist.isEmpty() do
  currentNode ← openlist.removeMin() {Get the current node with lowest f-value}
  if currentNode == goal then
    return TRUE
  end if
  closedlist.add(currentNode)
  openlist ← expandNode(openlist, closedlist, currentNode)
end while
return FALSE
```

Zu Beginn werden die geschätzten Kosten h aller Knoten zum Ziel berechnet, wobei von einer direkten Verbindung zwischen diesen ausgegangen wird.

$$\forall v \in V : v.h = cost(v, goal) \tag{2.1}$$

Zudem wird der Startknoten, welcher mit dem Zielknoten an den Algorithmus übergeben wurde, initialisiert und in die *openlist* eingefügt, welche die als nächstes zu untersuchenden Knoten beinhaltet. Vollständig untersuchte Knoten werden hingegen in die *closedlist* eingefügt. Bei der Initialisierung des Startknotens werden dessen Kosten g vom Startknoten zum aktuell betrachteten Knoten über alle abgegangenen Knoten berechnet. Da sich der A* Algorithmus vom Startknoten aus kontinuierlich ausbreitet, sind die Kosten über alle abgegangenen Knoten bis zum Vorgänger *parent* gespeichert und erleichtern somit die Berechnung von g.

$$g : V \times V \to \mathbb{R}, g(v, parent) = v.parent.g + cost(v.parent, v) \tag{2.2}$$

Bei der Initialisierung des Startknotens sind Startknoten und betrachteter Knoten derselbe, weswegen die Kosten $start.g = 0$ sind. Die Kosten f setzen sich aus den berechneten Kosten g vom Startknoten zum aktuell betrachteten Knoten und den geschätzten Kosten h bis zum Ziel zusammen.

$$f : V \to \mathbb{R}, f(v) = v.g + v.h \tag{2.3}$$

Bei der Initialisierung des Startknotens wird somit $start.f = start.h$ gesetzt. Dieser Wert gibt einen Anhaltspunkt, wie empfehlenswert es ist, ihn als nächstes zu untersuchen. Diese Komponente des A* Algorithmus ist auch die entscheidende Ergänzung zum Dijkstra Algorithmus. Nach der Initialisierung wird nun so lange der Graph durchsucht, bis entweder das Ziel erreicht wurde oder alle Knoten untersucht wurden. Dabei wird in jedem Schritt zunächst der vielversprechendste Knoten $v_{current} \in openlist$ genommen.

$$removeMin(openlist) \equiv \{v_1 \in openlist : \forall v_2 \in openlist : v_1.f \leq v_2.f\} \tag{2.4}$$

Falls $v_{current}$ nicht der Zielknoten ist, wird er in die *closedlist* eingefügt und die *openlist* wird um seine Nachbarn $v_{current}.neighbors$ erweitert. In Algorithmus 2 ist be-

schrieben, wie die *openlist* durch die Nachbarn von $v_{current}$ erweitert wird.

Algorithm 2 $openlist \leftarrow$ expandNode($openlist, closedlist, currentNode$)

```
for all neighbors of currentNode do
  if closedlist.contains(neighbor) then
    continue
  end if
  tmpG = currentNode.g+ cost(currentNode, neighbor)
  if openlist.contains(neighbor) and tmpG ≥ neighbor.g then
    continue
  end if
  neighbor.parent ← currentNode
  neighbor.g ← tmpG
  neighbor.f ← tmpG + neighbor.h
  if not openlist.contains(neighbor) then
    openlist.add(neighbor)
  end if
end for
return openlist
```

Hierbei werden alle Nachbarn von $v_{current}$, welche noch nicht in der *closedlist* sind, nacheinander betrachtet. Zunächst berechnet man vorläufig die Kosten *tmpG* für den betrachteten Nachbarn *neighbor* über den Vorgänger $v_{current}$ mit der Gleichung (2.2). Falls *neighbor* entweder noch nicht in *openlist* vorhanden ist, oder falls doch und $tmpG < neighbor.g$, dann werden die Werte von *neighbor* aktualisiert. Ist dieser nicht in *openlist* vorhanden, so wird er diese eingefügt. Nach vollständigem Durchlaufen der Nachbarn wird die aktualisierte *openlist* Liste zurückgegeben.

Der Weg vom Startknoten zum Zielknoten kann nun beim Rückgabewert $TRUE$ des Algorithmus 1 ausgelesen werden. Dies geschieht, indem vom Zielknoten aus immer $v.parent$ ausgelesen wird, bis man den Startknoten erreicht hat.

2.1.2 Nicht-holonome stichprobenbasierte Wegfindung

Eine weitere Art der Wegfindung basiert auf Stichproben, wobei stichprobenartig Koordinaten auf einer vorhandenen Karte ausgewählt werden können, welche als Knoten im Graphen dienen sollen. So wird zu Beginn kein Graph benötigt, jedoch genaue Kenntnisse über die Karte, auf welcher der Algorithmus angewandt werden soll.

Die Algorithmen *Rapidly-Exploring Random Tree* (RRT) [LaValle, Steven M, 1998] und *probabilistic roadmap* (PRM) [Kavraki et al., 1996] gehören der Klasse der stichprobenbasierten Wegfindungsalgorithmen an und sind dort sehr gut etabliert. Sie können in hochdimensionalen zusammenhängenden Räumen arbeiten und können dabei unter-

schiedlichste Bedingungen, wie auch die Abmessungen des Agenten, in die Problemformulierung mit einbeziehen. Dabei berechnen sie einen Weg, welcher vom Agenten unter allen gegebenen Bedingungen begangen werden kann. Das gelingt ihnen, indem sie einen topologischen Graphen generieren, welcher einen Startknoten und einen Zielknoten mit beliebig vielen Zwischenzielen verbindet. Jeder Knoten ist auf dem Weg zwischen Start und Ziel mit mindestens einem Nachbarknoten verbunden, wobei die Kanten jeweils ein navigierbares Wegsegment im freien Raum beschreiben. Aufgrund der stichprobenbasierten Generierung von Knoten ist der gefundenen Wege jedoch oft nicht optimal. Beim optimierten RRT Algorithmus RRT* [Karaman and Frazzoli, 2011] wurde dieses Problem behoben, wodurch er sich asymptotisch mit steigender Stichprobenanzahl dem optimalen Weg annähert. Dies wird durch die Neuverknüpfung von Knoten, nach dem Einfügen eines neuen Knoten, ermöglicht [Park, 2016].

Die folgende Beschreibung des RRT* Algorithmus orientiert sich hauptsächlich an [Karaman et al., 2011] und [Park, 2016].

Algorithm 3 $parents, cost \leftarrow \text{RRT*}(start, lim)$

```
parents ← dict()
cost ← dict()
count ← 0
parents[start] ←′ start′
cost[start] ← 0
while count < lim do
  new ← sample()
  nearest ← getNearest(new, parents)
  new ← extend(nearest, new)
  if new == NULL then
    continue
  end if
  neighbors ← nearTo(new, parents)
  min ← chooseParent(neighbors, new)
  parents[new] ← min
  cost[new] ← cost[min]+dist(min, new)
  neighbors ← nearFrom(new, parents)
  parents, cost ← reWire(neighbors, parents, cost, min, new)
  counter ← counter + 1
end while
return parents, cost
```

Der RRT* Algorithmus, welcher in Algorithmus 3 zu sehen ist, beginnt mit der Initialisierung des Verzeichnisses *parents*, welches den Graphen repräsentiert und den Vorgänger eines jeden Knoten speichert, und des Verzeichnisses *cost*, welches die Gesamtkosten vom Startknoten bis zum aktuell betrachteten Knoten speichert. Der an den

Algorithmus übergebene Startknoten wird daraufhin in die Verzeichnisse eingetragen. Im Folgenden wird der Graph so lange erweitert, bis er die gewünschte Anzahl an Knoten besitzt. Die Erweiterung des Graphen dabei erfolgt, indem ein zufälliger valider Knoten v_{new} auf der Karte generiert wird. Dieser bekommt, falls möglich, den nächstgelegenen Knoten $v_{nearest}$ als Vorgänger,

$$\begin{aligned} getNearest(v_{new}, parents) \equiv \{v_1 \in parents : \forall v_2 \in parents : \\ dist(v_1, v_{new}) \leq dist(v_2, v_{new})\} \end{aligned} \tag{2.5}$$

wobei v_{new} bei zu großer Entfernung zu $v_{nearest}$ noch in dessen Richtung verschoben werden kann. Falls diese Verbindung aufgrund zuvor definierter Beschränkungen nicht möglich ist, wird dieser Knoten verworfen und ein Neuer wird generiert. Bei einer validen Verbindung der beiden Knoten werden alle weiteren Nachbarn *neighbors* von v_{new}, welche zu v_{new} hinführen, herausgesucht.

$$nearTo(v_{new}, parents) \equiv \{v \in parents : dist(v, v_{new}) \leq radius\} \tag{2.6}$$

Der Vorgänger aus *neighbors* mit den geringsten Gesamtkosten von v_{min} zu v_{new} wird daraufhin bestimmt.

$$\begin{aligned} chooseParent(neighbors, v_{new}) \equiv \{v_1 \in neighbors : \forall v_2 \in neighbors : \\ cost[v_1] + dist(v_1, v_{new}) \leq cost[v_2] + dist(v_2, v_{new})\} \end{aligned} \tag{2.7}$$

Hierbei muss jedoch jeder Knoten in *neighbor* eine valide Verbindung von v_{new} aus besitzen. Der Knoten v_{new} wird mit v_{min} als Vorgänger in den Graphen eingefügt und aktuelle Verbindungen werden, falls nötig, neu über v_{new} verbunden. Dazu werden zuvor alle Nachbarn *neighbors*, welche vom v_{new} aus erreicht werden können,

$$nearFrom(v_{new}, parents) \equiv \{v \in parents : dist(v_{new}, v) \leq radius\} \tag{2.8}$$

mit $reWire$ aufgerufen. Die Neuverbindungen werde mit $reWire$ durchgeführt, indem die Validität der Verbindung von v_{new} zu *neighbor* überprüft wird, und die aktuellen Gesamtkosten von *neighbor* mit denen über v_{new} zu *neighbor* verglichen werden.

$$getToReWire(neighbors, costv_{min}, v_{new}) \equiv \{v \in neighbors \setminus \{v_{min}\} : validLink(v_{new}, v) \wedge cost[v_{new}] + dist(v_{new}, v) < cost[v]\} \quad (2.9)$$

Die Nachbarn, welche eine valide Verbindung und geringere Gesamtkosten über v_{new} haben, werden neu verbunden.

Nachdem der Graph vollständig aufgebaut wurde, kann die Zielposition (falls möglich) als Knoten eingefügt werden. Von ihm aus ist es möglich, den Weg über *parents* bis zu Startknoten zurückzuverfolgen und so den Weg auszulesen.

2.2 Positionsregelung

Das Ansteuern einer Position auf einer ebenen Fläche ist ein fundamentales Problem bei autonom fahrenden Agenten [Park, 2016]. Die Qualität des Weges, welcher zum Erreichen der Position gewählt wird, ist hierbei von großer Bedeutung, insbesondere beim Transport von Objekten oder Lebewesen. Bei einer Fortbewegung unter nichtholonomen Zwangsbeschränkungen ist die Berechnung eines Weges von hoher Qualität, aufgrund der zusätzlichen Beschränkungen, besonders kompliziert.

[Park, 2016] hat ein Regelgesetz entworfen, welches einem einrädrigen Fahrzeug ermöglicht, von seiner aktuellen Position eine gewünschte Zielposition anzufahren. Dies geschieht dabei so, dass die Bewegung sicher, komfortabel, schnell und intuitiv ist. Das Regelgesetz wurde als Einrad modelliert, da normale Autos oder auch Rollstühle als Solche vereinfacht werden können [LaValle, 2006]. Das Fahren in eine vollkommen beliebige Richtung ist hierbei nicht möglich. [Park, 2016]

Um die Kinematik des Fahrzeugs besser beschreiben zu können, wird von [Park, 2016] ein Polarkoordinatensystem verwendet, welches in Abbildung 2.1 zu sehen ist. Dadurch ist es möglich, die Sichtweise aus der Fahrerposition besser nachzuvollziehen und ein natürliches Steuerverhalten zu generieren.

Dabei wird angenommen, dass der Fahrer sich an Position p befindet und eine Zielposition p_0 in Entfernung r relativ zu seiner Position betrachtet. Die Orientierungen werden in Relation zur Luftlinie entlang von r beschrieben, wobei die Orientierung des Agenten durch δ und die Orientierung der Zielposition durch ϕ beschrieben wird. Die Orientierungen werden hierbei auf dem Intervall $\delta, \phi \in (\pi, \pi]$ definiert. Die Werte v und ω beschreiben die Linear- und Rotationsgeschwindigkeit des Agenten. [Park, 2016]

Die Kinematik kann somit durch Gleichung (2.10) beschrieben werden. [Park, 2016]

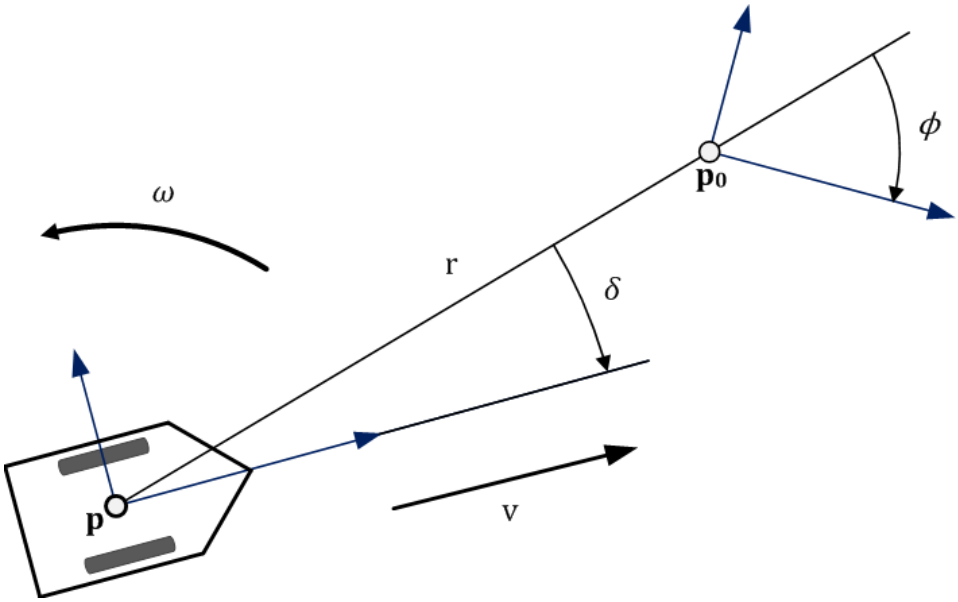

Abbildung 2.1: Egozentrische Polarkoordinaten [Park, 2016]
Der Agent p möchte hier die Position p_0 in Entfernung r ansteuern. Die Orientierung wird in Relation zur Luftlinie entlang von r beschrieben, wobei die Orientierung des Agenten durch δ und die Orientierung der Zielposition durch ϕ beschrieben wird. In der Abbildung haben sowohl δ als auch ϕ negative Werte. Im Fall, dass $r = 0$ ist, wird die Luftlinie entlang der Ausrichtung von p_0 gelegt, wodurch $\phi = 0$ ist, und δ in Relation zur Ausrichtung von p_0 gemessen wird. Die Werte v und ω beschreiben die Linear- und Rotationsgeschwindigkeit.

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} -v\cos\delta \\ \frac{v}{r}\sin\delta \\ \frac{v}{r}\sin\delta + \omega \end{pmatrix} \tag{2.10}$$

Aufbauend auf Gleichung (2.10) wurde ein Regelgesetz entwickelt, welches die Relativposition $(r, \phi, \delta)^T$ zum Ziel auf null regelt. Nehmen wir v als positiv, größer null und konstant an, so ist ω das einzige Kontrollsignal. Da ω zudem nur einen indirekten Effekt auf $(\phi, \delta)^T$ über δ hat und $(\phi, \delta)^T$ die eigentliche Position beschreiben, kann die Gleichung (2.10) in zwei Teile aufgeteilt werden. [Park, 2016]

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\delta \\ \frac{v}{r}\sin\delta \end{pmatrix} \tag{2.11}$$

$$\dot{\delta} = \frac{v}{r} \sin \delta + \omega \tag{2.12}$$

Dabei beschreibt die Gleichung (2.11) die Dynamik des Positionsunterraums, welcher das langsamere Untersystem ist, und die Gleichung (2.12) die Dynamik des Steuerungsunterraum, welcher das schnellere Untersystem ist. Die Orientierung δ ist hierbei die virtuelle Regelung im langsameren Untersystem, welches das Untersystem zur Zielposition steuert, und ω die reale Regelung im schnelleren Untersystem, welches δ hinreichend schnell zur gewünschten Regelung stabilisiert.[1] [Park, 2016]

Es ist anzumerken, dass diese Regelung analog zum menschlichen Steuerverhalten ist, wobei die Steuerung eines Lenkrads das schnellere Untersystem und die Ausrichtung des Fahrzeugs zu einem gewünschten Ziel das langsamere Untersystem ist. [Park, 2016]

Die virtuelle Regelung δ wurde von [Park, 2016] so entworfen, dass die Zielposition Lyapunov-Stabil ist,

$$\delta = \arctan(-k_\phi \phi) \tag{2.13}$$

wobei k_ϕ eine positive Konstante ist. Die reale Regelung ω ist mit der positiven Konstante k_δ, welche ungleich null ist, wie folgt definiert.

$$\omega = -\frac{v}{r}[k_\delta(\delta - \arctan(-k_\phi \phi)) + (1 + \frac{k_\phi}{1 + (k_\phi \phi)^2}) \sin \delta] \tag{2.14}$$

Mit den zuvor vorgestellten Regelgesetzen ist es möglich, einen sicheren, komfortablen, schnellen und intuitiven Weg von einer Startposition zu einer Zielposition zu berechnen. In Abbildung 2.2 sind einige Szenarien zu sehen, für die Wege von einer Startposition zu unterschiedlichen Zielpositionen berechnet wurden.

In Abhängigkeit von k_ϕ und k_δ kann die Form des Pfades verändert werden (siehe Abbildung 2.2).

[1] Für eine ausführlichere Beschreibung siehe [Park, 2016].

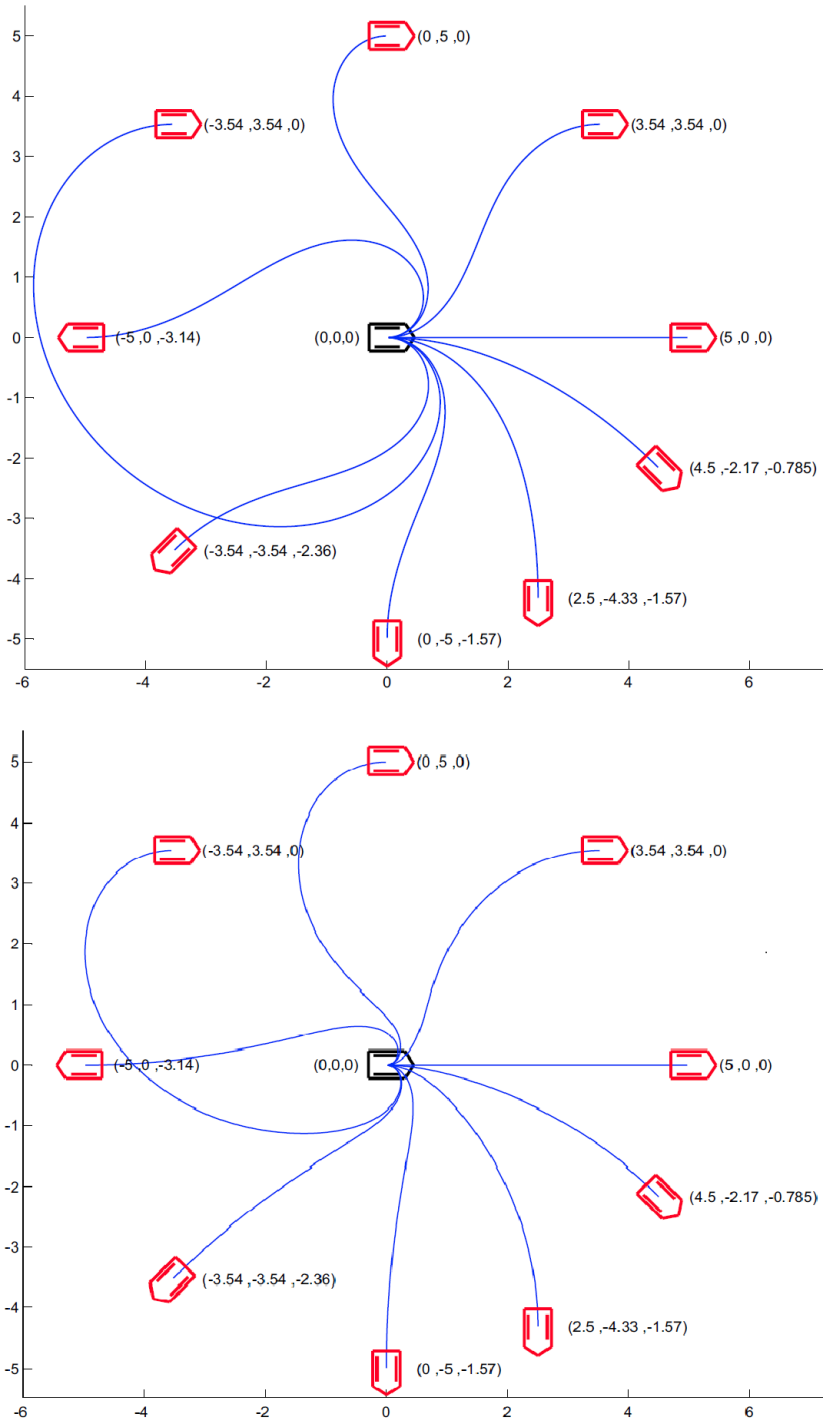

Abbildung 2.2: Beispielwege unter Verwendung der zuvor vorgestellten Regelgesetze. Die Koordinaten sind gegeben als (x, y, φ), wobei φ die Orientierung relativ zur x-Achse angibt. Die Startposition ist schwarz markiert bei $(0, 0, 0)$. Die Zielpositionen sind in rot markiert. In der oberen Abbildung sind die Konstanten $k_\phi = 1$ und $k_\delta = 3$ gewählt. In der unteren Abbildung sind die Konstanten $k_\phi = 1$ und $k_\delta = 10$ gewählt. [Park, 2016]

2.3 Distanzberechnung unter kinematischen Zwangsbeschränkungen

Zum effizienten Navigieren eines Agenten ist es notwendig, die Distanz von einer Startposition zu einer Zielposition berechnen oder einschätzen zu können. Eine einfache Funktion hierfür wäre die Euklidische Distanz. Unter nicht-holonomen Zwangsbeschränkungen, wie der Unfähigkeit sich seitlich zu bewegen, ist dieses Maß unter Umständen nicht geeignet, da es die tatsächlichen Kosten zur Zielposition nicht widerspiegelt. Beispielhaft ist dies in Abbildung 2.3 zu sehen. [Park and Kuipers, 2015]

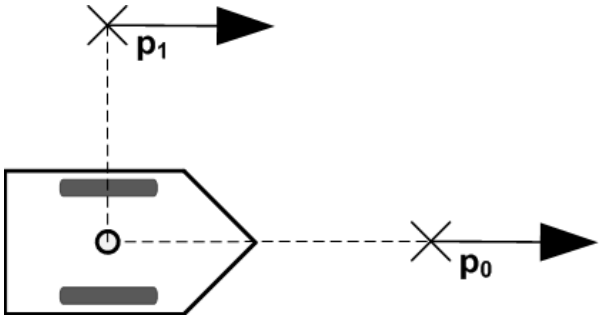

Abbildung 2.3: Es sind zwei Zielpositionen p_0, p_1 als Koordinaten (x, y, φ) gegeben. Obwohl die Euklidische Distanz zu p_1 geringer als zu p_0 ist, ist die tatsächliche Distanz zu p_1 größer. Dies ist bedingt durch die nicht-holonomen Zwangsbeschränkungen, welche ein seitwärts Bewegen des Agenten unterbinden, jedoch nicht ein geradeaus Bewegen. [Park and Kuipers, 2015]

Um Algorithmen wie den RRT* effizient erweitern zu können, ist auch eine Distanzfunktion nötig, welche die tatsächlichen Kosten zwischen zwei Positionen berechnet. Nur mit solch einer Distanzfunktion ist es dem RRT* Algorithmus möglich, sich asymptotisch dem optimalen Weg anzunähern [Park and Kuipers, 2015]. Dieses Problem wurde schon von anderen wie [Karaman and Frazzoli, 2013], [Webb and Berg, 2012] und [Park and Kuipers, 2015] behandelt.

Die Distanzfunktion von [Park and Kuipers, 2015], welche auch in [Park, 2016] auf einem RRT* Algorithmus angewandt wurde, besteht dabei aus zwei Komponenten. Die dabei verwendeten Variablen entspringen dem Polarkoordinatensystem, welche in Kapitel 2.2 genauer erklärt wurden und auch in Abbildung 2.1 nochmals sehr übersichtlich dargestellt sind.

Die erste Komponente aus Gleichung (2.15) der Distanzfunktion spiegelt die Entfernung einer (x, y) Koordinate zu einer Zielposition wider. [Park and Kuipers, 2015]

$$l^{-}(r,\phi) = \sqrt{r^2 + k_\phi^2 \phi^2} \tag{2.15}$$

Hierbei ist r die radiale Distanz zum Ziel und ϕ die Orientierung der Zielposition. Die positive Konstante k_ϕ gewichtet die Orientierung ϕ, sodass eine höhere Gewichtung von ϕ den Agenten schneller zum Ziel konvergieren lässt. Veranschaulicht wird dies in Abbildung 2.4 mit der Gewichtung von $k_\phi = 1$. [Park and Kuipers, 2015]

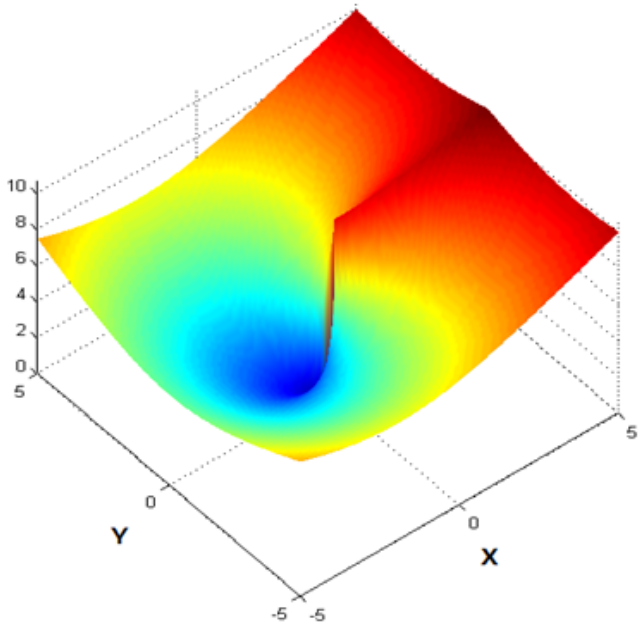

Abbildung 2.4: Visualisierung der nicht-holonomen Distanzfunktion (2.15).
Die Zielposition ist bei $(0, 0, 0)$, wobei die Orientierung $\phi = 0$ einer Orientierung parallel zur x-Achse in positiver Richtung entspricht. Die Distanz von einer (x, y) Koordinaten zum Ziel wird auf der z-Achse abgebildet. Die Gewichtung ist hierbei mit $k_\phi = 1$ gewählt. Zu bemerken ist, dass die Koordinaten kurz hinter der Zielposition sehr geringe Distanzen und die Koordinaten kurz vor der Zielposition sehr hohe Distanzen aufweisen. Die hohen Werte sind dadurch bedingt, dass sich der Agent nur vorwärtsbewegen soll, und somit eine lange Kurve fahren muss, um wieder hinter das Ziel zu kommen. [Park, 2016]

Die zweite Komponente aus Gleichung (2.16) spiegelt die Orientierung eines Agenten zu einer Zielposition wider und entspricht dabei der Differenz zwischen der Orientierung des Agenten und der gewünschten Orientierung. [Park and Kuipers, 2015]

$$\delta_e(r,\phi,\delta) = |\delta - \arctan(-k_\phi \phi)| \tag{2.16}$$

Die Gleichung 2.16 beschreibt die Abweichung zwischen der Orientierung δ und der gewünschten Orientierung $\arctan(-k_\phi \phi)$. Mit Gleichung (2.16) ist es also möglich, ungünstige Ausrichtungen des Agenten mit höheren Kosten zu belegen.

Zusammengefügt gibt die Funktion die Distanz von der Position eines Agenten zu einer Zielposition wieder. [Park and Kuipers, 2015]

$$l(r, \phi, \delta) = l^-(r, \phi) + k_\delta \delta_e(r, \phi, \delta) \tag{2.17}$$

Um die Distanz von Position p_0 zu p_1 zu berechnen, welche in den Koordinaten (x, y, φ) gegeben sind, müssen die Koordinaten erst in Polarkoordinaten entsprechend Abbildung 2.1 überführt werden. [Park and Kuipers, 2015]

$$\mathcal{P}_{p_1} : (x, y, \phi)^T \rightarrow (r, \phi, \delta)^T \tag{2.18}$$

Dann ist eine Berechnung der Distanz von p_0 zu p_1 entsprechend Gleichung (2.19) möglich. [Park and Kuipers, 2015]

$$dist(p_0, p_1) = l(\mathcal{P}_{p_1}(p_0)) \tag{2.19}$$

3 Implementierung

Dieses Kapitel beschreibt die konkrete Implementierung der in Kapitel 2 erläuterten Konzepte. Auftretende Probleme und Designentscheidungen werden diskutiert.

Es wird erläutert, wie eine effiziente Evaluation des menschlichen Steuerverhaltens möglich ist. Das Steuerverhalten wird dabei auf einem von einem Menschen gesteuerten elektrischen Rollstuhl evaluiert. Die Bewertung wird hierbei auf der Grundlage der Distanz des Fahrers zu einer Zielposition geschehen. Diese soll fortlaufend berechnet und mit den bereits berechneten Distanzen verglichen werden. Somit kann abhängig von der Veränderung der Distanz ein gutes oder inadäquates Steuerverhalten erkannt werden.

Hierbei entstehen zwei Teilprobleme:

- Das erste Teilproblem besteht darin, die Distanz des Fahrers zu einer Zielposition möglichst schnell und genau zu berechnen. Dieses Problem wird in Kapitel 3.1 behandelt.
- Das zweite Teilproblem besteht zunächst darin, die Methode der Distanzberechnung aus dem ersten Teilproblem so anzuwenden, dass sie schnell genug ist, die Distanz fortlaufend zu berechnen. Nachfolgend kann ein Vergleich der Distanzen und eine entsprechende Bewertung des Steuerverhaltens geschehen. Dieses Problem wird in Kapitel 3.2 behandelt.

3.1 Nicht-holonomer RRT*

Die Herausforderungen für die in diesem Kapitel behandelten Problemen bestehen darin, die Distanz des Fahrers zu einer Zielposition unter nicht-holonomen Zwangsbeschränkungen möglichst schnell und genau zu berechnen. Der in Kapitel 2.1.2 vorgestellte RRT* Algorithmus bietet hierfür einen guten Anhaltspunkt. Die folgenden Analysen und Grafiken wurden auf der Karte aus Abbildung 3.1 angefertigt.

Der RRT* Algorithmus benötigt zur Wegfindung keinen vorher generierten Graphen und der ermittelte Weg nähert sich dem optimalen Weg mit steigender Anzahl der Stichproben asymptotisch an [Park, 2016]. Dies ist besonders gut in Abbildung 3.2 zu sehen.

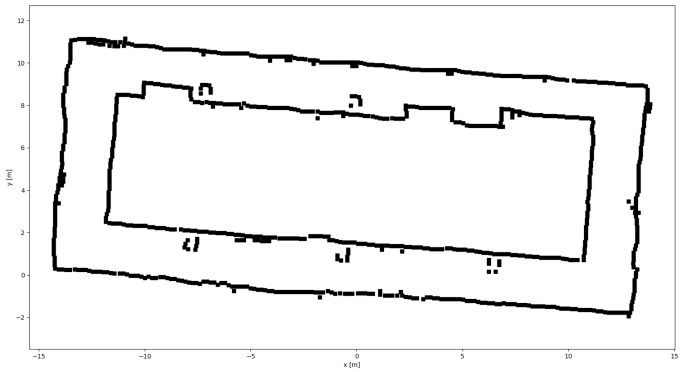

Abbildung 3.1: Karte eines Flurs.

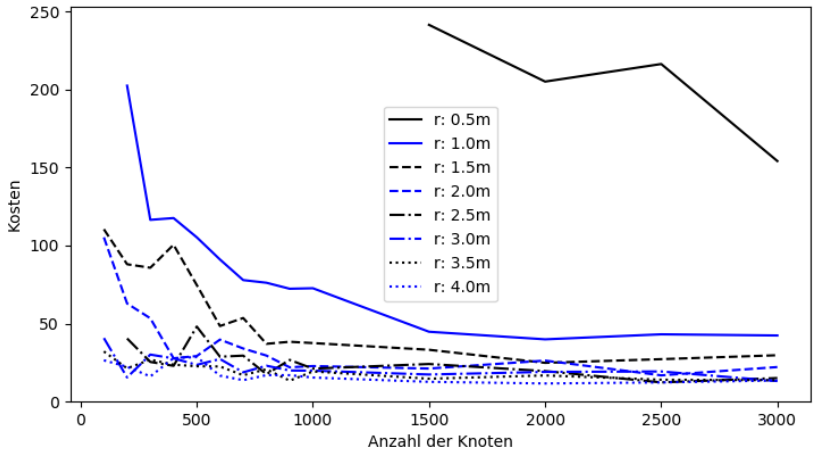

Abbildung 3.2: Größe des RRT* Graphen gegenüber den berechneten Kosten bei verschiedenen Suchradien der Funktionen 2.6 und 2.8 (nearTo und nearFrom). Die Konstanten der Gleichung 2.14 wurden hierbei mit $k_\phi = 1.2$ und $k_\delta = 10$ gewählt.

Hierbei wurden für verschiedene Suchradien der Funktionen aus Gleichung (2.6) und (2.8) (nearTo und nearFrom) Graphen berechnet. Auf der x-Achse sind dabei die Anzahl der Knoten und auf der y-Achse die Kosten zu einer Zielposition aufgetragen. Die teilweise ansteigenden Kosten bei steigender Knotenanzahl sind dabei durch die Neuberechnungen der Graphen und damit einhergehenden zufälligen Stichproben bedingt. Im Durchschnitt ist jedoch eine asymptotische Annäherung an einen optimalen niedrigen Wert zu betrachten. Zudem ist auch ersichtlich, dass ein höherer Suchradius die Kosten schneller gegen einen optimalen Wert konvergieren lässt.

Ein großer Nachteil des RRT* Algorithmus ist jedoch, dass er relativ viel Berechnungszeit benötigt, um einen näherungsweise optimalen Pfad und dessen Distanz zu

bestimmen. Dies ist in Abbildung 3.3 nochmal deutlicher zu erkennen.

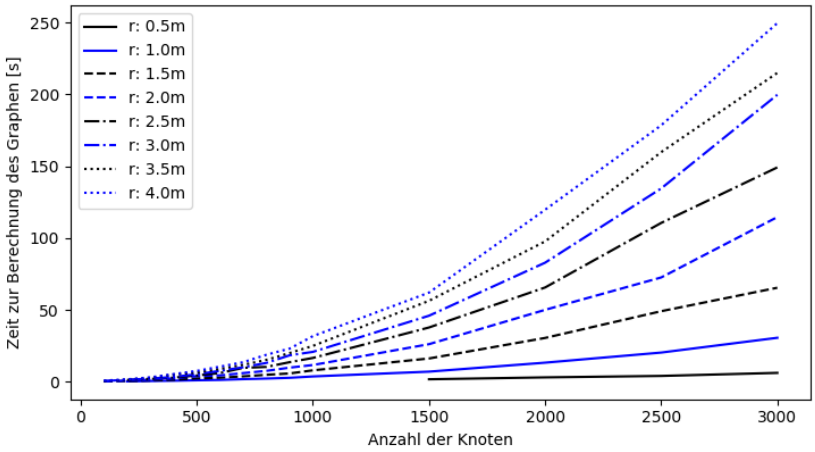

Abbildung 3.3: Größe des RRT* Graphen gegenüber der Berechnungszeit des Graphen bei verschiedenen Suchradien der Funktionen 2.6 und 2.8 (nearTo und nearfrom). Zu bemerken ist, dass die Berechnungszeiten für steigende Anzahl an Knoten exponentiell ansteigen. Dabei führt ein größerer Suchradius zu einem schnelleren Anstieg der Berechnungszeit.

In Abbildung 3.3 wurden erneut für verschiedene Suchradien der Funktionen (2.6) und (2.8) (nearTo und nearfrom) Graphen berechnet. Auf der x-Achse sind dabei die Anzahl der Knoten und auf der y-Achse die Berechnungszeiten aufgetragen. Die Berechnungszeiten steigen dabei exponentiell mit steigender Anzahl an Knoten an, wobei ein größerer Suchradius zu einem verstärkten Wachstum der Berechnungszeiten führt.

Auf Abbildung 3.2 ist zu erkennen, dass der Suchradius mit vier Metern am schnellsten gegen einen optimalen Wert konvergiert und ab schätzungsweise 1000 in den Graphen eingefügten Knoten sich nicht mehr erheblich verändert. In Abbildung 3.3 hat der Graph für einen Suchradius von vier Metern jedoch eine Berechnungszeit von ungefähr 35 Sekunden. Dies ist eine zu lange Berechnungszeit, um auf den berechneten Distanzen eine angemessene Bewertung des Fahrverhaltens durchzuführen, da mit einem Rollstuhl innerhalb von nur wenigen Sekunden schon mehrere Meter zurückgelegt werden können.

Eine schnellere Berechnungszeit der Distanz ist durch den A* Algorithmus möglich, da er zunächst keinen eigenen Graphen generieren muss. Jedoch muss diesem Algorithmus zu Beginn ein Graph bereitgestellt werden. Da sowohl in dem Ansatz mit RRT* als auch mit A* eine vorher generierte Karte benötigt wird, wurde die Implementierung so geändert, dass für eine Karte auch ein entsprechender Graph generiert wird. Im Folgenden wird eine Methode zur Generierung eines gerichteten Graphens unter Abwandlung des RRT* Algorithmus vorgestellt, auf welchem der A* Algorithmus angewandt wird.

3.1.1 Generierung des Graphens

Der generierte Graph soll gerichtet und die Knoten sollen auf dem navigierbaren Raum der Karte gleich verteilt sein. Die Kosten der Kanten werden durch die Distanzfunktion (2.19) repräsentiert, welche die nicht-holonomen Zwangsbeschränkungen eines Rollstuhls berücksichtigt.

Der Algorithmus zur Generierung des Graphens ist eine Abwandlung des RRT* Algorithmus, welcher in Kapitel 2.1.2 beschrieben ist. Dieser abgewandelte Algorithmus ist in Algorithmus 4 beschrieben.

Algorithm 4 $graph \leftarrow \text{RRTgraph}(start, lim)$

```
count ← 0
graph ← []
graph.append(start)
while count < lim do
  new ← sample()
  nearest ← getNearest(new, graph)
  new ← extend(nearest, new)
  if new == NULL then
    continue
  end if
  graph.append(new)
  neighbors ← getNeighbors(new, graph)
  linkNeighbors(neighbors, new)
  counter ← counter + 1
end while
return graph
```

Der Algorithmus wurde dahingehend verändert, dass nicht mehr der Vorgängerknoten eines Knotens in *parent* gespeichert wird, sondern alle Nachbarn eines Knotens, welche von dem betrachteten Knoten aus erreicht werden können. Hierfür werden zunächst alle Nachbarn *neighbors* eines neu generierten Knotens v_{new} herausgesucht.

$$getNeighbors(v_{new}, graph) \equiv \{v \in graph : euclDist(v, v_{new}) \leq radius\} \tag{3.1}$$

Die verwendete Distanzfunktion ist nicht mehr die Distanzfunktion von Gleichung 2.19, sondern die euklidische Distanz *euclDist*, welche den Abstand zwischen zwei Koordinaten bestimmt. Dadurch ist eine Verwendung der in NumPy[1] enthaltenen Funktion *numpy.linalg.norm* möglich, welche eine schnellere Distanzberechnung auf Grundlage

[1] Die Programmierbibliothek NumPy der Programmiersprache Python ist auf wissenschaftliches Rechnen spezialisiert. [NumPy, oD]

der euklidischen Distanz ermöglicht. Zudem muss nur noch einmal pro Schleifenschritt *neighbors* berechnet werden, was die Berechnungsgeschwindigkeit erneut erhöht.

Die Funktion *linkNeigbors* wird verwendet, um die Nachbarn von v_{new} mit v_{new} zu verbinden. Sie fügt alle $neighbor \in neighbors$ in $v_{new}.neighbors$ hinzu, sofern eine valide Verbindung von v_{new} zu *neighbor* existiert. Um den Knoten v_{new} auch erreichen zu können, werden auch alle Verbindungen von *neighbor* zu v_{new} überprüft und entsprechend v_{new} in $neighbor.neighbors$ eingefügt. Die Überprüfung einer Verbindung auf Validität geschieht hierbei durch die Funktion $validLink(v_{from}, v_{to})$.

Die Überprüfung der Verbindung durch die Funktion *validLink* erfolgt dabei wie folgt und ist in Abbildung 3.4 beispielhaft dargestellt.

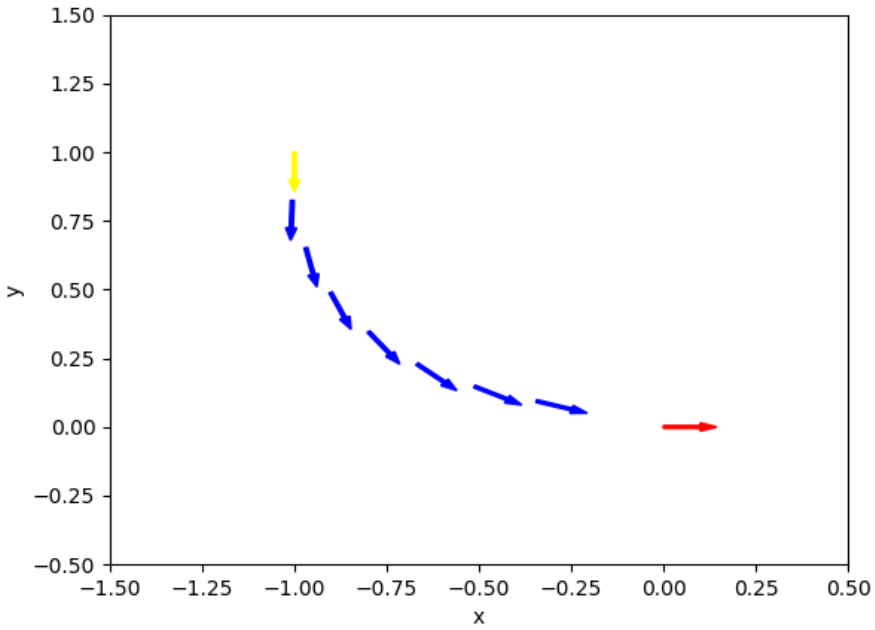

Abbildung 3.4: Simulierte Bewegung von $v_1 = (-1.0, 1.0, -1.57)$ zu $v_2 = (0.0, 0.0, 0.0)$. Hierbei ist v_1 in gelb, v_2 in rot und die simulieren Schritte von v_1 auf dem Weg zu v_2 in blau dargestellt. Die Konstanten der Gleichung 2.14 wurden hierbei mit $v = 0.22$, $k_\phi = 1.2$ und $k_\delta = 10$ gewählt.

Der Startknoten v_1 und der Zielknoten v_2 der zu überprüfenden Verbindung werden mithilfe der Funktion (2.18) in das Polarkoordinatensystem (r, ϕ, δ) aus Abbildung 2.1 überführt. Nehmen wir nun eine konstant positive Lineargeschwindigkeit v größer null an, so kann mithilfe der Gleichung 2.14 die Rotationsgeschwindigkeit ω berechnet werden. Unter Verwendung dieser beiden Geschwindigkeiten und der Updatefunktion (2.10) kann durch wiederholte Berechnung von ω eine simulierte Bewegung von v_1 in Richtung v_2 durchgeführt werden. Nach jedem Schritt in Richtung v_2 wird die Position von v_1

auf Kollisionen mit Objekten der Karte überprüft. Ist jede simulierte Position von v_1 auf dem Weg zu v_2 kollisionsfrei, so wird die Verbindung von v_1 zu v_2 als kollisionsfrei und somit valide angesehen. Die Schrittweite bei jedem Simulationsschritt wurde in der gesamten Arbeit als konstant angesehen.

Nachdem der Graph erfolgreich aufgebaut wurde, kann dieser verwendet werden, um auf ihm mit dem A* Algorithmus ein Weg von einer beliebigen Startposition zu ein beliebigen Zielposition zu finden.

3.1.2 Anwendung des A* Algorithmus

Eine schnelle und genaue Distanzberechnung von einer Startposition zu einer Zielposition ist im ersten Teilproblem besonders wichtig. Der A* Algorithmus ermöglicht es, den optimalen Weg auf einem gegebenen Graphen im Regelfall sehr schnell zu finden [LaValle, 2006] [Park, 2016]. Im Folgenden wird aufgezeigt, dass diese schnelle Wegfindung auf dem Graphen aus Kapitel 3.1.1 möglich ist.

Die Wegfindung auf dem generierten Graphen aus Kapitel 3.1.1 geschieht mit Hilfe des A* Algorithmus, welcher in Kapitel 2.1.1 beschrieben ist. Hierfür werden zunächst der Startknoten und der Zielknoten mit den Funktionen *getNeighbors* und *linkNeigbors* (beschrieben in Kapitel 3.1.1) in den Graphen eingefügt. Anschließend ist eine Anwendung des A* Algorithmus auf dem Graphen wie beschrieben möglich.

Die Berechnungszeiten des A* Algorithmus für verschiedene Graphen bei unterschiedlichen Wegen ist in Abbildung 3.5 zu sehen. Die dazugehörige Karte mit den entsprechenden Start- und Zielpositionen ist in Abbildung 3.6 abgebildet.

Für die Berechnungen wurde der Suchradius auf vier Meter gesetzt, da dieser in Abbildung 3.2 am schnellsten gegen den optimalen Wert für die Kosten konvergiert ist. Wie in Abbildung 3.6 zu sehen ist, sind die Zielpositionen in regelmäßigen Abständen entlang des Flurs platziert und haben alle die gleiche Startposition.

In Abbildung 3.5 (unten) ist zu erkennen, dass die Kosten ab ungefähr 1000 Knoten im Graphen wenig schwanken und nur noch leicht konvergieren, wobei die weiter entfernten Ziele (Ziel 4 und 5) noch etwas länger zum Konvergieren brauchen als die näher gelegenen Ziele (Ziel 0 und 1). Betrachten wir also in Abbildung 3.5 (oben) die Stelle der x-Achse mit 1000 Knoten, so ist zu erkennen, dass die Berechnungszeiten für näher gelegene Ziele unter einer Sekunde beträgt (siehe Ziel 0 und 1), jedoch für weiter entfernte Ziele über eine Sekunde betragen kann (siehe Ziel 5). Bei größeren Graphen verstärkt sich dieser Unterschied zunehmend.

Die schnelle und genaue Berechnung eines Weges ist also für kürzere Wege durchaus möglich, jedoch steigt die Berechnungszeit mit der Länge des Weges und der Komplexität

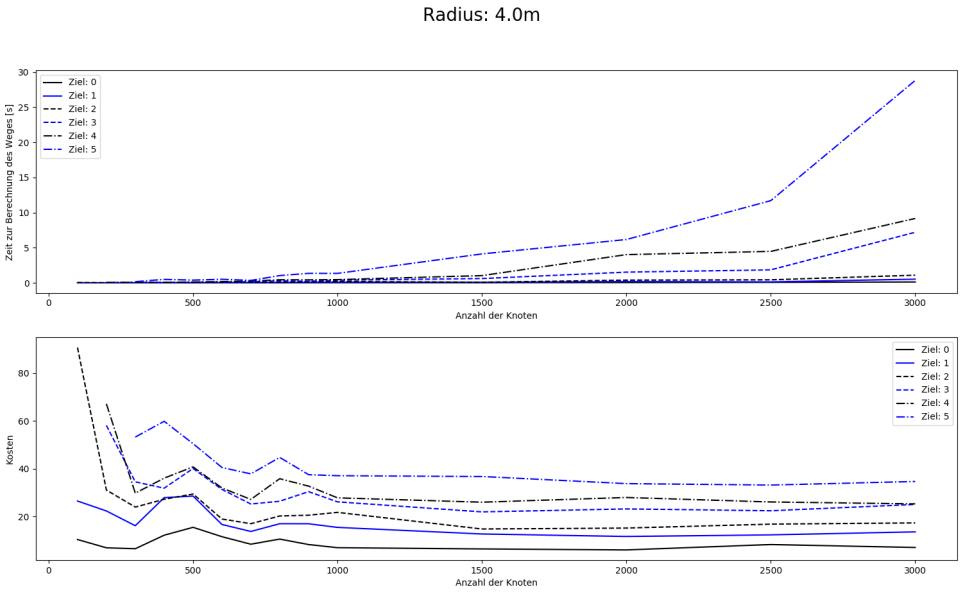

Abbildung 3.5: Analyse der in Abbildung 3.6 beschriebenen Wege. Die Konstanten der Gleichung 2.14 wurden hierbei mit $k_\phi = 1.2$ und $k_\delta = 10$ gewählt.
Unten: Größe des Graphen gegenüber den Kosten des Weges von der Startposition zur entsprechenden Zielposition. Ab einer Größe des Graphen von ungefähr 1000 Knoten schwanken die Kosten nur noch minimal. Die weiter entfernten Ziele (Ziel 4 und 5) brauchen zum Konvergieren gegen den optimalen Wert etwas länger als die näher gelegenen Ziele (Ziel 0 und 1).
Oben: Größe des Graphen gegenüber der Berechnungszeit des Weges bei unterschiedlichen Entfernungen zwischen Start-und Zielposition. Zu erkennen ist, dass die Berechnungszeit mit der Größe des Graphen und der Länge des Weges zunimmt.

des Graphen.

3.2 Bewertung menschlichen Steuerverhaltens

Nachdem in Kapitel 3.1 eine Methode zur schnellen Berechnung der Distanz vorgestellt wurde, wird in diesem Kapitel die wiederholte Berechnung der Distanz und die Bewertung des menschlichen Steuerverhaltens behandelt.

Zur Vereinfachung nehmen wir zunächst an, dass der Fahrer im Groben den zuvor berechneten Weg wählt. Somit können vor dem Antritt der Fahrt ein oder mehrere aufeinander aufbauende Wege mit dem A* Algorithmus auf dem gegebenen Graphen berechnet werden. Diese Wege müssen dann aufgrund der Vereinfachung nicht mehr neu berechnet werden. Vor der Fahrt sind Zwischenziele und Distanzen zwischen den Zwi-

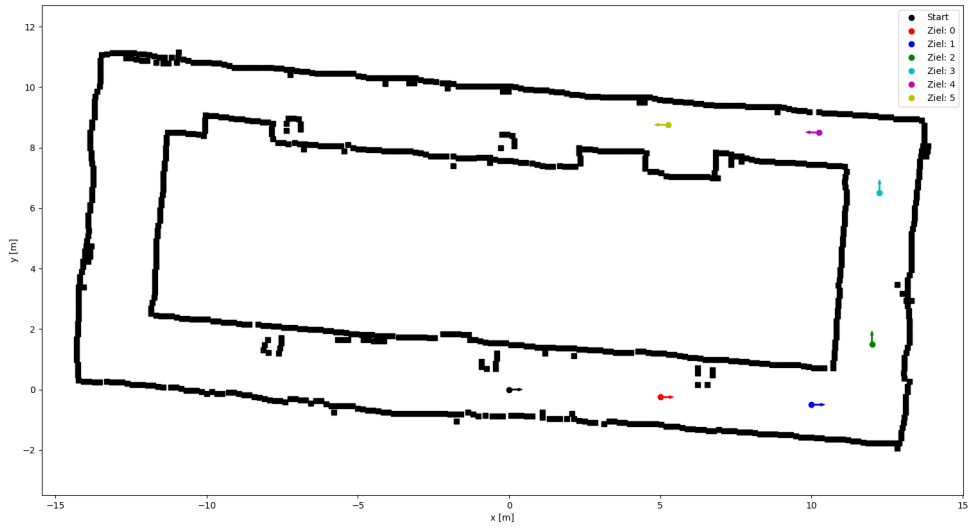

Abbildung 3.6: Abbildung 3.1 mit einer Startposition und verschiedenen Zielpositionen. Die Startposition ist in schwarz bei $(0, 0, 0)$ eingezeichnet. Die Zielpositionen sind in regelmäßigen Abständen den Flur entlang platziert. Die berechneten Wege von der Startposition zu den entsprechenden Zielpositionen werden in Abbildung 3.5 analysiert.

schenzielen gegeben. Die Gesamtdistanz berechnet sich durch die Summe der Distanz zum ersten Zwischenziel und der Distanzen zwischen den noch kommenden Zwischenzielen bis zum endgültigen Ziel.

$$Dist_G = dist(p, ig_{next}) + \sum_{i=next+1}^{goal} dist(ig_{i-1}, ig_i) \tag{3.2}$$

Hierbei beschriebt $Dist_G$ die Gesamtdistanz, p die aktuelle Position des Fahrers, ig (*interim goal*) ein Zwischenziel und *goal* das endgültige Ziel. Eine schnelle und wiederholte Berechnung wäre mit Gleichung (3.2) möglich, jedoch würde die Distanz beim Erreichen eines Zwischenziels einen Sprung machen. Dies ist dadurch bedingt, dass beim Umschalten auf das nächste Zwischenziel die Komponente $dist(p, ig_{next})$ abrupt wegfallen würde. Diese Komponente wäre beim Umschalten nur null, wenn der Fahrer exakt den berechneten Weg abfahren würde. Eine genaue Berechnung ist mit dieser Methode somit nicht möglich.

Die Sprünge können vermieden werden, indem die Distanz zum übernächsten Zwischenziel an Gewichtung gewinnt, je näher der Fahrer dem nächsten Zwischenziel kommt.

Somit würde die Komponente $dist(p, ig_{next})$ immer weiter an Gewichtung verlieren, bis sie beim Umschalten auf das nächste Zwischenziel keine Gewichtung mehr besitzt und in der Gesamtdistanz keinen Sprung mehr verursachen kann. Die Berechnung der Gesamtdistanz würde somit wie folgt aussehen.

$$Dist_G = \theta \cdot dist(p, ig_{secondnext}) + (1-\theta)(dist(p, ig_{next}) + dist(ig_{next}, ig_{secondnext}) + \sum_{i=next+2}^{goal} dist(ig_{i-1}, ig_i) \tag{3.3}$$

Die Gewichtung $\theta \in [0, 1]$ soll auf der Teilstrecke zwischen dem zuletzt erreichten Zwischenziel und dem nächsten Zwischenziel kontinuierlich mit dem Vorankommen des Fahrers zunehmen. Sofern der Fahrer auf dem direkten Weg entlang der Luftlinie zwischen den beiden Punkten fährt, kann θ wie folgt berechnet werden.

$$\theta = \frac{euclDist(ig_{previous}, p)^2}{euclDist(ig_{previous}, p)^2 + euclDist(p, ig_{next})^2} \tag{3.4}$$

Durch die Gleichung 3.4 entsteht beim Abfahren des direkten Weges zwischen zwei Zwischenzielen eine Kurve entsprechend Abbildung 3.7. Beim Erreichen eines Zwischenziels fällt θ nach der Steigung bis zu $\theta = 1$ wieder auf $\theta = 0$ zurück und beginnt mit einem neuen Anstieg.

Da der Fahrer jedoch nicht immer auf dem direkten Weg zwischen den beiden Punkten fährt, muss die Position des Fahrers auf diesen Weg projiziert werde. Veranschaulicht wird dies in Abbildung 3.8.

Es wird zunächst die Höhe h_c des Dreiecks, von c aus betrachtet, berechnet. Die Länge von c_1 kann dann mit dem Satz des Pythagoras und c_2 mit c_1 und c berechnet werden. Die genaue Berechnung von c_1 und c_2 ist in den Gleichungen 3.5 bis 3.8 beschrieben. Die Längen von c_1 und c_2 entsprechen dann $euclDist(ig_{previous}, p)$ und $euclDist(p, ig_{next})$ aus Gleichung 3.4.

$$s = \frac{1}{2}(a+b+c) \tag{3.5}$$

$$h_c = \frac{2}{c}\sqrt{s(s-a)(s-b)(s-c)} \tag{3.6}$$

$$c_1 = \sqrt{b^2 - h_c^2} \tag{3.7}$$

$$c_2 = c - c_1 \tag{3.8}$$

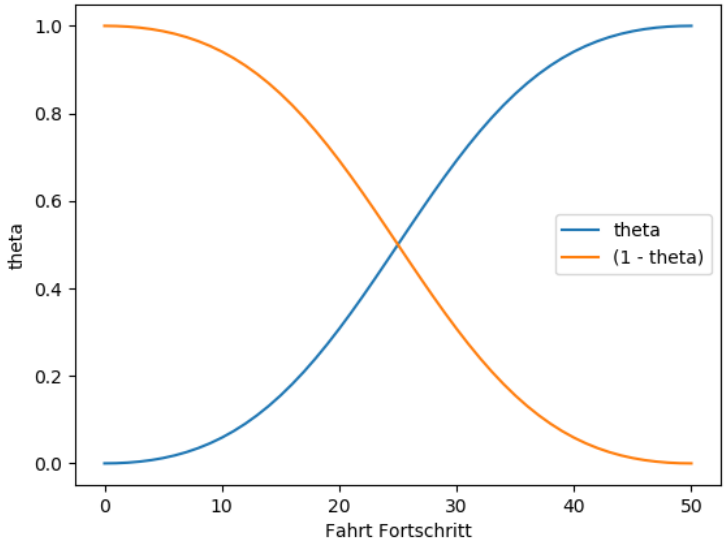

Abbildung 3.7: Beispielhafter Anstieg und Abfall von θ während des Abfahrens eines Weges zwischen zwei Zwischenzielen. Der Abfall von θ ist dabei durch das Erreichen des nächsten Zwischenziels bedingt.

Für die Anwendung von θ in Gleichung (3.3) muss $\theta \in [0, 1]$ sein. Dies ist mit der zuvor beschriebenen Methode der Projektion nur gegeben, wenn die Winkel $\alpha, \beta \leq \frac{\pi}{2}$ sind. Es bietet sich demnach beim Erreichen von $\alpha = \frac{\pi}{2}$ an, auf das nächste Zwischenziel umzuschalten, wodurch das Verfehlen eines Zwischenziels nahezu unmöglich wird.

Es existiert ein Übergangsbereich in der Nähe der Zwischenziele, was einen glatteres Umschalten zum nächsten Zwischenziel ermöglicht. Hierfür werden die Zwischenziele $ig_{previous}$ und ig_{next} um eine Distanz der Länge r in Richtung des entsprechend anderen Zwischenziels verschoben.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ig_0.x + r\cos\alpha \\ ig_0.y + r\sin\alpha \end{pmatrix} \tag{3.9}$$

Der Winkel α beschreibt den Winkel zwischen dem Vektor $\overrightarrow{ig_0 ig_1}$ und der x-Achse in positiver Richtung. Die Gewichtung θ wird somit nur noch zwischen den beiden verschobenen Punkten verändert und bei $\beta \geq \frac{\pi}{2}$ bleibt $\theta = 0$. Genauer veranschaulicht ist dies in Abbildung 3.9.

Die Bewertung des menschlichen Steuerverhaltens geschieht dann wie folgt:

Die während der Fahrt mit der Gleichung (3.3) berechneten Distanzen werden mit ihren vorangegangenen Distanzen verglichen. Eine negative Bewertung geschieht dann,

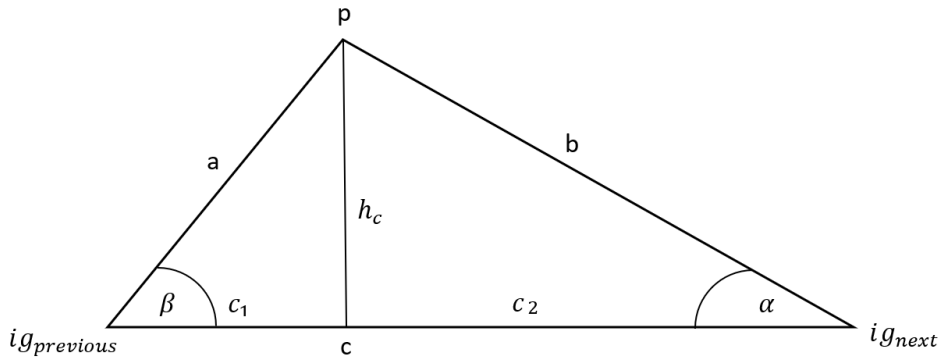

Abbildung 3.8: Projektion der Fahrerposition auf den direkten Weg zwischen den Zwischenzielen $ig_{previous}$ und ig_{next}. Die projizierte Position ist die Schnittstelle von h_c und c.

Abbildung 3.9: Veranschaulichung des Übergangsbereichs von θ. Die beiden Zwischenziele $ig_{previous}$ und ig_{next} sind um eine Distanz der Länge r in Richtung des entsprechend anderen Zwischenziels verschoben. Die Gewichtung θ wird nur zwischen den beiden verschobenen Punkten verändert.

wenn die betrachtete Distanz im Vergleich zu ihren Vorgängern über eine gewählte Zeit angestiegen ist. In diesen Bereichen wurde somit nicht der kürzeste Weg gewählt, wodurch sich die Distanz zum Ziel durchgängig über einen gewissen Zeitraum vergrößert hat.

4 Experimentelle Evaluation

In diesem Kapitel wird die in Kapitel 3 vorgestellte Methode zur Bewertung des menschlichen Steuerverhalten an einem ferngesteuerten Roboter und einem elektrischen Rollstuhl evaluiert. Beide Evaluationen wurden in einer physikalischen Umgebung durchgeführt. In Kapitel 4.1 wird ein mit einer Tastatur ferngesteuerter Roboter auf einer $6m \cdot 7m$ großen Karte evaluiert. Es wurden zudem relativ viele zu passierende Wegpunkte vorgegeben, um die Aufgabe weiter zu vereinfachen. Im Experiment, das in Kapitel 4.2 beschrieben wird, wurde hingegen ein elektrischer Rollstuhl mithilfe eines Joysticks durch einen langen Flur gefahren, wobei der Rollstuhl nicht wie in Kapitel 4.1 ferngesteuert wurde, sondern der Fahrer während der Fahrt auf dem Rollstuhl saß und somit aus der Egoperspektive navigieren konnte. Hierbei wurden nur wenige zu passierende Wegpunkte vorgegeben, sodass, verglichen mit den Wegen zuvor, längere Wege berechnet werden mussten.

Wie in den Beispielen zu erkennen ist, werden Zwischenziele so generiert, dass sie nacheinander auf dem Weg von der Startposition zur Zielposition mit sicheren, schnellen und intuitiven Bewegungen abgefahren werden können. Anhand dieses Weges aus Zwischenzielen und der Distanz zum Ziel wird daraufhin eine Bewertung des Steuerverhaltens vorgenommen.

Zu bemerken ist, dass die generierten Wege teilweise dem Zufall überlassen sind und sich mit steigender Komplexität des Graphens dem optimalen Weg nur annähern. Sofern der Graph nicht unendlich groß ist und somit der Weg nicht optimal wird, besteht immer die Möglichkeit einer Fehlinterpretation des Steuerverhaltens.

4.1 Evaluation eines ferngesteuerten Roboters

In diesem Kapitel wird gezeigt, wie das Bewertungsverfahren an einem ferngesteuerten Roboter getestet wurde. Hierfür wurde ein abgegrenzter Bereich geschaffen. Um eine Karte von diesem Bereich zu erstellen, wurde mit dem Roboter der gesamte navigierbare Bereich abgefahren. Mit Hilfe seiner Sensordaten konnte die Struktur der Umgebung erkannt und in eine Karte exportiert werden. Die Wände der Karte wurden herausgefiltert und in Objekte umgewandelt, welche im folgenden Wegfindungsprozess verwendet

werden konnten. In Abbildung 4.1 ist dargestellt, wie diese Karte in umgewandelter Form aussieht.

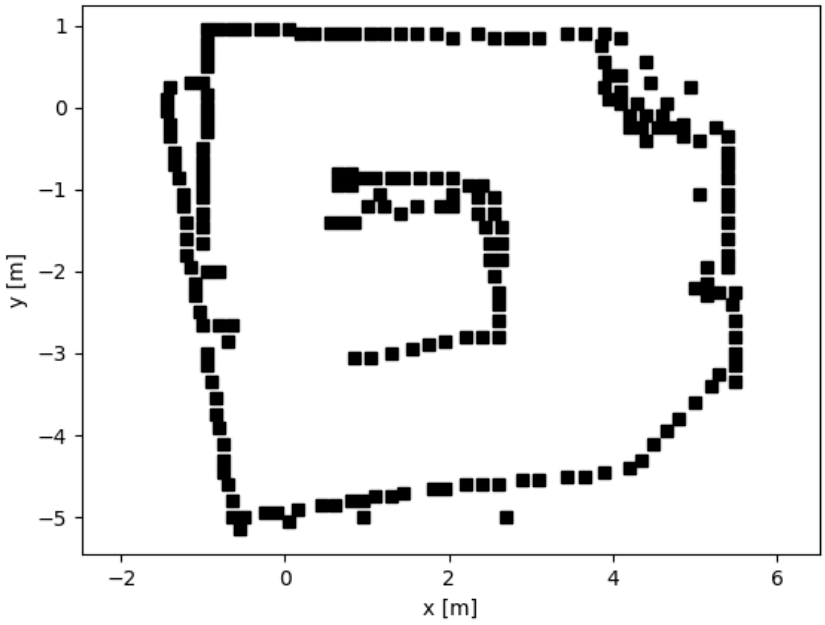

Abbildung 4.1: Karte eines abgegrenzten Bereichs in abgewandelter Darstellung. Jedes schwarze Viereck repräsentiert ein Objekt, welches in einem wählbaren Radius nicht passiert werden kann. Die Objekte symbolisieren die Wände der vom Roboter erstellten Karte.

Mit Hilfe dieser Darstellung einer Karte kann eine Position auf Validität überprüft werden, indem die Distanz zu allen Objekten überprüft wird. Ist die Distanz zu allen Objekten der Karte größer als der gewählte Radius, so gilt eine Position als valide. Dieser Radius wurde im Folgenden dauerhaft auf 0.25 Meter gesetzt.

Nach dem Erfassen der Karte wurde auf dieser ein Graph mit 2000 Knoten generiert. Bei der Generierung wurden die Konstanten $k_\phi = 1.2$, $k_\delta = 10$ und für Funktion (3.1) der $Suchradius = 4m$ gewählt.

Die Aufgabe des Fahrers wurde so definiert, dass er an der Position $(0, 0, 0)$ starten und nacheinander die Wegpunkte $(3.05, -0.5, -0.785), (2, -3.7, 3.14), (2.2, -1.6, 0)$ abfahren muss. Zu Beginn der Fahrt wurden vom A* Algorithmus Wege zwischen den Wegpunkten berechnet, welche zu einer langen Abfolge von Zwischenzielen aneinandergereiht wurden. Die Wegpunkte wurden nacheinander abgefahren, wodurch sich der in Abbildung 4.2 dargestellte gefahrene Pfad ergab.

Während der Fahrt wurde das Steuerverhalten des Fahrers fortlaufend analysiert, wobei für die Analyse die Konstante auf $k_\delta = 1$ geändert wurde. Diese ist in Abbildung

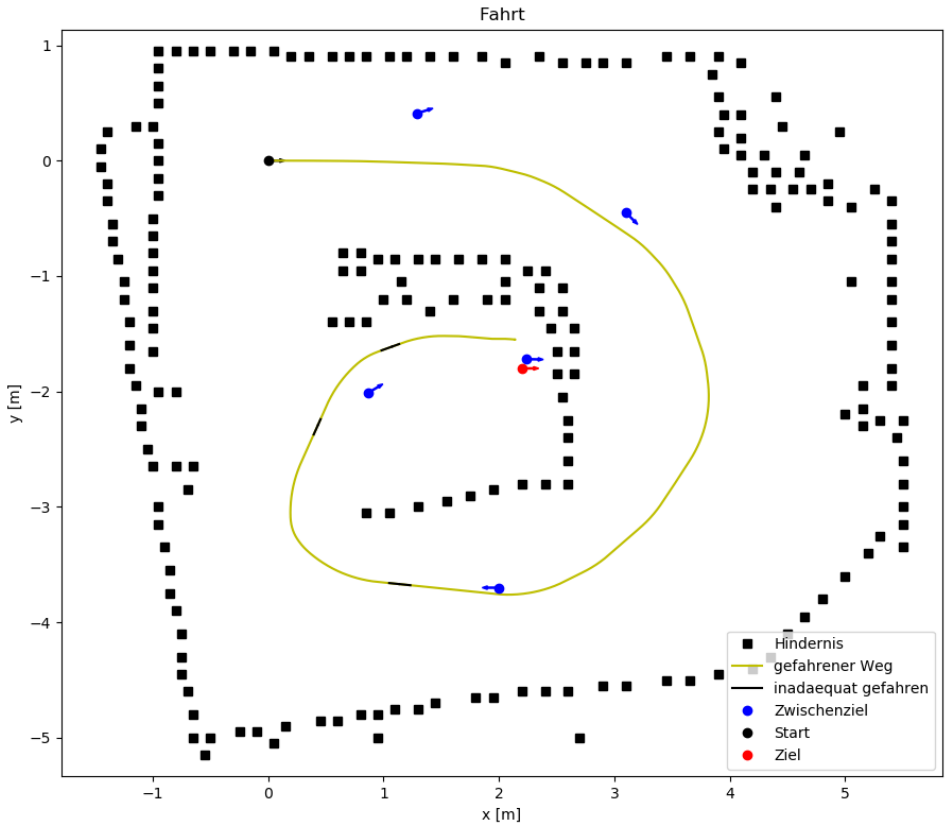

Abbildung 4.2: Der vom Fahrer gefahrene Weg beim Abfahren aller Wegpunkte. Auf schwarz markierten Streckenabschnitten wurde so gefahren, dass sich die Distanz zum Ziel erhöhte. Eine genauere Analyse ist in Abbildung 4.3 zu betrachten.

4.3 dargestellt.

Inadäquat gefahrene Streckenabschnitte wurden in beiden Abbildungen schwarz markiert. Als inadäquat gefahren wird ein Streckenabschnitt dann bezeichnet, wenn sich die Distanz zum Ziel über mehr als 0.5 Sekunden erhöht hat. Andernfalls wird eine positive Bewertung vorgenommen. Je schneller die Distanz in Abbildung 4.3 abfällt, desto besser wurde der entsprechende Streckenabschnitt abgefahren.

Das Analyseprogramm wurde beendet, nachdem der Fahrer dem Ziel näher als 0.5 Meter gekommen ist. Eine weitere Analyse auf solch kurze Distanz wäre für den Fahrer nicht mehr hilfreich, da die Distanz sich nur noch weiter verringert, wenn die Zielposition exakt angesteuert wird. Sobald der Fahrer neben oder über das Ziel hinausfahren würde, würde die Distanz zum Ziel wieder ansteigen. Verbildlicht wurde dies zuvor in Abbildung 2.4.

Es ist zu erkennen, dass der Fahrer meist gut gefahren ist. So wurde in den ersten zwei Dritteln der Fahrt kein einziger gravierender Fehler begangen. Zwischen dem ersten und dem zweiten Zwischenziel ist zudem der schnellste Abfall der Distanz verzeichnet

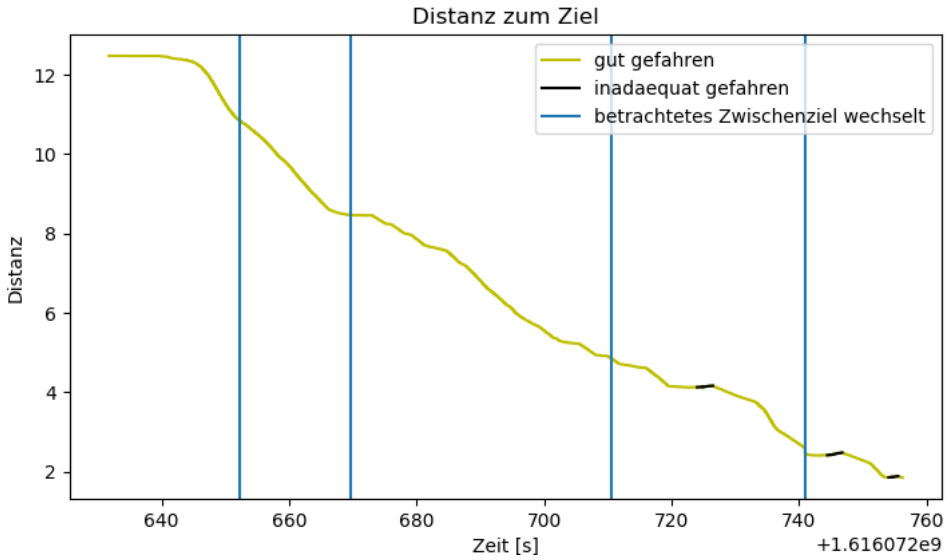

Abbildung 4.3: Die Veränderung der Distanz im Verlauf der Fahrt. Die Konstanten der Distanzfunktion (2.19) wurden hierbei mit $k_\phi = 1.2$ und $k_\delta = 1$ gewählt. Solange sich die Distanz nicht länger als 0.5 Sekunden erhöhte, wurde der gefahrene Streckenabschnitt als gut bewertet, andernfalls als inadäquat. Die vertikalen Linien markieren die Zeiten, an denen ein Zwischenziel erreicht wurde.

worden, weshalb dies der bestgefahrene Streckenabschnitt ist.

Im letzten Drittel der Fahrt, kurz nach dem dritten Zwischenziel, wurden jedoch einige größere Fehler begangen. Sie wurden jedoch schnell vom Fahrer erkannt und korrigiert, was an der kurzen Dauer des jeweiligen Fehlverhaltens zu erkennen ist.

Der erste Fehler wurde zwischen dem dritten und vierten Zwischenziel begangen. Es ist zu vermuten, dass zu lange geradeaus gefahren wurde, wodurch sich eine größere zu fahrende Kurve abbildete, welche die Distanz erhöhte. Der gleiche Fehler wiederholte sich dann vermutlich beim zweiten und dritten inadäquat gefahrenen Streckenabschnitt.

4.2 Evaluation einer Rollstuhlfahrt

In diesem Kapitel wird beschrieben, wie das Bewertungsverfahren während der Fahrt auf einem elektrischen Rollstuhl getestet wurde. Die Fahrt wurde im Flur eines Bürogebäudes durchgeführt. Wie in Kapitel 4.1 wurde der Flur beim Abfahren mit den Sensoren des Rollstuhls erfasst und eine Karte wurde in abgewandelter Form erzeugt. Diese Karte ist in Abbildung 4.4 zu sehen.

Für diese Karte wurde ein Graph mit 5000 Knoten generiert. Hierbei wurden die wählbaren Konstanten mit $k_\phi = 1.2$, $k_\delta = 10$ und für Funktion 3.1 der $Suchradius = 6m$ gewählt.

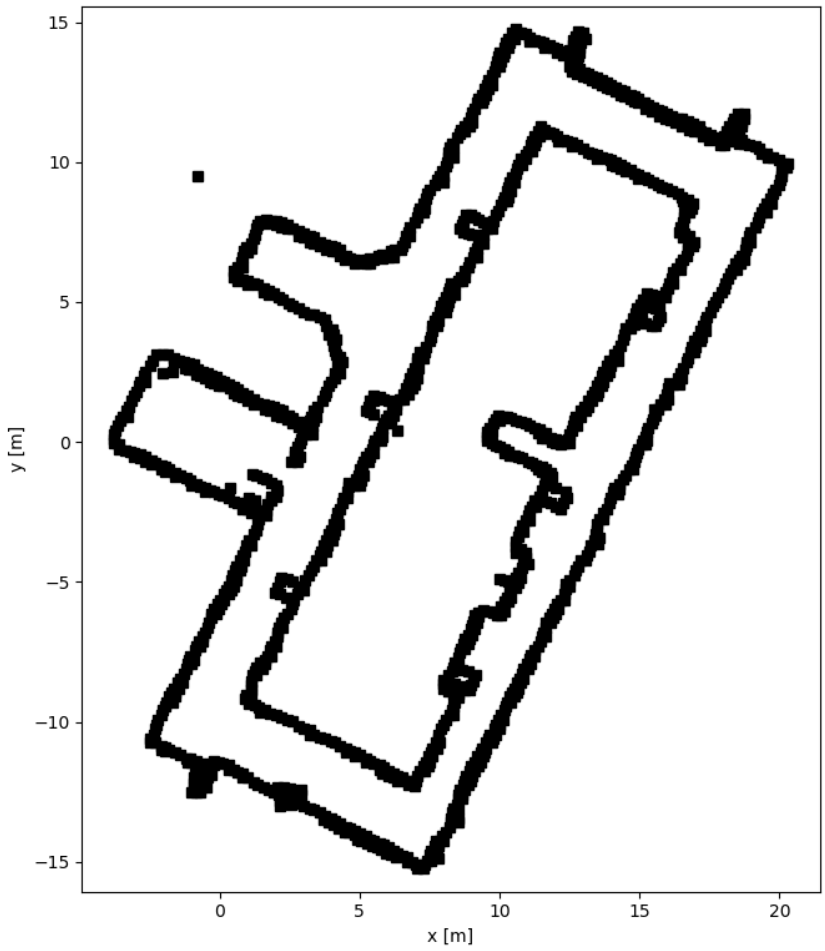

Abbildung 4.4: Karte eines Flurs in abgewandelter Darstellung. Jedes schwarze Viereck repräsentiert ein Objekt, welches in einem wählbaren Radius nicht passiert werden kann. Die Objekte symbolisieren die Wände der vom Rollstuhl erstellten Karte.

Die Aufgabe des Fahrers für diese Karte änderte sich dahingehend, dass die Wegpunkte $(9.3, 9.8, 1.17)$, $(17.2, 5.8, -1.97)$, $(8.6, -10.8, -1.97)$ und $(0.5, -6.5, 1.17)$ abgefahren werden mussten und der Startpunkt bei der Position $(0.0, 0.0, -0.4)$ lag.

Während der Fahrt wurde eine Analyse des Steuerverhaltens durchgeführt. Bei dieser Analyse wurde die Konstante $k_\delta = 1$ zur Distanzberechnung während der Analyse gewählt. Die Konstante $k_\phi = 1.2$ blieb unverändert.

Die Analyse der Distanz ist in Abbildung 4.5 und die dazugehörige gefahrene Strecke in Abbildung 4.6 zu sehen.

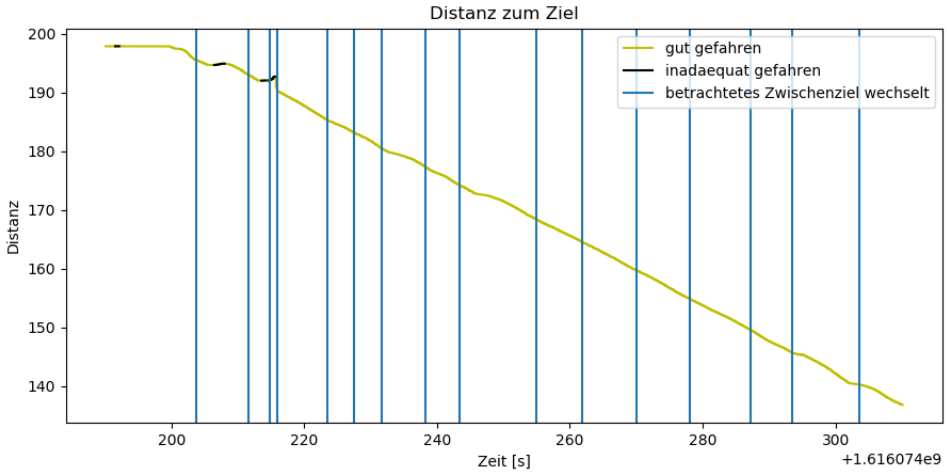

Abbildung 4.5: Die Veränderung der Distanz im Verlauf der Fahrt. Die Konstanten der Distanzfunktion 2.19 wurden hierbei mit $k_\phi = 1.2$ und $k_\delta = 1$ gewählt. Solange sich die Distanz nicht länger als 0.5 Sekunden erhöhte, wurde der gefahrene Streckenabschnitt als gut bewertet, andernfalls als inadäquat. Die vertikalen Linien markieren die Zeiten, an denen ein Zwischenziel erreicht wurde.

Inadäquat gefahrene Streckenabschnitte wurden in beiden Abbildungen schwarz markiert und folgen dabei der Definition aus Kapitel 4.1.

Abbildung 4.6 zeigt, dass der Fahrer gut gefahren ist. Dies spiegelt sich auch größtenteils in der Abbildung 4.5 durch einen konstanten Abfall der Distanz wider, jedoch nicht am Anfang der Fahrt. Die Markierungen bis kurz vor dem ersten Zwischenziel sind hierbei jedoch zu ignorieren, da der Rollstuhl in dieser Zeit stillstand, was auch an der konstanten Distanz in diesem Abschnitt zu erkennen ist. Nachdem der Fahrer losgefahren ist, sind jedoch zwei Streckenabschnitte inadäquat abgefahren worden.

In Abbildung 4.6 ist zu sehen, dass beim ersten von diesen inadäquat abgefahrenen Streckenabschnitten eine Linkskurve erwartet wurde, welche aufgrund der Wand vom Fahrer nicht so früh durchgeführt werden konnte. Zudem ist die Orientierung des ersten Zwischenziels in die Richtung einer Wand. Diese nicht intuitive Wahl des Zwischenziels

Abbildung 4.6: Der vom Fahrer gefahrene Weg beim Abfahren aller Wegpunkte. Auf schwarz markierten Streckenabschnitten wurden so gefahren, dass sich die Distanz zum Ziel erhöhte. Eine genauere Analyse ist in Abbildung 4.5 zu betrachten.

ist durch die Eigenarten der Regelung aus Kapitel 2.2 bedingt. Diese führte wahrscheinlich zu der Fehlinterpretation des Steuerverhaltens. Es ist nur zu vermuten, dass eine höhere Anzahl an Knoten im Graphen zu einem intuitiveren Weg geführt hätte. Wie dieser Streckenabschnitt theoretisch gefahren werden sollte, wurde in Abbildung 4.7 dargestellt.

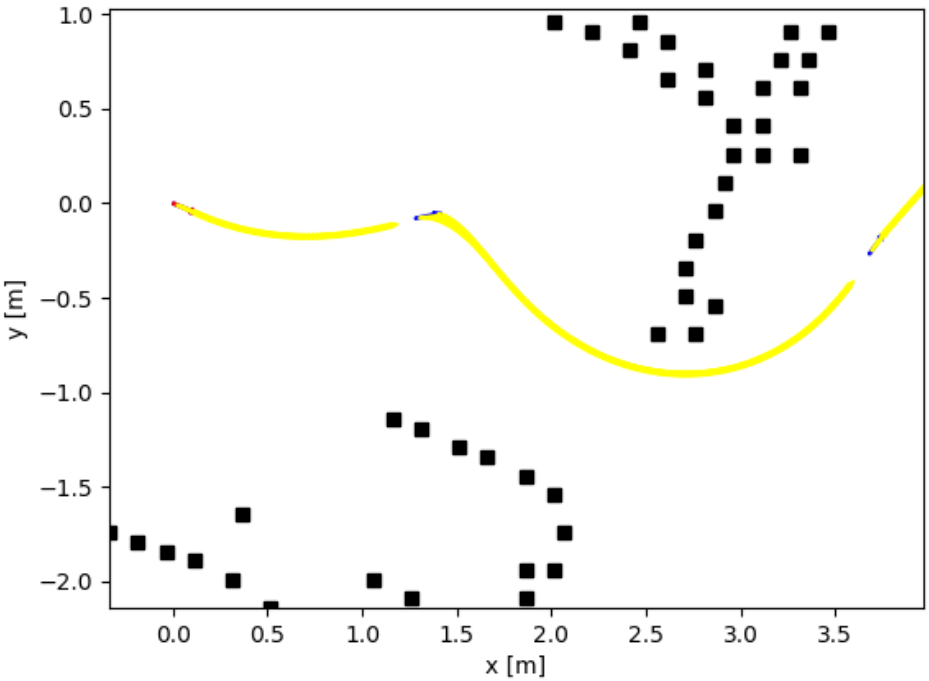

Abbildung 4.7: Analyse des ersten inadäquat gefahrenen Streckenabschnitts aus Abbildung 4.5. In Blau sind die Startposition und die ersten beiden Zwischenziele abgebildet. In Gelb sind die berechneten Wege zwischen den Zwischenzielen dargestellt. Die schwarzen Vierecke symbolisieren nicht passierbare Objekte der Karte. Es ist zu erkennen, dass der berechnete Weg nicht besonders intuitiv navigierbar ist.

Der zweite inadäquat abgefahrene Streckenabschnitt befindet sich kurz nach dem zweiten Zwischenziel. Hier ist jedoch erneut zu erkennen, dass theoretisch kein Fehler vom Fahrer begangen wurde. Markiert wurde dieser Streckenabschnitt dennoch, da zwei Zwischenziele kurz hintereinander platziert wurden. Der Fahrer passierte beide generierten Zwischenziele seitlich mit etwas Entfernung, wodurch die Distanz zu beiden Zwischenzielen fast zeitgleich anstieg.

Im weiteren Verlauf der Fahrt wurde kein weiteres Fehlverhalten festgestellt. Generell fällt die Distanz konstant ohne größere Schwankungen ab, was für ein sehr gutes Steuerverhalten spricht. Die Evaluation einer Fahrt auf dem Rollstuhl mit bewusst inadäquat gefahrenen Streckenabschnitten war aufgrund von fehlerhaften Daten und mangelnder Zeit nicht mehr möglich.

5 Ausblick

5.1 Fazit

Die Aufrechterhaltung der Selbstständigkeit einer immer älter werdenden Gesellschaft gewinnt zunehmend an Relevanz. Ein elektrischer Rollstuhl mit Assistenzfunktionen kann dies im Bereich der Mobilität ermöglichen. In dieser Arbeit wurde eine Methode zur Bewertung des menschlichen Steuerverhaltens entwickelt und deren Anwendung unter anderem auf einem elektrischen Rollstuhl getestet. Die Bewertung erfolgt dabei auf Grundlage der berechneten Distanz vom Fahrer zum Ziel unter nicht-holonomen Zwangsbeschränkungen. Die Distanz wird dabei auf einem zuvor generierten Graphen über den A* Algorithmus berechnet und wird während der Fahrt fortlaufend aktualisiert. Hierbei basiert der generierte Graph auf einer zufälligen Stichprobe an Knoten auf dem navigierbaren Raum und besitzt Kanten, welche unter nicht-holonomen Zwangsbeschränkungen abgefahren werden können. Anzumerken ist hierbei, dass auch über sehr lange Distanzen eine fortlaufend schnelle und genaue Distanzberechnung möglich ist.

Dies liegt besonders an folgenden Umständen:

- Der verwendete Graph wird unmittelbar nach dem Erfassen der Karte generiert, da dieser an die Karte gebunden ist und somit wieder verwendet werden kann. Eine Generierung des Graphens während der Fahrt ist somit nicht notwendig.
- Der A* Algorithmus zur Bestimmung des optimalen Weges muss nur zu Beginn der Fahrt einmalig aufgerufen werden und fällt somit durch seine Berechnungszeit nur kurzweilig ins Gewicht.
- Alle weiteren Distanzberechnungen werden auf der Grundlage des zuvor berechneten Weges durchgeführt und benötigen somit nur noch minimale Berechnungszeit.

5.2 Weitere Arbeiten

Aufgrund von fehlerhaften Daten und mangelnder Zeit war es nicht mehr möglich eine Fahrt auf dem Rollstuhl zu evaluieren, bei der bewusst inadäquat gefahren wurde. Um

das Bewertungsverfahren genauer zu evaluieren, wäre es somit empfehlenswert, solch eine Fahrt zu analysieren. Hierbei wären besonders sehr weit gefahrene Kurven und gefahrene Schlangenlinien als inadäquat gefahrene Streckenabschnitte interessant, da dies typische Fehlverhalten beim Steuern des Rollstuhl sind.

Es gibt einige weitere Punkte, in denen das Bewertungsverfahren verbessert werden sollte. So ist es derzeit nur möglich, eine Angabe darüber zu machen, wann inadäquat gefahren wurde, jedoch nicht, wieso. Eine Weiterentwicklung wäre somit dahingehend wünschenswert vom System eine Empfehlung an den Fahrer zu geben, wie ein besseres Steuerverhalten in der aktuellen Situation aussehen könnte. Weitergehend werden beim Berechnen des Weges vereinzelt Streckenabschnitte berechnet, welche nicht sonderlich intuitiv sind. So wäre eine Verbesserung der Berechnung des Weges durch zusätzliche Überprüfungen und Angleichungen des berechneten Weges in Betracht zu ziehen.

6 Anhänge

Die während der Arbeit generierten Implementierungen sind unter Folgendem Link zu finden: `https://github.com/Arne-Hegel/Bachelor_thesis_Arne_Hegel`

Literaturverzeichnis

[Choi and Choi, 2017] Choi, J.-H. and Choi, B.-J. (2017). Design of Autonomous Driving Platform for an Electric Wheelchair. Technical report, Daegu University, Gyeongsan Gyeongbuk.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107. Conference Name: IEEE Transactions on Systems Science and Cybernetics.

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894. Publisher: SAGE Publications Ltd STM.

[Karaman and Frazzoli, 2013] Karaman, S. and Frazzoli, E. (2013). Sampling-based optimal motion planning for non-holonomic dynamical systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 5041–5047. ISSN: 1050-4729.

[Karaman et al., 2011] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. (2011). Anytime Motion Planning using the RRT*. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. ISSN: 1050-4729.

[Kavraki et al., 1996] Kavraki, L. E., Svestka, P., Latombe, J., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580. Conference Name: IEEE Transactions on Robotics and Automation.

[Kevin M. Lynch and Frank C. Park, 2018] Kevin M. Lynch and Frank C. Park (2018). Graph Search - Modern Robotics.

[Konolige, 2000] Konolige, K. (2000). A gradient method for realtime robot control. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 639–646 vol.1.

[Korkmaz and Durdu, 2018] Korkmaz, M. and Durdu, A. (2018). Comparison of optimal path planning algorithms. In *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*, pages 255–258.

[LaValle, 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

[LaValle, Steven M, 1998] LaValle, Steven M (1998). Rapidly-exploring random trees: A new tool for path planning. *Ames, IA 50011 USA*, page 4.

[NumPy, oD] NumPy (o.D.). NumPy. https://numpy.org/.

[Park, 2016] Park, J. J. (2016). *Graceful Navigation for Mobile Robots in Dynamic and Uncertain Environments*. PhD thesis, University of Michigan.

[Park and Kuipers, 2015] Park, J. J. and Kuipers, B. (2015). Feedback motion planning via non-holonomic RRT* for mobile robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4035–4040, Hamburg, Germany. IEEE.

[Poncela et al., 2009] Poncela, A., Urdiales, C., Perez, E. J., and Sandoval, F. (2009). A New Efficiency-Weighted Strategy for Continuous Human/Robot Cooperation in Navigation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 39(3):486–500. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans.

[Webb and Berg, 2012] Webb, D. J. and Berg, J. v. d. (2012). Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints. *arXiv:1205.5088 [cs]*. arXiv: 1205.5088.