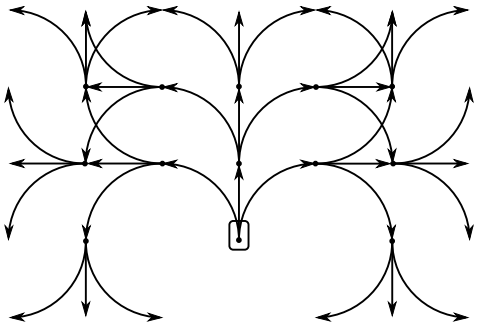
CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Masterabeit

Bahnplanungsframework für ein autonomes Fahrzeug



vorgelegt von

Jürgen Boger

Gutachter:

Prof. Dr. Martin Georg Fränzle
Dipl. Inform. Sönke Eilers

27. Februar 2013

ZUSAMMENFASSUNG

Diese Masterarbeit befasst sich mit dem Thema Bahnplanung für ein autonomes Fahrzeug mit Achsschenkellenkung. Aufgrund der Vielzahl verschiedener Algorithmen, die dieses Problem angehen, wurde ein Analysewerkzeug entwickelt, das rapid prototyping erlaubt. Mit einfachen Mitteln lassen sich mehrere Algorithmen parametrisieren und hintereinander ausführen. Die Statistikauswertung erlaubt es Algorithmen untereinander vergleichen zu können und deckt mit einer großen Zahl unterschiedlichster Zeitmessungen Lücken in Algorithmen auf, wodurch Verbesserungsansätze entstehen können.

INHALT

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Aktueller Stand . . . 1
- 1.3 Ziele der Arbeit . . . 2
- 1.4 Verfügbares System . . . 3
- 1.5 Aufbau der Arbeit . . . 4

2 Grundlagen **5**
- 2.1 Ackermann Modell . . . 5
- 2.2 Objekterkennung . . . 8
- 2.3 Kollisionserkennung . . . 11
  - 2.3.1 D-Hülle um Polygone . . . 11
  - 2.3.2 Schritt 1: Bounding Sphere Schnittest . . . 14
  - 2.3.3 Schritt 2: Schnittest Gerade mit Polygon . . . 17
  - 2.3.4 Schritt 3: Schnittest Polygon mit Polygon . . . 18
- 2.4 Regelung . . . 19
  - 2.4.1 Lenkwinkelregelung . . . 19
  - 2.4.2 Geschwindigkeitsregelung . . . 21
  - 2.4.3 Look Ahead . . . 23

3 Entwickeltes Framework **27**
- 3.1 Gaphische Programmierung . . . 28
- 3.2 Vehicle Parameters . . . 29
- 3.3 Statistik . . . 30

4 Motion Primitives **33**
- 4.1 Gerade . . . 33
- 4.2 Kreisbogen . . . 34
- 4.3 Klothoide . . . 35
- 4.4 Continuous Steering . . . 39
- 4.5 Verbindungen . . . 41
- 4.6 Ergebnisse und Evaluation . . . 42

5 Sampling **43**
- 5.1 Control Space . . . 43
  - 5.1.1 Arc-Based search space . . . 43
  - 5.1.2 Adaptive search space . . . 44
- 5.2 State Space . . . 46

5.2.1 State Lattice . . . 47
5.2.2 Online Sampling . . . 49
5.3 Ergebnisse und Evaluation . . . 50

6 Suchalgorithmen 53
6.1 A* . . . 54
6.2 Heuristische Funktion . . . 58
6.2.1 Weglassen der Heuristik . . . 58
6.2.2 Euklidischer Abstand . . . 59
6.2.3 Gewichteter Abstand . . . 60
6.2.4 Gewichtete Orientierung . . . 63
6.2.5 Kreisbogenabstand . . . 65
6.3 Datenstrukturen . . . 68
6.3.1 Skew Heap . . . 69
6.3.2 Sortierte Liste . . . 72
6.3.3 Octree . . . 73
6.4 Hierarchische Suche . . . 76
6.4.1 Gridsuche . . . 76
6.4.2 Theta* . . . 77
6.5 Ergebnisse und Evaluation . . . 79

7 Reeds Shepp 83
7.1 Reeds Shepp . . . 84
7.1.1 Konstruktion: Äußere Kreistangenten . . . 85
7.1.2 Konstruktion: Innere Kreistangenten . . . 86
7.2 Continuous Steering . . . 87
7.2.1 Konstruktion: μ-Tangenten . . . 88

8 Fazit und Ausblick 91

Glossar I

Abkürzungen III

Abbildungen V

Tabellenverzeichnis IX

Literatur XI

1 EINLEITUNG

Zunächst wird das Thema motiviert und der Aktuelle Stand aufgezeigt. Danach wird die Problemstellung dieser Arbeit erläutert und das verwendete System. Letztlich wird eine kurze Übersicht der Kapitel dieser Arbeit genannt.

1.1 MOTIVATION

Die Bahnplanung ist ein wichtiges Thema in der Robotik. Es gibt zahlreiche Algorithmen die dieses Problem angehen. Diese haben auch meist viele verschiedene Parameter, die den Algorithmus auf vielerlei Weisen modifizieren können. Aufgrund vieler verschiedener Algorithmen ist die Entscheidung für einen nicht einfach, ohne diesen vorher zu testen. Das Bahnplanungsframework soll dieses Problem vereinfachen, da durch graphische Programmierung rapid prototyping betrieben werden kann. Die Simulation des Frameworks erlaubt eine visuelle Betrachtung der Arbeitsweise verschiedener Algorithmen. In dieser Simulation können statische Szenarien editiert werden, indem Hindernisse hinzugefügt werden und somit eine Bahn um diese geplant werden muss. Die Statistikauswertung der Algorithmen bietet einen Blick auf die Arbeitsweise, zeigt durch Zeitmessungen verschiedener Zugriffszeiten Schwächen auf und somit Verbesserungsansätze für diese Algorithmen.

Die Schwierigkeit der Bahnplanung entsteht bei den kinematischen Einschränkungen von Fahrzeugen mit Lenkachse (Furtuna et al. 2008 [8]). Das Fahrzeug kann nur bis zu einem maximalen Lenkwinkel einschlagen, wodurch ein minimaler Wendekreis resultiert und beim Planen in engen Gassen zu Problemen führen kann. Auch das Erreichen des maximalen Lenkeinschlags ist durch den mechanischen Servomotor, der das Lenkrad bewegt, physikalisch in der Geschwindigkeit begrenzt, so dass man aus gerader Fahrt nicht sofort den maximalen Lenkwinkel einschlagen kann. Durch die letztere Einschränkung, kann man folgern, dass Geraden und Kreisbögen allein nicht ausreichen und man mathematisch Komplexere Funktionen benutzen muss, um das Problem zu lösen.

1.2 AKTUELLER STAND

Das Thema Bahnplanung gliedert sich in mehrere kleine Teilbereiche, die im Verlauf dieser Arbeit aufgezeigt werden und getrennt von einander betrachtet werden können, weil in jedem dieser Teilbereiche eigene Algorithmen ausgetauscht werden können.

In erster Linie gibt es den Bereich der Motion Primitiven (siehe Kapitel 4), die für die Berechnung der elementaren Bewegungsprimitiven zuständig sind, die von einem Fahrzeug abfahrbar sind. Die Entwicklung in diesem Bereich ging von Berechnung über Geraden (Tsumura et al. 1986 [40]), Kreisbögen (Komoriya, Tachi & Tanie 1986 [20]), Klothoiden (Tounsi & Le Corre 1996 [39]) bis hin zu kubischen Spiralen (Kanayama & Hartman 1989 [16], Kelly & Nagy [17]). Schon vorher wurde mit

aneinader gereihten elementaren Primitiven die minimale Verbindung zwischen einer Startkonfiguration und einer Endkonfiguration für ein vorwärts fahrendes Fahrzeug von Dubins aufgezeigt (Dubins 1957 [4]), diese wurde später von Reeds und Shepp für beide Fahrtrichtungen generalisiert (Reeds & Shepp [32]).

Des Weiteren gibt es den Bereich Sampling (siehe Kapitel 5), der dafür zuständig ist mehrere Motion Primitiven zu berechnen und diese zu einem Template zusammenzufügen, welches nachträglich für die Suche gebraucht wird (Pivtoraiko & Kelly 2011 [30], Pivtoraiko 2012 [31]). Das Template kann auch gekrümmt werden, um sich so an die Umgebung anzupassen (McNaughton et al. 2011 [27])

Letztlich sind die heuristischen Suchen (siehe Kapitel 6) dafür zuständig aufbauend aus mehreren Motion Primitiven, die nacheinander abgefahren werden, einen Pfad von einer Startkonfiguration bis zu der Endkonfiguration zu suchen. Die Schar an verschiedenen heuristischen Suchen, kann in mehrere Kategorien gegliedert werden. Zunächst gibt es die kontinuierlichen heuristischen Suchen, die nur ein einziges mal ausgeführt werden, um ein Weg von Start zum Ziel zu finden. Die Entwicklung startete bei Dijkstra (Dijkstra 1959 [2]) bis hin zu A* (Hart, Nilsson & Raphael 1968 [9]). Wenn jedoch nicht die volle Information über die Umgebung vorliegt und sich somit im Verlaufe einer Fahrt neue Erkenntnisse darüber ergeben, so dass umgeplant werden muss, gibt es auch hier dynamische Algorithmen, die eine Bahnkorrektur vornehmen können ohne das alte Ergebnis zu verwerfen und ständig neu zu planen. Zunächst einmal gibt es den D* Algorithmus (Stentz 1995 [37]), der zu D* Lite optimiert wurde (Koenig & Likhachev 2002 [19]). Manchmal muss in dynamischen Umgebungen schnell reagiert werden, weshalb es Algorithmen gibt, die eine Zeitspanne vorgegeben bekommen, einen schnellen Weg planen und in verbleibender Zeit versuchen diesen zu optimieren. Die Entwicklung begann bei Anytime Algorithmen (Horvitz 1987 [13], Dean & Boddy 1988 [1], Zilberstein & Russel 1995 [41]) bis hin zu ARA* (Likhachev, Gordon & Thrun 2003 [26]). Auch dynamische Anytime Algorithmen sind möglich (Likhachev et al. 2005 [25]). Die meisten dieser Algorithmen wurden im Paper (Ferguson, Likhachev & Stentz 2005 [6]) miteinander verglichen.

Ein dynamisches Umplanen muss nicht immer eine neue heuristische Suche sein, sondern kann bei neuen Erkenntnissen von Hindernissen die bereits bekannte Bahn krümmen (Lamiraux, Bonnafous & Lefebvre 2004 [23], Rufli & Siegwart 2010 [33])

1.3 Ziele der Arbeit

Es soll ein graphisches Bahnplanungsframework entwickelt werden, mit dem man verschiedene Algorithmen der Bahnplanung ausprobieren kann. Diese sollen per Drag&Drop miteinander verbunden werden können. Die Parameter sollen dabei einstellbar sein, um die Auswirkungen verschiedener Parameter auf die Algorithmen betrachten zu können.

Des Weiteren ist die Arbeit in vier Teilbereiche zu gliedern, die die einzelne Aspekte der Bahnplanung beleuchten. Zu diesen gehören in erster Linie als grundlegend die Motion Primitiven (engl. Bahnsegment) und ihre Verknüpfung untereinander. Die verschiedenen Samplingverfahren, Sampling im Control Space und Sampling im State Space. Verschiedene Suchalgorithmen und ihre Parametriesierbarkeit durch Heuristiken. Als auch der Reeds Shepp - Algorithmus (Reeds & Shepp 1990 [32]), mit dem Posen angefahren werden können und die Erweiterung durch die physikalische Einschränkung der Lenkgeschwindigkeit zu Continuous Curvature (Fraichard & Scheuer 2004 [7]). Alle Kapitel sollen einzeln untersucht und evaluiert werden.

1.4 Verfügbares System

Zum Testen der Algorithmen unter realen Bedingungen stehen zwei elektrische Fahrzeuge zur Verfügung. Dazu gehört der Scooter von Götting KG, der von OFFIS in dem Projekt SaLsA benutzt wird, oder das elektrische „EcoCraft“-Fahrzeug, welches aber nur auf dem Firmengelände von Götting KG zur Verfügung steht (Abb. 1.1).

Der Scooter ist ein von Götting KG umgebauter Elektrorollstuhl. Dieser ist mit einem Laserscanner ausgestattet, mit dem er die Umgebung wahrnimmt (siehe Kapitel 2.2). Des Weiteren hat er ein Bahnführungsrechner von Götting KG, der über Sensoren den eingeschlagenen Lenkwinkel und die Rotation der Räder misst, wodurch er über Odometrie die Pose des Scooters bestimmen kann. Weiterhin kann der Bahnführungsrechner die Aktoren des Scooters beeinflussen, dazu gehört Lenkwinkel einstellen und Geschwindigkeit regeln.

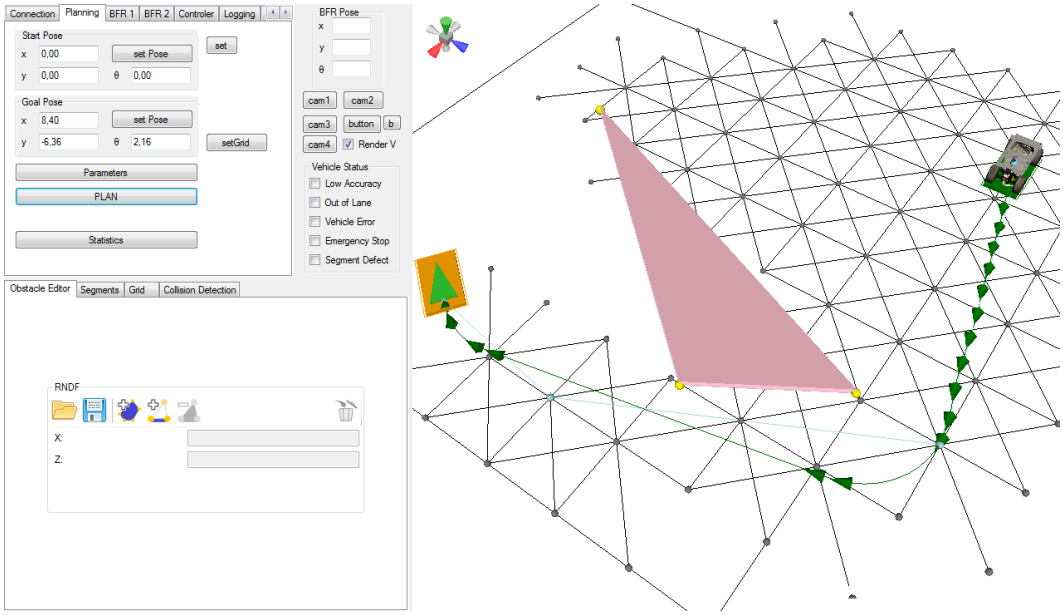
Zu der Ausstattung des EcoCraft, gehören alle der eben genannten Sensoren und Aktoren wie bei dem Scooter. Die Lokalisierung läuft über dGPS, wobei auf Odometrie zurückgegriffen werden kann, wenn kein dGPS Signal zur Verfügung steht. Das EcoCraft hat weitere Sensorik wie z. B. einen RGBD-Sensor, da aber dieses Fahrzeug nur auf dem Testgelände von Götting KG in Lehrte erreichbar ist, wurde auf die zusätzliche Sensorik verzichtet.

(a) Elektrisches Testfahrzeug „EcoCraft“ von Götting KG.

(b) Umgebautes Krankenfahrzeug, das OFFIS als Testfahrzeug dient.

(Abb. 1.1): *Testfahrzeuge.*

Zur Visualisierung und Simulation der Algorithmen wurde das Programm Bahnplanung verwendet, welches im Rahmen eines OFFIS Projekts SaLsA entwickelt wurde. Das Programm wurde mit C# mithilfe von DirectX programmiert. Dieses Programm ist die Grundlage zur Entwicklung der Bahnplanungframeworks (Abb. 1.2).



(Abb. 1.2): *Screenshot der Bahnplanung.*

1.5 Aufbau der Arbeit

Im nächsten Kapitel werden zunächst die Grundlagen für diese Arbeit erklärt. Im dritten Kapitel wird das entwickelte Framework vorgestellt. Die nächsten vier Kapitel bilden den wissenschaftlichen Teil der Arbeit, wobei in jedem Kapitel zunächst das Thema vorgestellt wird und letztlich mit einer Evaluation und Ergebnissen abschließt. Diese Themen sind zunächst die Motion Primitiven, wo darauf eingegangen wird, welche Primitiven sich für die Bahnplanung eignen und welche Probleme bei der Verbindung derer untereinander entstehen können. Im fünften Kapitel werden verschiedene Methoden gezeigt Motion Primitiven zu erstellen und auf deren Vor- und Nachteile eingegangen. Im sechsten Kapitel werden verschiedene Suchalgorithmen und deren Auswirkungen durch Parametrisierung aufgezeigt. Um das ganze abzuschließen und eine Pose anfahren zu können, wird noch im siebten Kapitel der Reeds-Shepp-Algorithmus und dessen Erweiterung erklärt. Schließlich folgt im achten Kapitel das Fazit und weiterer Ausblick zu der Arbeit.

2

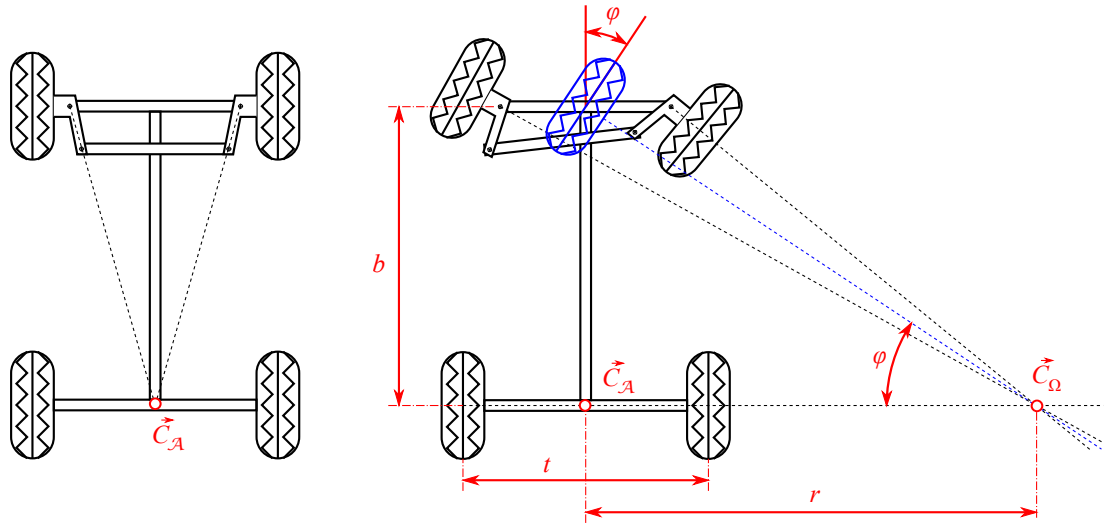
GRUNDLAGEN

Der Teil „Ackermann Modell“ beschreibt den mathematischen Hintergrund eines Fahrzeugs und führt in die Nomenklatur von fahrdynamischen Systemen ein. Der Teil „Objekterkennung“ beschäftigt sich damit, auf welche Art anhand der Sensorik eine Umgebungskarte mit Hindernissen entsteht. Im Teil „Kollisionserkennung“ wird darauf eingegangen, wie Kollisionen zwischen Objekten und Fahrzeug während der Bahnplanung berechnet werden können. Der letzte Teil „Regelung“ zeigt den Reglerentwurf und getätigte Verbesserung an diesem.

2.1 ACKERMANN MODELL

Die Achsschenkellenkung wurde von dem Münchener Wagenbauer Georg Lankensperger im Jahre 1816 erfunden. Dabei wies er zeichnerisch darauf hin, dass das kurveninnere Rad einen größeren Lenkeinschlag als das kurvenäußere Rad haben müsse, so dass sich die Verlängerungen der Radachsen in einem Punkt (dem Kreismittelpunkt) treffen müssen. Da es zu dieser Zeit noch kein nationales Patentrecht gab, ließ Lankensperger dieses in England von Rudolph Ackermann patentierten, weshalb es heute noch unter dem Namen Ackermann- oder A-Steering geführt wird (Eckermann 1998 [5]).

Die Annahme dieses Modells war, wenn sich das Fahrzeug auf einer Kreisbahn bewegt, so muss das kurveninnere Rad einen größeren Lenkwinkel einschlagen, da es sich auf einer kleineren Kreisbahn bewegt. Im Idealfall schneiden sich die Normalen der Räder in einem Punkt, der Mittelpunkt des Kreises, um den sich das Fahrzeug bewegen soll. Verwendet man Achsschenkel, deren Verlängerung sich an der Hinterachse treffen, so entsteht ein Trapez, dessen Lenkgeometrie sich an diese Vorgaben hält (Abb. 2.1). Die Vorteile dieser Lenkung, im Gegensatz zu der davor üblichen Drehschemellenkung, sind in erster Linie ein geringer Abrieb der Räder in Kurven, da jedes Rad den richtigen Kurvenradius einschlägt. Aufgrund der sich kaum ändernden Auflagefläche, weil sich nur die Räder drehen und nicht die gesamte Achse, entsteht eine viel größere Stabilität, wodurch die Kippgefahr von Fahrzeugen minimiert wird. Die Tatsache, dass sich die gesamte Achse bei der Drehschemellenkung bewegen muss, verbraucht diese im vorderen Bereich sehr viel Platz, der bei der Achsschenkellenkung für anderen Dinge, wie Motor gebraucht werden kann. Der Nachteil ist der größere Wendekreis, der durch den stark begrenzten Lenkeinschlag der Achsschenkellenkung entsteht. Die Drehschemellenkung bleibt bei 90° Lenkeinschlag mit dem Mittelpunkt der Hinterachse auf einem Punkt, wohingegen dieser bei der Achsschenkellenkung einen Kreis beschreibt, der als minimaler Wendekreis bezeichnet wird (Eckermann 1998 [5]).



(a) Die Achsschenkel, sind so ausgerichtet, dass deren Verlängerungen sich an der hinteren Achse treffen.

(b) Das Fahrzeug mit Lenkwinkel φ. Jedes Rad hat dabei unterschiedliche Einschläge, deren Normalen sich im Punkt $\vec{C_\Omega}$ treffen, um den sich das Fahrzeug bewegt.

(Abb. 2.1): *Modell eines Fahrzeugs mit Achsschenkellenkung.*

Unter der Annahme, dass $\vec{C_\mathcal{A}}$ die Pose des Fahrzeugs ist, kann der Kurvenradius direkt berechnet werden. Wenn sich alle Normalen in einem Punkt $\vec{C_\Omega}$ treffen, so hat das gedachte innere Rad (blau in Abb. 2.1 (b)) den Mittelwert als Lenkeinschlag der beiden Vorderräder. Im Folgenden wird φ als Lenkeinschlag des gedachten mittleren Rades angenommen. Dieser Winkel wiederholt sich zwischen der Normalen des gedachten Rades und der Geraden zwischen $\vec{C_\mathcal{A}}$ und $\vec{C_\Omega}$. Es entsteht zusammen mit der Mittelachse ein rechtwinkliges Dreieck, dessen Katheten der Radstand (engl. wheelbase) b und der gesuchte Kurvenradius r sind (Abb. 2.1 (b)). Dieser berechnet sich dann wie folgt:

$$r = \frac{b}{\tan(\varphi)} \tag{2.1}$$

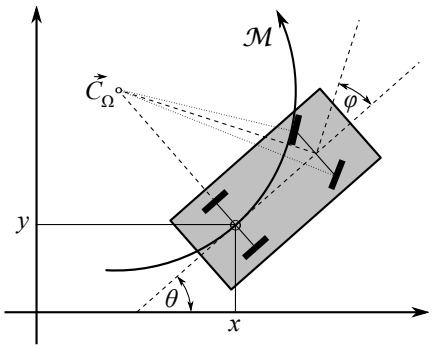
Aus dieser folgt unmittelbar der minimale Wendekreis $r_{\min}$, der bei maximalem Lenkeinschlag $\varphi_{\max}$ entsteht:

$$r_{\min} = \frac{b}{\tan(\varphi_{\max})} \tag{2.2}$$

Der Zustand $\mathcal{A}$ des Fahrzeugs kann nun als 4-Tupel (Quadrupel) aufgefasst werden, zu dem die Position x, y die Orientierung θ und der aktuelle Lenkeinschlag φ gehören. Ohne den Lenkeinschlag ist der Zustand die aktuelle Pose des Fahrzeugs. Wenn man sich bei der Berechnung der Bahn auf $\mathbb{R}^2$ beschränkt, so kann man die Pose kompakt als eine affine Matrix $M_\mathcal{A} \in \mathbb{R}^{3\times3}$ ausdrücken, die für spätere Transformationen gebraucht wird.

$$M_\mathcal{A} = \begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

Die Orientierung θ des Fahrzeugs ist der Winkel zwischen X-Achsen-Richtungsvektor und dem Richtungsvektor des Fahrzeugs gegen den Uhrzeigersinn. Der Lenkwinkel φ ist der Mittelwert beider Räder und ist auf dem Intervall $[-\varphi_{\max}; \varphi_{\max}]$ definiert, wobei $\varphi_{\max}$ der maximal mögliche Mittelwert der Einschläge der Räder ist (Abb. 2.2). Weitere Eigenschaften des Fahrzeugs sind seine Geschwindigkeit v und seine Lenkgeschwindigkeit v_{steer}.



(Abb. 2.2): *Zustandsquadrupel (x, y, θ, φ) des Ackermannmodells und die Kreisbahn $\mathcal{M}$, bei sich nicht änderndem Lenkwinkel φ.*

Dieses System lässt sich mithilfe von physikalischen Differenzialgleichungen beschreiben.

$$\mathcal{A} = \begin{bmatrix} x \\ y \\ \theta \\ \varphi \end{bmatrix} \in \begin{bmatrix} \mathbb{R} \\ \mathbb{R} \\ \mathbb{R}, 0 \leq \theta < 2\pi \\ \mathbb{R}, -\varphi_{\max} \leq \varphi \leq \varphi_{\max} \end{bmatrix}$$

$$\dot{\mathcal{A}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \tan(\varphi)/b \\ 0 \end{bmatrix} \cdot v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot v_{\text{steer}}$$

2.2 OBJEKTERKENNUNG

Die Objekterkennung beschäftigt sich mit der Fragestellung aus Sensordaten ein Abbild der Realität so zu gestalten, dass darauf später Bahnplanung mit Kollisionserkennung operieren kann. In dieser Arbeit wird dafür ein 2D-Laserscanner verwendet (Abb. 2.3).

(Abb. 2.3): *2D-Laserscanner von SICK*

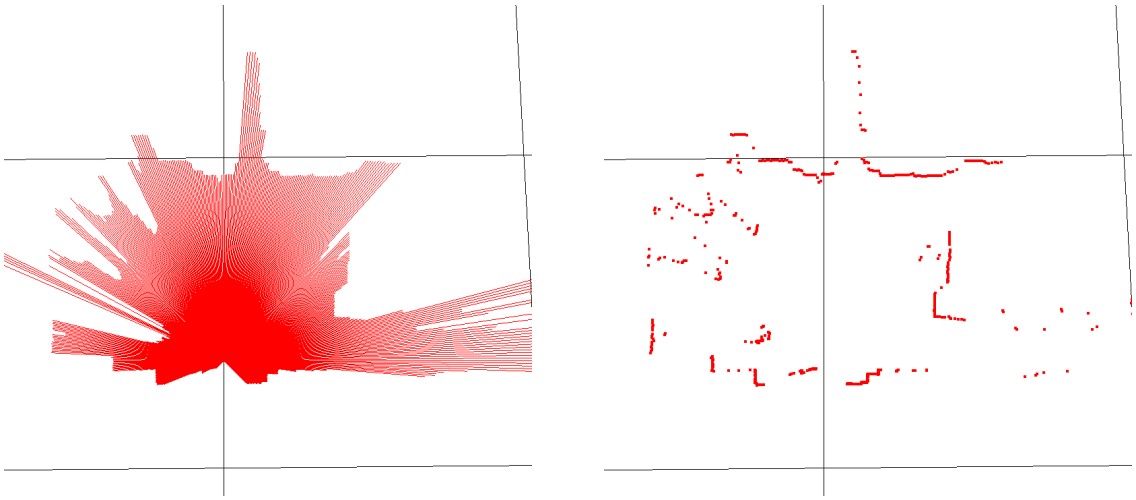
Die Arbeitsweise eines solchen 2D-Laserscanners ist ähnlich einem normalen Entfernungsmessgerätes mit einem Laser (Kirch 2007 [18]). Dabei wird ein Laseremmiter oben platziert und senkrecht nach unten auf einen Spiegel projiziert, der im Winkel von 45° zum Boden ausgerichtet ist und um die vertikale Achse rotiert wird (Abb. 2.4 (a)). Die Rotationsfrequenz des Spiegels ist dabei einstellbar, entweder 25 Hz oder 50 Hz. Die Entfernung wird dabei mit einer Frequenz von 36 kHz gemessen, wodurch eine Winkelauflösung, je nach Frequenz der Rotation des Spiegels, von 0,25° oder 0,5° entsteht. Aufgrund der Bauweise des Scanners sind die hinteren 90° des Blickfeldes versperrt, wodurch nur die vorderen 270° in Betracht gezogen werden (Abb. 2.4 (b)). Dabei entsteht ein Array von 540 oder 1080 Entfernungswerten, die in einem geeignetem Container über TCP versandt werden. Nebenbei enthält der Container auch Reflexionswerte und weitere Werte, die an dieser Stelle nicht von Relevanz sind und deshalb nicht betrachtet werden.

(a) Ansicht von der Seite mit Laseremmiter *L* und rotierendem Spiegel *R*.

(b) Ansicht von oben. Öffnungswinkel des Scanners 270°.

(Abb. 2.4): *Schematisches Modell der Laserscanner.*

Beim Messen auf eine Entfernung von 20m und einer Winkelauflösung von 0,5° ist der Abstand zwischen einzelnen Messwerten $\sqrt{(20m)^2 + (20m)^2 - 2 \cdot 20m \cdot 20m \cdot \cos(0,5°)} \approx 0,175m$. Das heisst, dass auf diese Entfernung mit 17,5cm Auflösung das Bein eines Menschen übersehen werden kann. Ein typisches Bild eines solchen Laserscanners kann man in Abbildung 2.5 sehen.



(a) Strahlenansicht **(b)** Punktansicht

(Abb. 2.5): *Typische Ansichten von Laserscanns.*

Die übermittelten Daten enthalten nur Entfernungswerte, so dass man diese zunächst transformieren muss, um auf solche Bilder (vgl. Abb.2.5) zu kommen, oder um die globale Postion von Objekten zu berechnen. Zunächst einmal kann man Sensoren auf zwei Arten verwenden, stationär und beweglich. Stationäre Sensoren stehen nur an einem Ort, so dass die Werte nur in diesen Ort transformiert werden müssen. Bewegliche Sensoren sind z. B. an einem Fahrzeug angebracht, so dass die Werte zusätzlich noch um die Pose des Fahrzeugs transformiert werden müssen.

Beschränkt man sich auf Laserscanner, so müssen zuerst alle Entfernungswerte in das lokale Koordinatensystem des Laserscanners transformiert werden, dabei wird die X-Achse als seine Ausrichtung betrachtet. Somit beginnt die Auswertung bei -135° und geht in 0,25° (oder 0,5° - kann in dem Datencontainer abgelesen werden) Schritten bis 135°. Für jeden Punkt $\vec{P_{\text{loc},i}}$ gilt dann:

$$\vec{P_{\text{loc},i}} = \begin{pmatrix} d_i \cdot \cos(-135° + i \cdot 0,25°) \\ -d_i \cdot sin(-135° + i \cdot 0,25°) \\ 1 \end{pmatrix}$$

Dabei ist d_i der Entfernungswert an der Stelle i und die 1 als Z-Koordinate macht aus dem Punkt $\vec{P_{\text{loc},i}}$ einen affinen Vektor, so dass dieser durch affine Matrizen transformiert werden kann. Die Pose des Laserscanners, kann genauso wie die Pose des Fahrzeugs (Eq. 2.3) durch eine affine Matrix M_L beschrieben werden. Somit gilt für alle Punkte $\vec{P_i}$ der stationären Laserscanner:

$$\vec{P_i} = M_L \cdot \vec{P_{\text{loc},i}}$$

Und für bewegliche Laserscanner gilt, wobei $M_{\mathcal{A}}$ die Pose im lokalen Koordinatensystem des Fahrzeugs ist und für das Fahrzeug unbeweglich angebracht ist:

$$\vec{P_i} = M_{\mathcal{A}} \cdot M_L \cdot \vec{P_{\mathrm{loc},i}}$$

Um die gewonnenen Punkte für die Kollisionserkennung nutzen zu können kann man auf verschiedenen Wegen verfahren, diese sollen an dieser Stelle kurz angeschnitten werden.

Occupancy Grid

Das so genannte Occupancy Grid (Thrun & Bücken 1996 [38]) diskretisiert den Raum und teilt diesen in Kacheln ein, wobei diese Kacheln entweder als belegt oder als unbelegt markiert werden können. Für einen gescannten Punkt werden die Kachelkoordinaten berechnet und in das Occupancy Grid eingefügt, wodurch diese Kachel als nicht befahrbar gilt. Für die Kollisionserkennung werden für alle belegten Kacheln des Occupancy Grids Polygone mit der Größe der Kachel erstellt. Abhängig von der Streuung der Objekte können so sehr viele kleine Polygone entstehen, wodurch die Kollisionserkennung unnötig verkompliziert wird. Nachteilhaft ist auch, dass durch einen Punkt direkt ein Quadrant von zuvor besimmter Kachelgröße als nicht befahrbar markiert wird. Diagonalen können dadurch sehr kachelig werden und große Bereiche werden unbefahrbar, obwohl keine Hindernisse vorliegen.

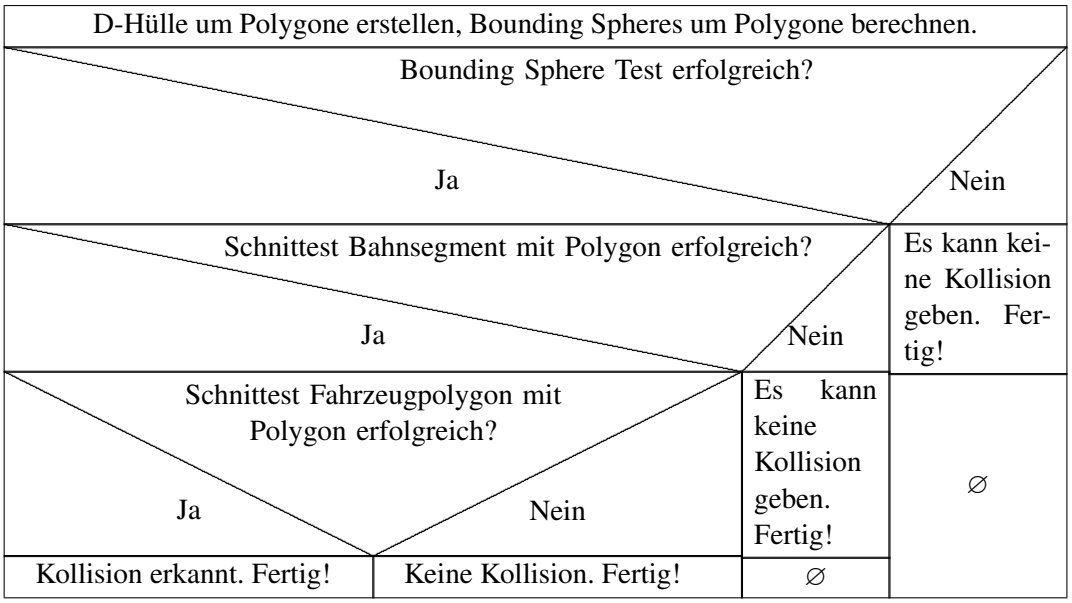
Douglas-Peucker-Algorithmus

Scannt man an einer Wand entlang, so entstehen mehrere nahezu auf einer Geraden liegende Punkte. „Nahezu“ aufgrund geringer Ungenauigkeiten beim Scannen, dabei weichen die Punkte teilweise um bis zu 5 cm ab. Im zeitlichen Verlauf der Daten erkennt man ein Flimmern der Punkte. Zwischen den Punkten kann man sich eine Wand denken und damit eine Kollisionserkennung machen. Aber aufgrund der hohen Anzahl der Punkte müssen viele Schnitttests (siehe Kapitel 2.3) gemacht werden. Weil die Wand aber gerade ist, wäre es da nicht einfacher, wenn nur vom ersten bis zum letzten Punkt ein Schnitttest gemacht werden würde? Als erster Schritt müssen zuerst alle Punkte geclustert werden, wodurch nachfolgende Punkte mit einem geringen Abstand als zusammengehörend erfasst werden, was indirekt heisst, das zwischen den einzelnen Punkten sich eine Wand befindet. Somit werden einzelne Cluster der Punktwolke zu Objekten definiert. Der Douglas-Peucker-Algorithmus (Douglas & Peucker 1973 [3]) lässt alle nicht notwendigen Punkte, die nicht weit genug von einer Geraden abweichen weg, wodurch die Anzahl der Punkte minimiert wird und dadurch auch die Anzahl der Schnitttest. Der Algorithmus arbeitet mit dem Prinzip Devide & Conquer, indem für die gegebene Menge an Punkten zunächst der Punkt mit dem höchsten Abstand b zu der Geraden gefunden wird, die den ersten und den letzten Punkt miteinander verbindet. Wenn dieser Abstand nicht den vorgegebenen Maximalabstand $d_{\max}$ überschreitet, so liegen **alle** Punkte nahezu auf einer Geraden mit einer maximalen Abweichung $b \leq d_{\max}$. Dann bilden der erste und der letzte Punkt die Gerade und alle anderen Punkte sind redundant und können weggelassen werden. Anderenfalls bildet der Punkt mit dem Abstand b einen neuen Kantenpunkt. Der gleiche Algorithmus wird nun für alle Punkte vom ersten Punkt bis zum Punkt b und vom Punkt b bis zum letzten Punkt noch einmal rekursiv durchgeführt, bis alle Kantenpunkte gefunden wurden.

2.3 KOLLISIONSERKENNUNG

Die Kollisionserkennung und -vermeidung spielt eine zentrale Rolle in der Bahnplanung. Ohne Kollisionserkennung würde die Bahnplanung zu einem einfachen Wegpunktregler degenerieren, wenn eine Position angefahren werden soll oder einfaches Reeds Shepp (siehe Kapitel 7) würde bereits das vorliegende Problem lösen, wenn eine Pose angefahren werden soll. An dieser Stelle werden nur die verwendeten Techniken in dieser Masterarbeit angesprochen.

Für die Kollisionserkennung müssen zunächst in einem offline Schritt alle Polygone vorbereitet werden, indem die D-Hülle um diese berechnet wird. In bis zu drei weiteren Schritten wird durch hierarchisch aufgebaute Schnitttests die Kollision berechnet (Abb. 2.6). Dabei nimmt die Komplexität der Schnittest mit jedem Schritt zu. Wo im ersten Schritt nur ein Abstandscheck der Boundingspheres gemacht werden muss $\mathcal{O}(1)$, muss im zweiten schon ein Schnitt einer Linie mit mehreren Linien ausgeführt werden $\mathcal{O}(n)$ und im letzten Schritt sogar der Schnitt mehrerer Linien mit mehreren Linien berechnet werden $\mathcal{O}(n \cdot m)$. Es gilt in jedem Schritt möglichst viele Kollisionen vorher auszuschliessen.



(Abb. 2.6): *Struktogramm des gesamten Kollisionstestprozesses*

2.3.1 D-HÜLLE UM POLYGONE

Um eine Kollision eines Polygons mit dem Fahrzeug zu bestätigen, muss ein Schnittest eines Rechtecks mit dem Polygon ausgeführt werden. Da dieser Schnittest komplex ist und auf der Bahn des Fahrzeugs mehrmals ausgeführt werden muss, kann dieser vereinfacht werden, indem das Fahrzeug als Punkt angenommen wird und alle Hindernisse um eine Distanz D vergrößert werden. Die neu entstandenen Polygone heißen D-Hüllen, weil sie die alten Polygone im Abstand D umhüllen.

Zur Berechnung des Abstandes eines Punktes von einer Linie, gibt es mehrere Wege, hier wird das Verfahren über das Kreuzprodukt nach (Kowalk 2010 [22]) verwendet.

Das Kreuzprodukt zweier Vektoren ist folgendermassen definiert:

$$\vec{V_0} \times \vec{V_1} = \begin{pmatrix} x_0 \\ y_0 \\ 0 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ x_0 \cdot y_1 - x_1 \cdot y_0 \end{pmatrix}$$

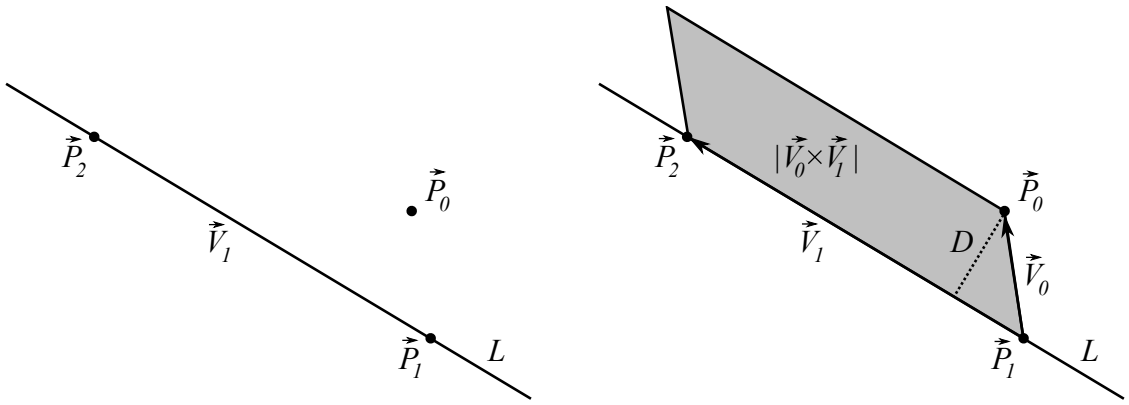
Im $\mathbb{R}^2$ kann man sich auf ein Skalar als Ergebnis einschränken:

$$[\vec{V_0} \times \vec{V_1}] = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = x_0 \cdot y_1 - x_1 \cdot y_0 \tag{2.4}$$

Dabei beschreibt [ ] die vorzeichenbehaftete Länge des Vektors, oder die Z-Komponente im $\mathbb{R}^2$. Die Länge (oder der Betrag) eines Vektors im $\mathbb{R}^2$ kann berechnet werden durch:

$$|\vec{V_i}| = \sqrt{x_i^2 + y_i^2} \tag{2.5}$$

Gegeben ist der Punkt $\vec{P_0}$ und die Gerade $L : \vec{P_1} + \lambda \cdot \vec{V_1}$. Das Kreuzprodukt vom Geradenrichtungsvektor $\vec{V_1}$ und Richtungsvektor $\vec{V_0} = \vec{P_0} - \vec{P_1}$, der sich aus der Differenz von Punkt und Ortsvektor der Geraden ergibt, hat eine äquivalente Länge zur Fläche des Parallelogramms, das von diesen beiden Vektoren eingespannt ist. Teilt man nun diese Größe durch die Länge von $\vec{V_1}$, so erhält man die Höhe dieses Parallelogramms, welche das gesuchte D ist (Abb. 2.7).



(a) Ausgangszustand **(b)** Parallelogramm des Kreuzprodukts

(Abb. 2.7): *Abstand Punkt Linie über das Kreuzprodukt.*

Beschränkt man sich bei der Berechnung auf $\mathbb{R}^2$, so ist das Ergebnis des Kreuzproduktes ein Vektor, der in Richtung der nicht vorhandenen Z-Achse zeigt und das Ergebnis auf ein Skalar eingeschränkt werden kann (Z-Wert des Vektors, die anderen beiden sind 0). Das Ergebnis kann sowohl positiv, als auch negativ sein, was den Abstand links oder rechts in Laufrichtung von V_1 definiert. Somit ergibt sich die Formel für den Abstand zu:

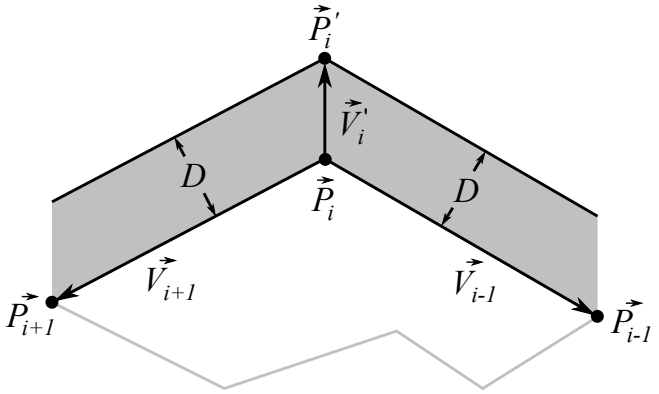
$$D = \frac{[\vec{V_0} \times \vec{V_1}]}{|\vec{V_1}|} \overset{(Eq.2.4)\&(Eq.2.5)}{=} \frac{x_0 \cdot y_1 - x_1 \cdot y_0}{\sqrt{x_1^2 + y_1^2}}$$

Sei das Polygon definiert als:

$$\mathcal{P} = \{\vec{P}_i | i \in \mathbb{Z}_n, \vec{P}_i \in \mathbb{R}^2\} \tag{2.6}$$

Dabei bezeichnet $\mathbb{Z}_n$ den Restklassenring *mod n* und n gibt die Anzahl der Punkte in diesem Polygon an. Um für einen Punkt $\vec{P}_i$ den Punkt $\vec{P}_i'$ der D-Hülle zu berechnen, braucht man zunächst die Richtungsvektoren zu den jeweiligen Nachbarn des Punktes (Abb. 2.8).

$$\begin{aligned} \vec{V_{i-1}} &= \vec{P_{i-1}} - \vec{P_i} \\ \vec{V_{i+1}} &= \vec{P_{i+1}} - \vec{P_i} \\ \vec{V_i'} &= \vec{P_i'} - \vec{P_i} \end{aligned}$$



(Abb. 2.8): *Punkt P_i' der D-Hülle eines Polygons*

Aus diesen lässt sich ein Gleichungssystem ableiten:

$$\begin{aligned} D = \frac{[\vec{V_{i-1}} \times \vec{V_i'}]}{|\vec{V_{i-1}}|} \quad &\Rightarrow \quad [\vec{V_{i-1}} \times \vec{V_i'}] = |\vec{V_{i-1}}| \cdot D \\ D = \frac{[\vec{V_i'} \times \vec{V_{i+1}}]}{|\vec{V_{i+1}}|} \quad &\Rightarrow \quad [\vec{V_i'} \times \vec{V_{i+1}}] = |\vec{V_{i+1}}| \cdot D \end{aligned}$$

In Paremeterdarstellung (nach Anwendung (Eq. 2.4)) hat man das folgende lineare Gleichungssystem:

$$\begin{aligned} x_{i-1} \cdot y_i - x_i \cdot y_{i-1} &= |\vec{V_{i-1}}| \cdot D \\ x_i \cdot y_{i+1} - x_{i+1} \cdot y_i &= |\vec{V_{i+1}}| \cdot D \end{aligned}$$

Die Matrixform des LGS, nach x_i und y_i:

$$\begin{pmatrix} -y_{i-1} & x_{i-1} & |\vec{V_{i-1}}| \cdot D \\ y_{i+1} & -x_{i+1} & |\vec{V_{i+1}}| \cdot D \end{pmatrix}$$

Nach der cramerschen Regel mithilfe der Determinanten, ergibt sich folgende Lösung für x_i und y_i

$$x_i = \frac{-|\vec{V_{i-1}}| \cdot D \cdot x_{i+1} - |\vec{V_{i+1}}| \cdot D \cdot x_{i-1}}{x_{i+1} \cdot y_{i-1} - x_{i-1} \cdot y_{i+1}} \overset{(Eq.2.4)}{=} D \cdot \frac{|\vec{V_{i-1}}| \cdot x_{i+1} + |\vec{V_{i+1}}| \cdot x_{i-1}}{[\vec{V_{i-1}} \times \vec{V_{i+1}}]}$$

$$y_i = \frac{-|\vec{V_{i+1}}| \cdot D \cdot y_{i-1} - |\vec{V_{i-1}}| \cdot D \cdot y_{i+1}}{x_{i+1} \cdot y_{i-1} - x_{i-1} \cdot y_{i+1}} \overset{(Eq.2.4)}{=} D \cdot \frac{|\vec{V_{i+1}}| \cdot y_{i-1} + |\vec{V_{i-1}}| \cdot y_{i+1}}{[\vec{V_{i-1}} \times \vec{V_{i+1}}]}$$

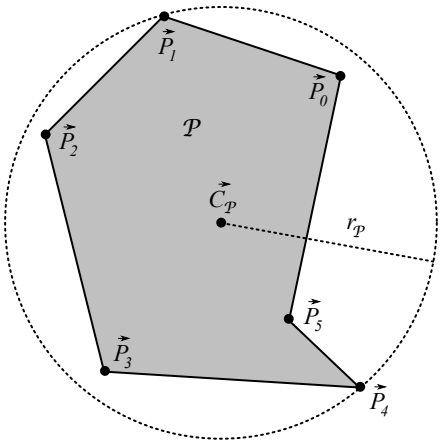
2.3.2 Schritt 1: Bounding Sphere Schnittest

Schnittests mit Polygonen sind in der Regel sehr langsam verglichen mit einem Abstandscheck der Bounding Spheres. Auf einer Bahn müssen diese Schnitttests auch mehrmals ausgeführt werden. Damit die Berechnungszeit etwas eingeschränkt wird, werden die Kollisionstests hierarchisch ausgeführt. Zuerst wird auf den Schnitt der Bounding Sphere der Bahn und der Bounding Sphere des Polygons getestet. Dieser Test kann lediglich eine Kollision ausschließen, diese aber nicht bestätigen (Abb. 2.9). In den meisten Fällen liegen die Bahnsegmente, gegen die getestet wird abseits von Polygonen und der Test gibt ein negatives Resultat, weshalb der Test bereits einen Großteil der Arbeit einspart.

Bounding Sphere Test erfolgreich?	
Ja	Nein
Weitere komplexe Berechnungen notwendig.	Es kann keine Kollision geben. Fertig!

(Abb. 2.9): *Struktogramm für einen Bounding Sphere Schnittest*

Sei das Polygon $\mathcal{P}$ definiert nach (Eq. 2.6). Dann berechnet sich die Bounding Sphere, indem man zunächst den Schwerpunkt (engl. centroid) $\vec{C_\mathcal{P}}$ für dieses Polygon berechnet und danach den Radius $r_\mathcal{P}$ findet, so dass alle Punkte des Polygons in dem entstandenen Kreis, der durch $\vec{C_\mathcal{P}}$ und $r_\mathcal{P}$ definiert wird, enthalten sind (Abb. 2.10).



(Abb. 2.10): *Bounding Sphere eines Polygons.*

Berechnung des Schwerpunktes:

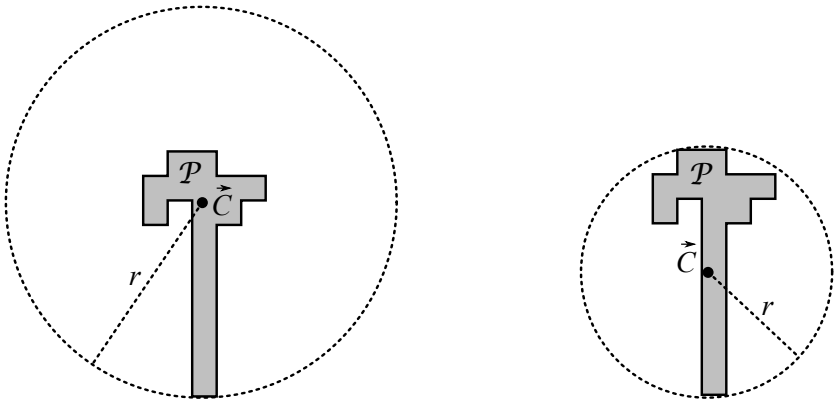
$$A = \frac{1}{2} \cdot \sum_{i=0}^{n-1} [\vec{P_i} \times \vec{P_{i+1}}]$$

$$\vec{C_{\mathcal{P}}} = \frac{1}{6 \cdot A} \cdot \sum_{i=0}^{n-1} (\vec{P_i} + \vec{P_{i+1}}) \cdot [\vec{P_i} \times \vec{P_{i+1}}]$$

Berechnung des Radius:

$$r_{\mathcal{P}} = \max_{i=0,\ldots,n-1} |(\vec{P_i} - \vec{C_{\mathcal{P}}})|$$

Da die Berechnung der Bounding Sphere für alle Polygone nur ein mal offline vor der Berechnung der Bahn erfolgt, wurde hier der Ansatz gewählt den Schwerpunkt genauer zu berechnen, als mit dem naiven Ansatz über die Summe aller Punkte, dividiert durch die Anzahl derer. Der Grund dafür ist, dass der Schwerpunkt beim naiven Ansatz in Richtung dichter Punktmengen verschoben wird, wodurch der Radius der Bounding Sphere unnötig anwächst und viele Kollisionen im Bereich nicht vorhandener Polygone verursacht und dadurch weitere zeitraubende Schnittest notwendig werden (Abb. 2.11).



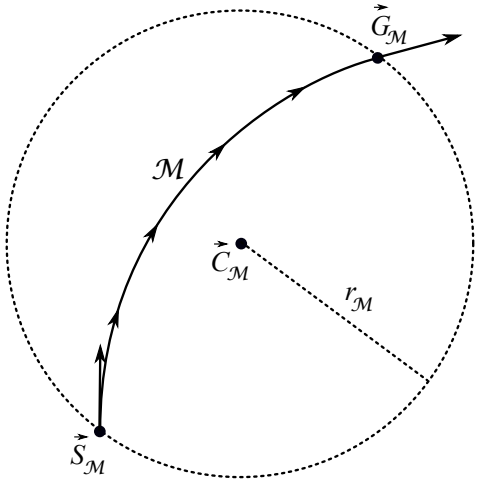
(a) Naiver Ansatz **(b)** Gewichteter Ansatz

(Abb. 2.11): *Vergleich der Bounding Spheres vom naiven Ansatz und dem gewichteten Ansatz.*

Sei eine Motion Primitive (engl. Bahnsegment) folgendermassen definiert:

$$\mathcal{M} = \{\vec{S_{\mathcal{M}}}, \vec{G_{\mathcal{M}}} | \vec{S_{\mathcal{M}}}, \vec{G_{\mathcal{M}}} \in \mathbb{R}^3\}$$

Dabei steht $\vec{S_{\mathcal{M}}}$ für die Startpose und $\vec{G_{\mathcal{M}}}$ für die Endpose (engl. goal). Die dritte Komponente des Tupels in der Start- und Endpose wird für die Berechnung der Bounding Sphere nicht benötigt. Für eine genaue Definition und Beschreibung der Motion Primitiven siehe Kapitel 4. Ähnlich wie bei einem Polygon soll auch hier ein Kreis gefunden werden, der möglichst klein ist und die gesamte Bahn einschliesst (Abb. 2.12).



(Abb. 2.12): *Bounding Spheres einer Motion Primitive.*

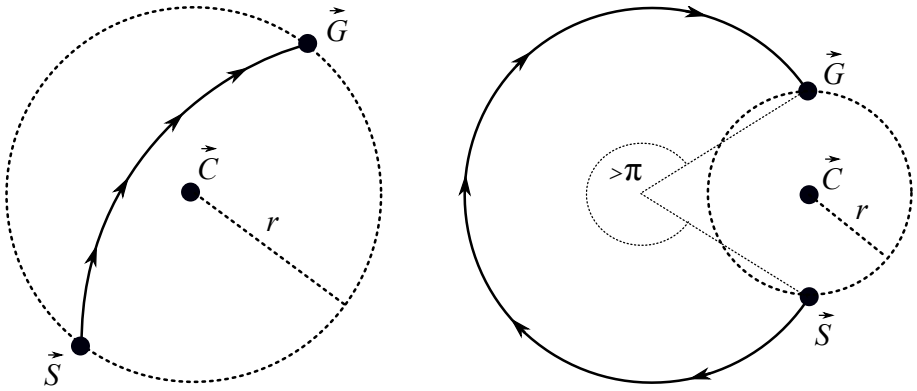
Für die Berechnung des Schwerpunktes $\vec{C_{\mathcal{M}}}$ gilt:

$$\vec{C_{\mathcal{M}}} = \frac{1}{2} \cdot (\vec{S_{\mathcal{M}}} + \vec{G_{\mathcal{M}}})$$

Berechnung des Radius:

$$r_{\mathcal{M}} = \frac{1}{2} \cdot |(\vec{S_{\mathcal{M}}} - \vec{G_{\mathcal{M}}})|$$

In den meisten Fällen ist diese Definition korrekt, da die Motion Primitiven sich zielgerichtet mit geringer Krümmung ausbreiten. Es gibt aber Ausnahmen, wenn man sich z. B. auf einer Kreisbahn bewegt, wo die einzelnen Motion Primitiven einen inneren Winkel größer als π haben. Dann würde diese Formel nicht das gesamte Segment in der Bounding Sphere erfassen und muss somit überschrieben werden (Abb. 2.13).



(Abb. 2.13): *Bei Bogensegmenten, deren innerer Winkel größer als π ist, muss die Bounding Sphere anders berechnet werden. Sonst entsteht eine Bounding Sphere, die nicht die gesamte Bahn umfasst (rechts).*

2.3.3 SCHRITT 2: SCHNITTEST GERADE MIT POLYGON

Unter der Annahme, dass Bahnsegmente niemals in Hindernissen vorkommen werden, da bereits bei einem Wechsel zwischen fahrbarem Bereich und Hindernis abgebrochen wird, so kann man den Schnittest auf einen Schnittest von Geradensegment mit jeder Kante (Geradensegment) des Polygons vereinfachen.

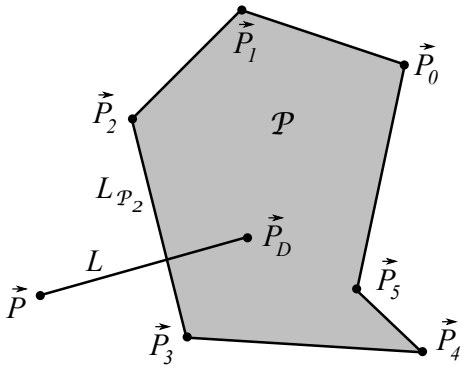
Sei das Polygon $\mathcal{P}$ definiert nach (Eq. 2.6). Dann gilt für das Geradensegment L der Bahn und das Geradensegment $L_{\mathcal{P}_i}$ des Polygons (Abb. 2.14):

$$\begin{aligned} L &: \vec{P} + \lambda \cdot (\vec{P_D} - \vec{P}) && | \, 0 \leq \lambda \leq 1 \\ L_{\mathcal{P}_i} &: \vec{P_i} + \mu \cdot (\vec{P_{i+1}} - \vec{P_i}) && | \, 0 \leq \mu \leq 1 \end{aligned}$$

Daraus folgt für λ und μ:

$$\begin{aligned} \lambda &= \frac{[(\vec{P_D} - \vec{P}) \times (\vec{P_i} - \vec{P})]}{[(\vec{P_{i+1}} - \vec{P_i}) \times (\vec{P_D} - \vec{P})]} \\ \mu &= \frac{[(\vec{P_{i+1}} - \vec{P_i}) \times (\vec{P_i} - \vec{P})]}{[(\vec{P_{i+1}} - \vec{P_i}) \times (\vec{P_D} - \vec{P})]} \end{aligned}$$

Da die Geradensegmente auf dem Intervall $0 \leq \lambda, \mu \leq 1$ definiert sind, kann man den Test nach Berechnung von λ mit negativem Ergebnis bereits abbrechen, wenn dieses die Einschränkung bereits nicht erfüllt. Ansonsten muss auch μ berechnet werden.



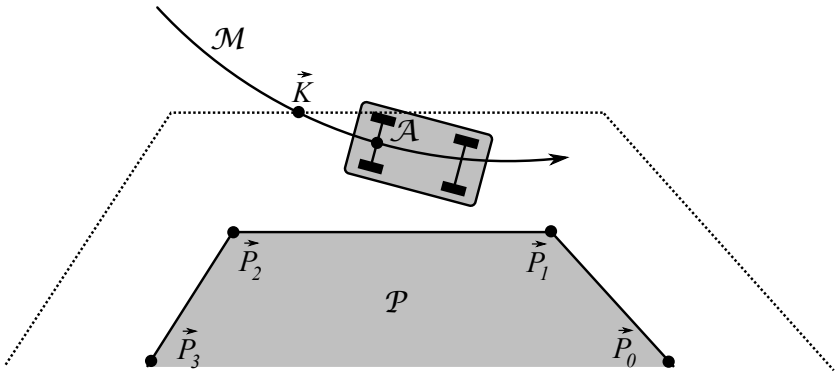
(Abb. 2.14): *Schnitttest Gerade mit Polygon. Vereinfacht auf Schnittest zweier Geradenesgmente L und $L_{\mathcal{P}_i}$.*

Sollte der Test mit einer der Kanten erfolgreich sein, so gibt es eine Kollision und andere Kanten müssen nicht mehr überprüft werden. Für ein negatives Ergebnis müssen Schnitttests mit allen Kanten durchgeführt werden.

Ein Bahnsegment wird in mehrere Geradensegmente aufgeteilt und dieser Schnittest mit jedem der Segmente ausgeführt. Sobald einer der Tests Erfolgreich ist, so findet eine Kollision mit der D-Hülle statt und es muss für dieses und alle folgenden Geradensegmente der Schnitttest mit dem Fahrzeug erfolgen (nächstes Kapitel).

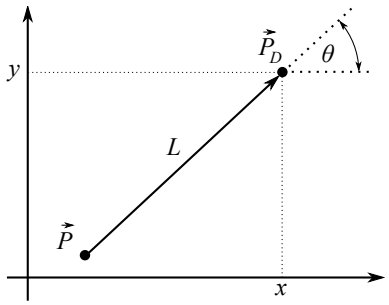
2.3.4 SCHRITT 3: SCHNITTEST POLYGON MIT POLYGON

Dieser Schnittest ist der Langsamste und der Letzte in der Berechnungshierarchie. Wird eine Kollision der Bahn mit dem Polygon festgestellt, so heisst dies, dass der Fahrzeugmittelpunkt die D-Hülle des Polygons geschnitten hat. Der Abstand für die D-Hülle wurde als kleinster Radius, der das gesamte Fahrzeug umfasst (Abstand Fahrzeugmittelpunkt zur vorderen Ecke des Fahrzeugs) gewählt. Wenn das Fahrzeug seitlich an diesem Hindernis vorbeifährt, so ist die halbe Wagenbreite viel geringer, als der gewählte Abstand der Hülle und es gibt doch keine Kollision. Dieser Test wird für die Endpunkte aller Geradensegmente der Bahn ausgeführt (Abb. 2.15).



(Abb. 2.15): *Bahnsegment $\mathcal{M}$ schneidet die D-Hülle (gestrichelt) im Punkt $\vec{K}$, jedoch berührt das Fahrzeug mit der Konfiguration $\mathcal{A}$ das Polygon $\mathcal{P}$ nicht.*

Für die Berechnung müssen alle vier Eckpunkte des Fahrzeugs, aus dem Fahrzeuglokalen System ins globale System (indem auch die Polygone definiert werden) transformiert werden. Dazu wird ähnlich zu (Eq. 2.3) die affine Positionsmatrix berechnet und alle Eckpunkte des Fahrzeugs damit (von links) multipliziert. Um die affine Positionsmatrix zu berechnet, nimmt man den Endpunkt des Geradenegments L und den von der X-Achse und der Geraden gegen den Uhrzeigersinn eingeschlossenen Winkel, als Orientierung (Abb. 2.16).



(Abb. 2.16): *Anschauliche Werte zur Herleitung der affinen Positionsmatrix.*

Schließlich wird jede Kante des Objektpolygons mit dem transformierten Fahrzeugpolygon geschnitten. Zur Berechnung wird der gleiche Algorithmus wie (Kapitel 2.3.3) benutzt. Auch hier gilt, wenn der erste Test Erfolgreich war, so müssen die nachfolgenden Tests nicht mehr ausgeführt werden und eine Kollision wurde festgestellt.

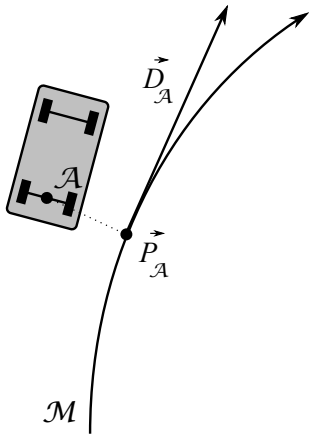
2.4 REGELUNG

Die Regelung des Fahrzeugs wurde für diese Arbeit als gegeben vorausgesetzt, diese wurde aber zum Testen des Fahrzeugs benutzt und soll an dieser Stelle vorgestellt werden. Der Regelungsentwurf und die anfängliche Lenkwinkelregelung stamm von dem Betreuer der Arbeit Sönke Eilers und wurde von mir nachträglich um die Geschwindigkeitsregelung und den Look Ahead erweitert. Diese drei Konzepte werden in den folgenden Kapiteln vorgestellt, ausserdem sollen diese ein Gefühl dafür vermitteln, was man zusätzlich bei der Berechnung der Bahn noch braucht, um diese später auch befahren zu können.

Sehr wichtig dabei für die Regelung ist die Lokalisierung des Fahrzeugs. Diese fand auf dem Eco-Craft mithilfe von DGPS statt, was es erlaubte die Position des Fahrzeugs Zentimetergenau auszumessen. Der Scooter lokalisierte sich mithilfe von Odometrie, mit der man nur die Transformation des Fahrzeugs anhand der getätigten Aktionen vom Startpunkt aus messen kann, um so auf die aktuelle Position zu schließen. Beide Verfahren erlauben es Posen ohne große Wertsprünge zu messen, die zu undefiniertem Verhalten der Regelung führen könnten.

2.4.1 LENKWINKELREGELUNG

Für die Regelung des Lenkwinkels, bekam die Motion Primitive die Funktion `ClosestPoint`, die als Parameter die aktuelle Position des Fahrzeugs hat und als Ausgabe den nähesten Punkt $\vec{P_{\mathcal{A}}}$ auf der Bahn zum übergebenen Punkt, die Tangente in dem Punkt (als normalisierter Richtungsvektor) $\vec{D_{\mathcal{A}}}$ und die Position als Interpolationsgewicht t im Intervall [0,1] errechnet (Abb. 2.17).



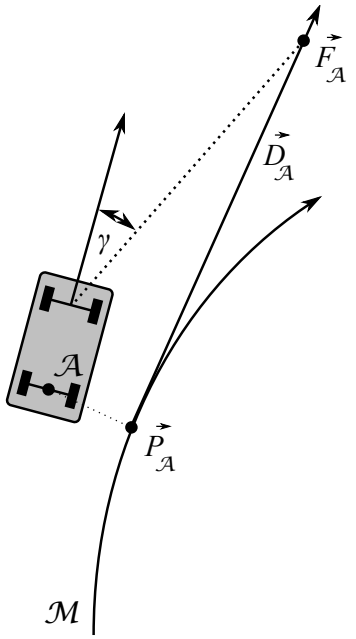
(Abb. 2.17): *Closest point $\vec{P_{\mathcal{A}}}$ und Tangente $\vec{D_{\mathcal{A}}}$ auf der Bahn $\mathcal{M}$, zu der Konfiguration $\mathcal{A}$*

Die Funktion `Radius` bekommt als Parameter einen Interpolationswert im Intervall [0, 1] und liefert als Ergebnis den Radius r der Bahn an dieser Stelle, der der Kehrwert zu der Krümmung κ der Kurve ist. Mit dem zuvor berechneten t lässt sich nun der Radius $r_{\mathcal{A}}$ und die Kürmmung $\kappa_{\mathcal{A}}$ in diesem Punkt bestimmen. Aus dem Ackermannmodell (siehe Kapitel 2.1) und (Eq. 2.1) folgt unmittelbar der Lenkwinkel $\varphi_{\mathcal{A}}$, der an dieser Stelle notwendig wäre.

$$\varphi_{\mathcal{A}} = \tan^{-1}\left(b \cdot \kappa_{\mathcal{A}}\right)$$

Der Grund, dass man hier mit der Krümmung, statt wie in (Eq. 2.1) definiert mit dem Radius arbeitet, ist der mathematische Hintergrund, dass der Radius für die Gerade unendlich ist und die Krümmung bei 0 beginnt und anwächst und damit leichter zu rechnen ist.

Wenn die Fahrzeugparameter richtig bestimmt wurden und das Fahrzeug exakt auf der Bahn mit der richtigen Ausrichtung wäre, so wäre $\varphi_{\mathcal{A}}$ der endgültige Lenkwinkel. Da aber nicht alle physikalischen Einflüsse vorab bestimmt werden können, sieht der Normalfall etwas anders aus, dass das Fahrzeug von der Bahn abweicht, somit muss nachgeregelt werden. Dazu wird ein Vorhaltepunkt $\vec{F_{\mathcal{A}}}$ in Richtung der Tangente bestimmt, den das Fahrzeug zusätzlich ansteuern soll. Dafür wird der Winkel γ zwischen der aktuellen Ausrichtung des Fahrzeugs von der Vorderachse zu dem Vorhaltepunkt bestimmt und direkt als Lenkwinkelkorrektur angenommen (Abb. 2.18).

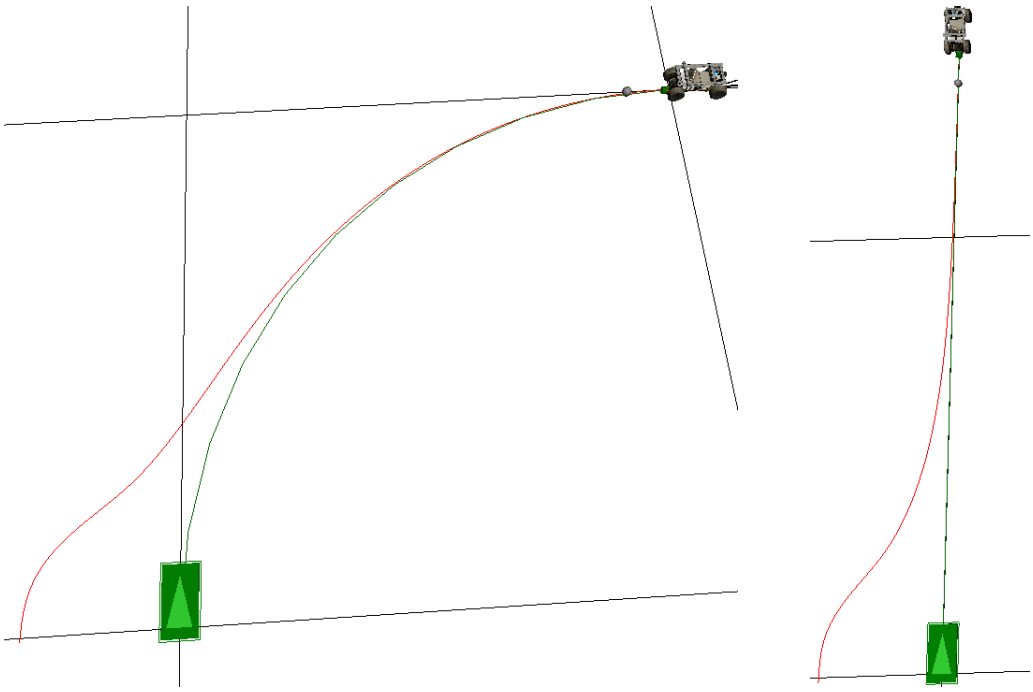


(Abb. 2.18): *Lenkwinkelkorrektur γ, als Winkel zwischen Vorderachse und dem Vorhaltepunkt $\vec{F_{\mathcal{A}}}$*

Durch die Entfernung des Vorhaltepunktes kann das Korrekturverhalten gesteuert werden. Je weiter der Punkt entfernt ist, desto langsamer wirkt sich die Korrektur aus, je näher der Punkt ist, desto schneller reagiert die Korrektur, neigt jedoch dazu schnell auszubrechen und führt zu oszillierendem Verhalten. Der endgültige Lenkwinkel φ ergibt sich aus der Summe, der beiden zuvor berechneten Winkel:

$$\varphi = \varphi_{\mathcal{A}} + \gamma$$

Durch den Korrekturwinkel γ ergibt sich eine asymptotische Annäherung an die vorgegebene Bahn. Je gößer die Ablage, desto stärker lenkt das Fahrzeug in Richtung des Vorhaltepunktes, übersteuert diesen aber nicht, da der Winkel mit sich ändernder Orientierung immer kleiner wird und je näher man dann der Bahn kommt, wird der Lenkwinkel negativ und das Fahrzeug nähert sich der Bahn an (Abb. 2.19).



(a) Kreisbahn **(b)** Gerade

(Abb. 2.19): *Asymptotische Annäherung einer gefahrenen Bahn (rot) zu einer berechneten Bahn (grün).*

2.4.2 GESCHWINDIGKEITSREGELUNG

Bei höheren Geschwindigkeiten rollt der Scooter, nach dem Abschalten des Motors mehrere Meter aus. Wenn man eine Pose möglichst exakt anfahren möchte, dann muss man auch rechtzeitig vom Gas gehen, oder sogar Bremsen, um an der gewünschten Stelle stehen zu bleiben. Auch beim Rangieren ist es praktisch möglichst an der berechneten Stelle zum Stehen zu kommen, als mehrere Meter auszurollen und vom Kurs abzukommen. Dies ist auch wichtig, da teilweise zum Rangieren oder Einparken nah an einer Wand geplant wird und somit gewährleistet werden soll, dass der Scooter rechtzeitig zum Stehen kommt.

Da alle Motion Primitiven und deren Längen bis zum Endpunkt bekannt sind, kann man die Länge der gesamten Bahn ausrechnen. Zu Beginn der Fahrt werden so genannte Bremspunkte berechnet, indem nach alternierenden Fahrtrichtungen (sind in den Motion Primitiven definiert) geschaut wird. Die Bremspunkte sind Angaben in Metern vom Beginn der Bahn. Da die Längen der Motion Primitiven bekannt sind und während der Fahrt der Interpolationsparameter t berechnet wird, weiß man immer, wie weit man auf der aktuellen Bahn ist und kann den Abstand zum nächsten Bremspunkt berechnen. Nun bedient man sich der physikalischen Formeln für Beschleunigung, um auszurechnen wie schnell das Fahrzeug sein muss, um in der berechneten Länge zum Stillstand zu kommen.

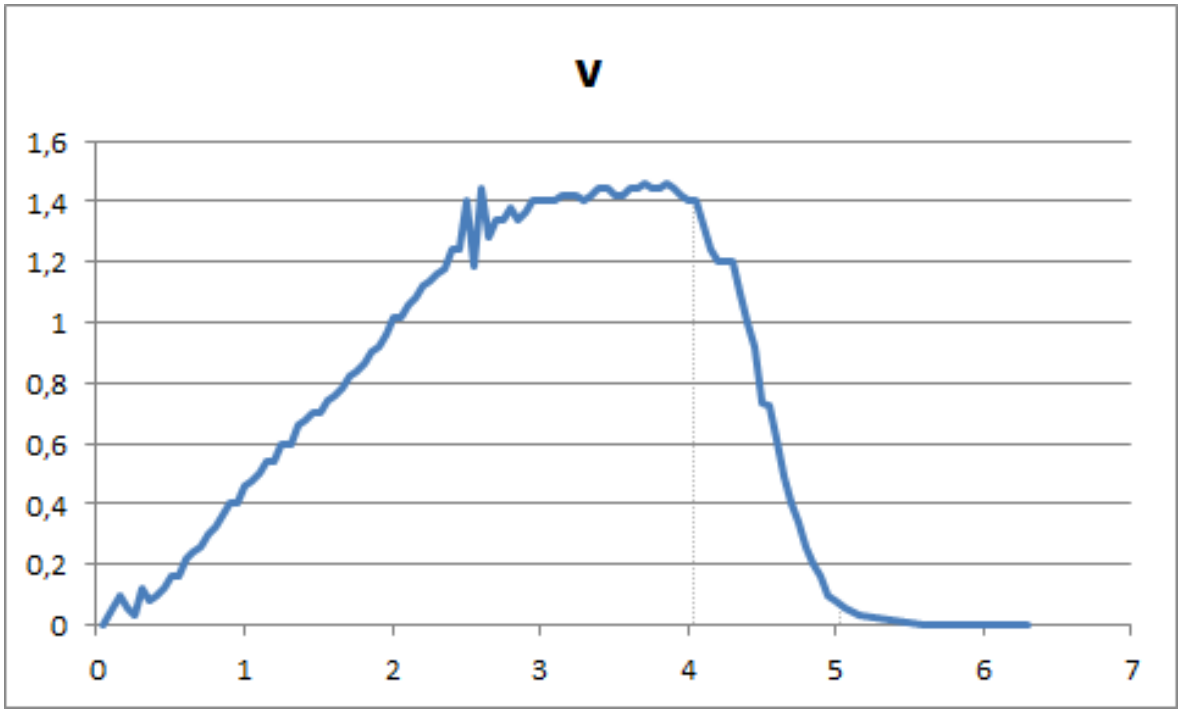
$$s = \frac{1}{2} \cdot a \cdot t^2 \tag{2.7}$$

$$a = \frac{v}{t} \tag{2.8}$$

Dabei ist s der zurückgelegte Weg in m, t die dafür benötigte Zeit in s, v die Geschwindigkeit in $\frac{m}{s}$ und a die so genannte Bremsbeschleunigung in $\frac{m}{s^2}$. Löst man (Eq. 2.8) nach t auf und setzt diese in (Eq. 2.7) ein, so kann man die Formel nach der Geschwindigkeit umformen.

$$s = \frac{v^2}{2a}$$
$$\Rightarrow v = \sqrt{2 \cdot s \cdot a}$$

Dabei ist s der zuvor berechnete Weg bis zu dem nächsten Bremspunkt in Metern. Die Bremsbeschleunigung a ist eine Fahrzeugkonstante und muss vorher für jedes Fahrzeug neu bestimmt werden. In einem Versuch wurde das Fahrzeug auf die maximale Geschwindigkeit beschleunigt und wieder abgebremst. Dabei wurde die Geschwindigkeit aufgezeichnet und nachträglich in einem Diagramm dargestellt (Abb. 2.20).



(Abb. 2.20): *Messung der Bremsbeschleunigung a. Geschwindigkeit v im Verlauf der Zeit t.*

Im Diagramm (Abb. 2.20) erkennt man die Geschwindigkeit v im Verlauf der Zeit t. Der Scooter beschleunigt dabei innerhalb von $2.5s$ von 0 auf ca $1.4\frac{m}{s}$. Bei $4s$ fängt der Scooter an zu bremsen und kommt bei $5s$ zum Stillstand. Nach (Eq. 2.8) kann die Bremsbeschleunigung bestimmt werden.

$$a = \frac{v}{t} \approx \frac{1.4\frac{m}{s}}{1s} = 1.4\frac{m}{s^2}$$

Wählt man die Bremsbeschleunigung zu hoch, so wird der Bremsweg viel kürzer eingeschätzt und das Fahrzeug fängt später an zu bremsen und wird über das Ziel hinausschießen. Wählt man die Bremsbeschleunigung geringer, so wird der Bremsweg länger eingeschätzt und das Fahrzeug beginnt früher an zu bremsen und kann während des Bremsvorgangs geregelt werden. Hier fängt die eigentliche Regelung an. Dabei wird ständig die tatsächliche Geschwindigkeit überwacht und sollte diese größer sein, als die berechnete, so wird eine Motorbremsung ausgeführt, indem volle Kraft in entgegengesetzte Richtung gegeben wird. Bei größeren Entfernungen errechnen sich surreal hohe Geschwindigkeitswerte, die auf die maximale Geschwindigkeit $v_{\max}$ beschränkt werden müssen.

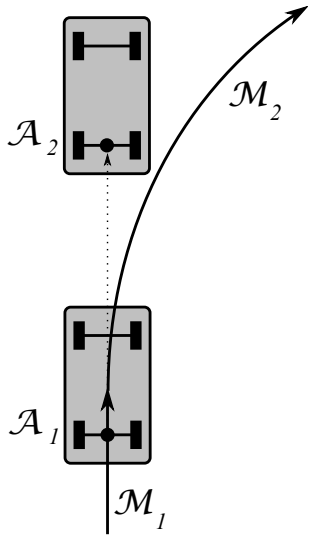
$$v = \min\{v_{\max}, \sqrt{2 \cdot s \cdot a}\}$$

Da man mit dieser Formel zu früh zum Stillstand kommen könnte und damit nicht mehr den Endpunkt erreicht, sollte man nicht langsamer fahren, als eine Minimalgeschwindigkeit $v_{\min}$, die so gewählt werden sollte, dass der Motor dass Fahrzeug noch in Bewegung hält und der Ausrollweg vernachlässigbar ist. Somit lautet die endgültige Formel für die Geschwindigkeit.

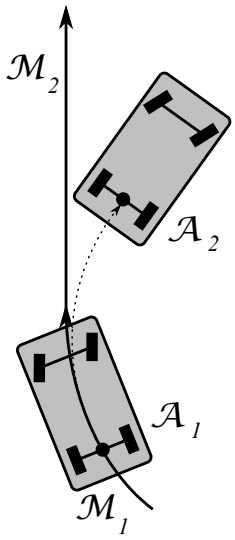
$$v = \max\{v_{\min}, \min\{v_{\max}, \sqrt{2 \cdot s \cdot a}\}\}$$

2.4.3 Look Ahead

Die Regelung arbeitet in diskreten Zeitabständen (normalerweise mit $50Hz$). Wenn also das Fahrzeug aus einer Geraden in eine Kurve fährt und die Regelung erkennt, dass eine Kurve vorliegt, ist das Fahrzeug bereits einige Meter geradeaus in der Kurve gefahren und fängt zu spät an einzulenken. Aufgrund physikalischer Einschränkungen braucht die Lenkung einige Zeit, bis sich der gewollte Lenkwinkel einstellt. In dieser Zeit ist das Fahrzeug bereits weiter gefahren und misst einen noch höheren Lenkeinschlag, hat aber erst den Lenkeinschlag der letzten Messung eingestellt, so verpasst das Fahrzeug die Kurveneinfahrt und entfernt sich immer mehr von der Bahn, was extra nachgeregelt werden muss. Selbiges Problem entsteht bei der Kurvenausfahrt, denn das Fahrzeug hat noch einen Lenkwinkel berechnet und hält diesen ein, im nächsten Schritt wird eine Gerade erkannt, wobei das Fahrzeug noch einen Lenkeinschlag hat und bereits an der Ausfahrt vorbeigefahren ist. Erst jetzt wird der Lenkmotor in Gang gesetzt und das Fahrzeug übersteuert noch mehr bei Kurvenausfahrt (Abb. 2.21).



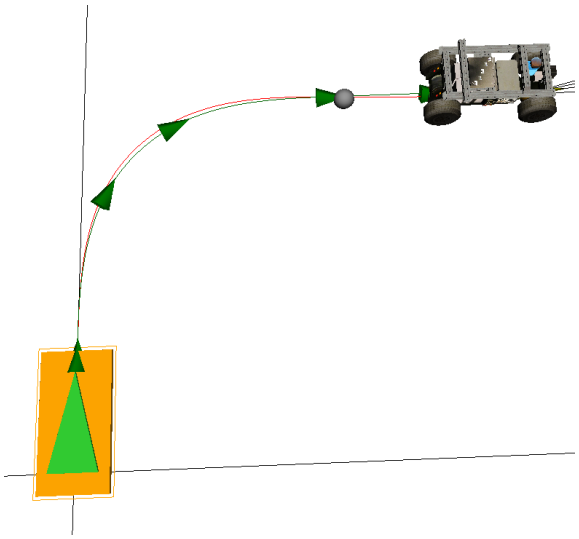
(a) Fahrzeug verpasst Kurveneinfahrt.



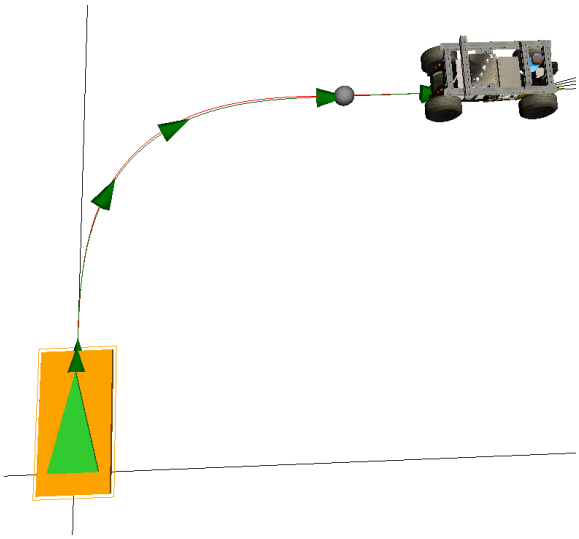
(b) Fahrzeug verpasst Kurvenausfahrt.

(Abb. 2.21): *Diskrete Messungen von Fahrzeugkonfigurationen $\mathcal{A}_1$ und $\mathcal{A}_2$ zwischen zwei Bahnsegmenten $\mathcal{M}_1$ und $\mathcal{M}_2$.*

Um diesem Phänomen vorzubeugen wird anhand der Fahrzeugkonfiguration $\mathcal{A}_1$ die darauf folgende Konfiguration $\mathcal{A}_2$ mit den aktuellen Parametern berechnet. Die Regelung arbeitet dann anhand der neu berechneten „Look Ahead“-Konfiguration. Dies hat zur Folge, dass das Fahrzeug bereits in der Geraden anfängt einzulenken und somit genauer der Bahn folgt. Auch bei Kurvenausfahrt wird schon früher angefangen den Lenkwinkel zu reduzieren, so dass das Fahrzeug nicht übersteuert (Abb. 2.22).



(a) Standard Bahnregler.

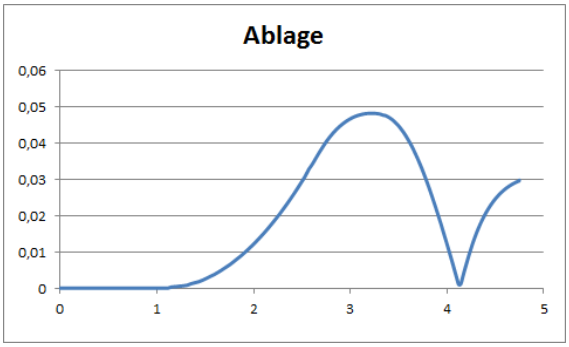


(b) Bahnregler mit Look Ahead.

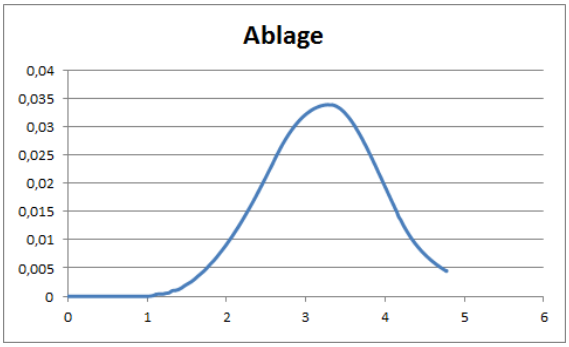
(Abb. 2.22): *Simulationsversuch mit verschiedenen Bahnreglern. Gefahrene Bahn (rot) auf einer vorberechneten Testbahn (grün).*

In einem Simulationsversuch wurden beide Bahnregler getestet (Abb. 2.22). Man erkennt, dass der Standard Bahnregler zu spät einlenkt und somit eine größere Ablage von der Bahn bekommt, als der Look Ahead Bahnregler (vgl. Abb. 2.23). Am Ende der Bahn lenkt dieser auch zu spät aus und übersteuert in der Geraden. Der Look Ahead Bahnregler lenkt sauber in die Kurve ein, hat nahezu die gesamte Kurve lang keine Ablage von der Bahn und übersteuert auch nicht zum Schluß.

Viel genauer lässt sich das anhand einer Auswertung der aufgezeichneten Pfade zeigen. Anhand des Ablagegraphen sieht man, dass die maximale Ablage im standard Bahnregler auf $\approx 0.05m$ ansteigt, wohingegen diese beim Look Ahead Regler den Wert von $\approx 0.035m$ nicht übersteigt (Abb. 2.23). Summiert man die Abstände multipliziert mit den Längendifferenzen Δl zu den jeweiligen Zeitpunkten auf, so entspricht es dem Integral über die Ablagefunktion. Dieser zeigt den gesamten kumulirten Fehler in der Ablage. Bei dem standard Bahnregler erreicht dieser einen Wert von $\approx 0.09m^2$, beim Look Ahead Regler geht dieser knapp über einen Wert von $\approx 0.06m^2$ (Abb. 2.24).

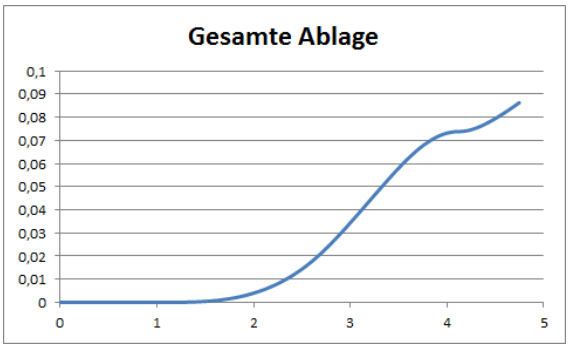


(a) Standard Bahnregler.

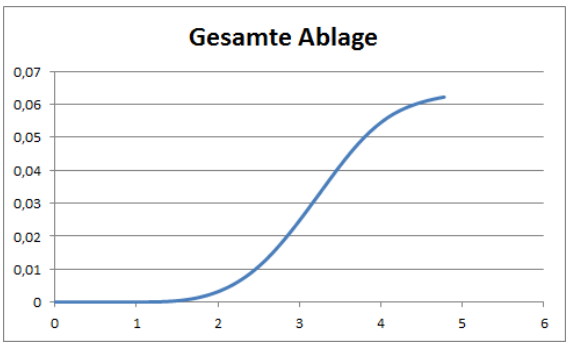


(b) Look Ahead Bahnregler.

(Abb. 2.23): *Aufgezeichnete Ablagedaten $dis(l)$ im Verlauf der Länge l in dem Szenario (Abb. 2.22).*



(a) Standard Bahnregler.



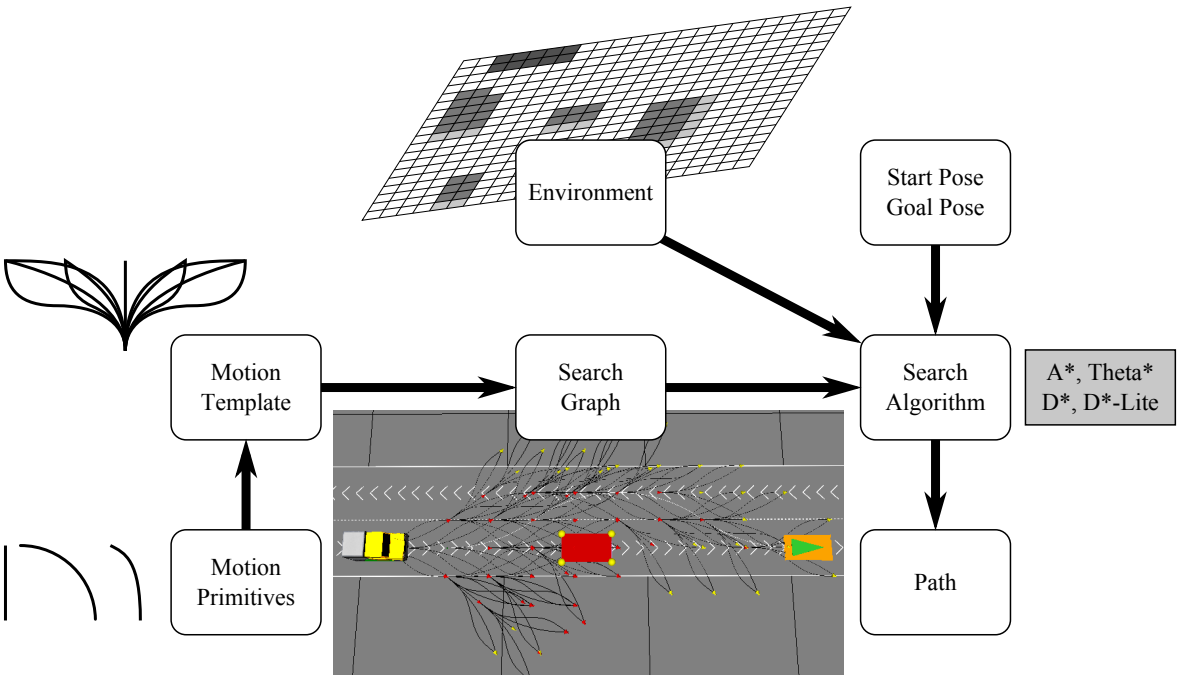
(b) Look Ahead Bahnregler.

(Abb. 2.24): *Integral über die Ablagefunktion $\int dis(l)dl$ in dem Szenario (Abb. 2.22).*

3

ENTWICKELTES FRAMEWORK

Die Abbildung (Abb. 3.1) zeigt das Architekturdiagramm des entwickelten Frameworks. Die Suche nach einem Pfad ist aus mehreren „austauschbaren“ Komponenten aufgebaut:



(Abb. 3.1): *Framework Architektur.*

Motion Primitives: Diese stellen die kleinsten Pfade, die das Fahrzeug mit allen seinen kinematischen Einschränkungen abfahren kann. Diese beinhalten die Starpose, die Endpose und eine Funktion, die alle Punkte zwischen Star- und Endpose liefert und somit die Bahn darstellt (Kapitel 4).

Motion Template: Diese beinhalten eine Menge an Motion Primitiven, mit denen Die Suche abläuft. Sie definieren die erreichbaren Konfigurationen aus einer Konfiguration heraus. Es gibt mehrere Möglichkeiten diese Menge zu berechnen, worauf in Sampling (Kapitel 5) eingegangen wird.

Search Graph: Der Search Graph stellt den Erreichbarkeitsbaum dar, der zur Laufzeit aufgebaut wird. Je besser der Suchalgorithmus, desto kleiner bleibt dieser Baum und breitet sich in Richtung des Ziels aus.

Environment: Das Environment beinhaltet die erkannten Objekte in der Umgebung (Kapitel 2.2) als Grid, oder die statischen Objekte, die mithilfe des Objekteditors als Polygone erstellt wurden. Auf diesen wird später die Kollisionserkennung (Kapitel: 2.3) stattfinden.

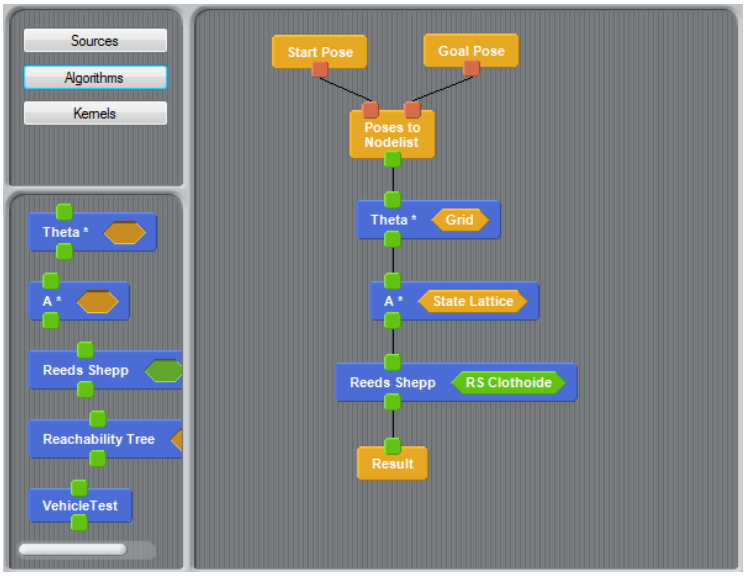
Start Pose & Goal Pose: Die Start Pose (grün mit Dreieck) und die Goal Pose (orange mit Dreieck) stellen Konfigurationen dar, die in der 3D-Ansicht verschoben und rotiert werden können. Auf diese kann in den Algorithmen zugegriffen werden, um diese als Suchpunkte zu benutzen.

Search Algorithm: Dieser stellt das Herzstück der Bahnplanung dar und beinhaltet den Algorithmus, mit dem von einer Start Pose zu einer Goal Pose gesucht wird, dabei operiert der Algorithmus auf dem Search Graphen, der zur Laufzeit aufgebaut wird. Mithilfe der Kollisionserkennung, die auf dem Environment arbeitet, werden Pfade durch Hindernisse ausgeschlossen. Verschiedene Algorithmen und deren Optimierungsmöglichkeiten werden in Suchalgorithmen (Kapitel 6) betrachtet.

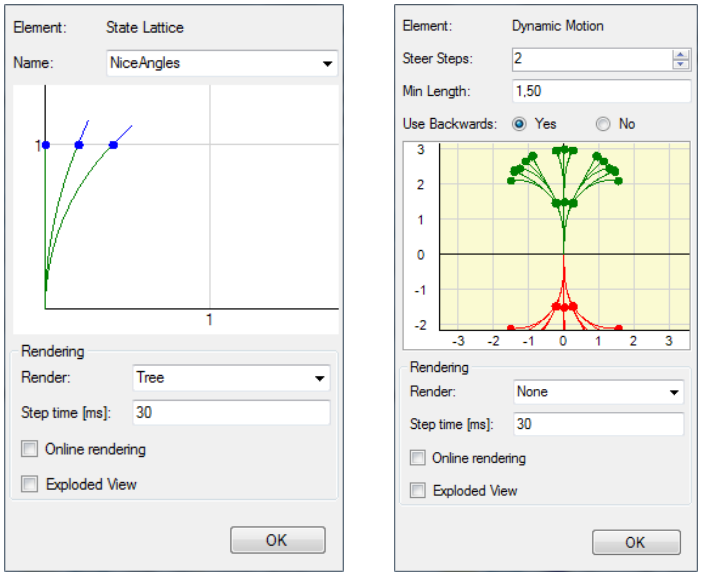
Path: Ist eine Liste von Motion Primitiven und das gesuchte Ergebnis der Bahnplanung.

3.1 Gaphische Programmierung

Zum Testen der Algorithmen und um die Parametrisierbarkeit derer zu gewährleisten, wurde ein graphisches Programmierwerkzeug implementiert. Dabei können mehrere Algorithmen graphisch per Drag & Drop miteinander verbunden werden und deren Parameter angepasst werden. Dadurch entsteht ein mächtiges Werkzeug, welches es erlaubt die Algorithmen auszuprobieren und auch mehrere miteinander zu kombinieren. Abbildung (Abb. 3.2) zeigt ein Screenshot des entwickelten graphischen Werkzeugs. Dabei kann oben links die Art der Elemente ausgewählt werden, unten links ist die Liste dieser Elemente aufgeführt und rechts die Kombinationskette der Algorithmen. Elemente aus der linken Spalte können per Drag & Drop in das rechte Fenster geschoben werden und miteinander verbunden werden. Mit der rechten Maustaste öffnet sich das Parameterfenster, wo man die Einstellungen für das Element ändern kann (Abb. 3.3).



(Abb. 3.2): *Entwickeltes Framework.*

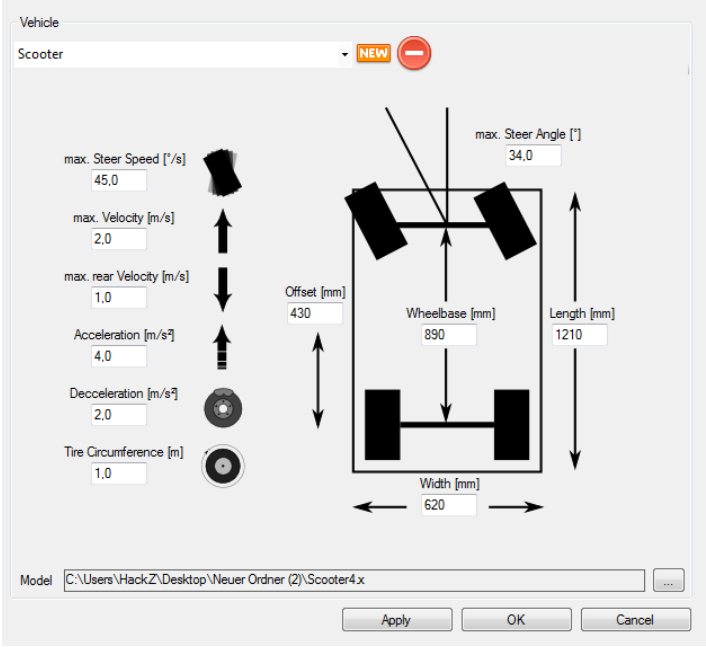


(a) State Lattice **(b)** Adaptive Search Space

(Abb. 3.3): *Parameter verschiedener Elemente.*

3.2 VEHICLE PARAMETERS

Die Vehicle Parameters bieten die Möglichkeit die physikalischen und geometrischen Größen des Fahrzeugs einzugeben, so dass die Bahnplanung für alle Fahrzeuge und ihre spezifischen kinematischen Eigenschaften einen Weg planen kann (Abb. 3.4).



(Abb. 3.4): *Vehicle Parameters.*

GEOMETRISCHE GRÖSSEN

Width: Ist die Breite des Fahrzeugs in *mm* und wird für die Kollisionserkennung benötigt.

Length: Ist die Länge des Fahrzeugs in *mm* und wird für die Kollisionserkennung benötigt.

Offset: Ist die Verschiebung des Fahrzeugs in *mm* an der X-Achse, so dass die Mitte der hinteren Achse in den Ursprung kommt. Wird für die Kollisionserkennung beötigt.

Wheelbase: Ist der Abstand von der Vorder- zu der Hinterachse in *mm* und stellt eine wichtige Größe dar, die gebraucht wird um den minimalen Wendekreis zu berechnen (Kapitel 2.1)

max. Steer Angle: Ist der maximal mögliche Lenkeinschlag pro Seite in Grad. Diese Größe wird auch zum Berechnen des minimalen Wendekreises benötigt. Des Weiteren wird die Wheelbase und der maximale Lenkeinschlag benötigt um die Fahrbarkeit von Motion Primitiven zu validieren.

Tire Circumference: Ist der Umfang des Reifens und wird für die Animation des Modells benötigt und spielt für die Bahnplanung keine weitere Rolle.

Model: Ist der Pfad zu einem DirectX-Modell, welches für die Darstellung benutzt werden soll. Spielt für die Bahnplanung keine Rolle. Bei einem leeren Pfad wird eine Box mit der angegebenen Länge und Breite gerendert.

PHYSIKALISCHE GRÖSSEN

max. Steer Speed: maximale Geschwindigkeit der Lenkbewegung in Grad pro Sekunde. Gibt an wie schnell der Servomotor des Fahrzeugs den Lenkwinkel ändern kann. Wird benötigt um Klothoiden und Continuous Steering Primitiven zu berechnen (siehe Kapitel 4).

max. Velocity: Ist die maximale Geschwindigkeit des Fahrzeugs in $\frac{m}{s}$. Wird benötigt um Klothoiden und Continuous Steering Primitiven zu berechnen.

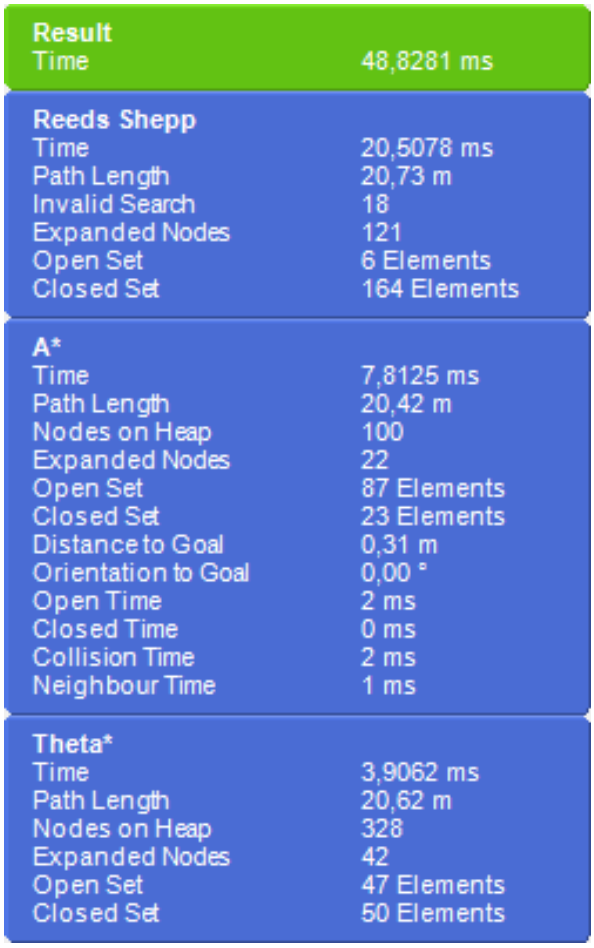
max. rear Velocity: Ist die maximale Rückwärtsgeschwindigkeit des Fahrzeugs in $\frac{m}{s}$. Wird benötigt um Klothoiden und Continuous Steering Primitiven zu berechnen.

Acceleration: Ist die Beschleunigung des Fahrzeugs in $\frac{m}{s^2}$. Wird für die Simulation des Fahrzeugs benötigt, damit das Simulationsfahrzeug ähnliche Eigenschaften wie das echte Fahrzeug besitzt.

Decceleration: Ist die Bremskraft des Fahrzeugs in $\frac{m}{s^2}$. Wird für die Berechnung der Bremspunkte für die Geschwindigkeitsregelung (siehe Kapitel 2.4.2) benötigt.

3.3 STATISTIK

Zu weiteren Besonderheiten des Bahnplanungsframeworks gehört die umfangreiche Statistikauswertung. Diese erfasst bereits jetzt bis zu fünf verschiedene Zeitmessungen und liefert zusätzlich Informationen zu verbrauchten Datenmengen (Abb. 3.5). Diese Statistiken können nachträglich genutzt werden, um Algorithmen gezielt an ihren Schwächen zu verbessern.



(Abb. 3.5): *Statistikauswertung.*

Diese Statistiken können erfasst werden und haben folgende Bedeutungen:

Time: Misst die gesamte Laufzeit des Algorithmus in *ms*, von Aufruf bis dessen Abarbeitung.

Path Length: Die Länge des gefundenen Pfades in Metern.

Distance to Goal: Die heuristischen Suchen liefern kein exaktes Ergebnis bis zum Ziel. Hier ist der Abstand der letzten Konfiguration vom gewünschten Ziel in Metern.

Orientation to Goal: Auch die Orientierung spielt bei heuristischen Suchen keine Rolle, hier steht die Abweichung von der gewünschten Orientierung in Grad.

Invalid Search: Nur bei Reeds Shepp. Reeds Shepp sucht für alle gelieferten Punkte, ob es einen kollisionsfreien Weg findet, ab dem ersten. Der erste Weg, ohne Kollision wird zurückgegeben, alle anderen waren nicht erfolgreich, deren Anzahl hier gemessen wird.

Expanded Nodes: Zählt wie oft eine Konfiguration expandiert wird, also deren Nachfolgekonfigurationen berechnet werden. Dabei werden mehrere Folgekonfigurationen bei einer Expansion berechnet.

Nodes on Heap: Jede Konfiguration die neu berechnet wird und somit auf den Arbeitsspeicher kommt wird hier gezählt.

Open Set: Anzahl der Konfigurationen in dem Open Set.

Closed Set: Anzahl der Konfigurationen in dem Closed Set.

Open Time: Die Zeit in *ms*, die der Algorithmus für Zugriffe auf das Open Set benötigt.

Closed Time: Die Zeit in *ms*, die der Algorithmus für Zugriffe auf das Closed Set benötigt.

Collision Time: Die Zeit in *ms*, die der Algorithmus für Kollisionserkennung braucht.

Neighbour Time: Die Zeit in *ms*, die der Algorithmus für das Erstellen neuer Nachbarkonfigurationen (Expandieren) braucht.

4

MOTION PRIMITIVES

Eine Motion Primitive beschreibt eine Bahn von einer Startkonfiguration zu einer Endkonfiguration. Es gibt zunächst unendlich viele Möglichkeiten von einer gegebenen Startkonfiguration in die Endkonfiguration zu kommen, so dass die Motion Primitive Funktionen bietet, um die interpolierten Konfigurationen dazwischen zu berechnen. Interessant an dieser Stelle sind nur Primitiven, deren Bahn von einem Fahrzeug mit all seinen kinematischen Einschränkungen befahrbar ist. Start- und Endkonfigurationen liegen dabei als affine Transformationsmatrizen vor. Für eine einfache Berechenbarkeit der Bahn wird die Einheitsmatrix als Starttransformation verwendet (Pose im Ursprung mit Orientierung $\theta = 0$).

Sei die Motion Primitive folgendermaßen definiert:

$$\mathcal{M} = \{\vec{S_{\mathcal{M}}}, \vec{G_{\mathcal{M}}}, F_{\mathcal{M}}(t) | \vec{S_{\mathcal{M}}}, \vec{G_{\mathcal{M}}} \in \mathbb{R}^3, t \in \mathbb{R}, 0 \leq t \leq 1\}$$

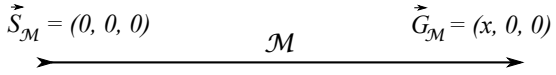
Dabei bezeichnet $F_{\mathcal{M}}(t)$ eine Funktion, die zu einem Wert t im Intervall [0, 1] die interpolierte Konfiguration von Start- zur Endkonfiguration liefert. Diese Funktion ist nur für die Regelung oder die Visualisation der Bahn interessant. Der Wert $\vec{S_{\mathcal{M}}}$ steht für die Startkonfiguration und $\vec{G_{\mathcal{M}}}$ für die Endkonfiguration (engl. goal), die als Parameter die Position x, y und die Orientierung θ haben. Nach (Eq. 2.3) können daraus die affinen Transformationsmatrizen $M_{S_{\mathcal{M}}}$ und $M_{G_{\mathcal{M}}}$ berechnet werden. Sei weiterhin $M_{\mathcal{A},i}$ die aktuelle Transformation des Fahrzeugs, so berechnet sich die Endtransformation $M_{\mathcal{A},i+1}$, nach Durchfahren der Motion Primitive, unter der Voraussetzung, dass die Starttransformation $M_{S_{\mathcal{M}}}$ die Einheitsmatrix ist, wie folgt:

$$M_{\mathcal{A},i+1} = M_{G_{\mathcal{M}}} \cdot M_{\mathcal{A},i}$$

So entspricht die gesamte Endkonfiguration nach mehreren Motion Primitiven dem Produkt aller Endtransformationsmatrizen. Da Matrixmultiplikation nicht kommutativ ist, muss darauf geachtet werden, dass die Bahnsegmente jeweils nacheinander von links multipliziert werden.

4.1 GERADE

Die Gerade ist die einfachste Primitive. Unter der Voraussetzung, dass die Starttransformation die Einheitsmatrix ist, so wird diese komplett über den x-Wert der Endkonfiguration bestimmt und hat eine Länge von $|x|$. Im Normalfall hat die Gerade als Orientierung $\theta = 0$, denn sollte θ einen anderen Wert annehmen, entsteht bei der Multiplikation der Folgeprimitive ein Knick in der Bahn, der von einem Fahrzeug mit Achsschenkellenkung kinematisch gar nicht befahren werden kann, wäre aber ideal und leicht zu berechnen für Fahrzeuge, die auf der Stelle wenden können (Abb. 4.1).

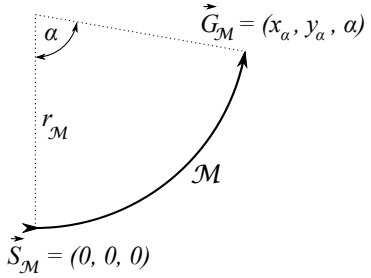


(Abb. 4.1): *Motion Primitive: Gerade*

4.2 KREISBOGEN

Mit den Kreisbögen wird die erste Art Kurven zu durchfahren gezeigt. Der Vorteil von Kreisbögen ist die leichte mathematische Berechenbarkeit. Will man die Richtung des Fahrzeugs um α ändern, so muss den innere Winkel des Kreisbogens α betragen. Der Radius entscheidet, wie eng man die Kurve nimmt und ist durch den minimalen Radius begrenzt, der sich aus dem maximalen Lenkwinkel und dem Achsenabstand (engl. wheelbase) ergibt (Eq. 2.2).

Es gibt zwei Möglichkeiten den gewünschten Kreisbogen zu berechnen. Entweder man möchte das Fahrzeug um den Winkel α abbiegen lassen und muss somit x_α und y_α berechnen, die sich aus α und $r_{\mathcal{M}}$ ergeben, welches in dem Fall meist die engste Kurve mit $r_{\min}$ ist. Oder man möchte das Fahrzeug an die Stelle x_α, y_α bringen, wodurch sich daraus α und $r_{\mathcal{M}}$ ergeben. Dabei muss darauf geachtet werden, dass $r_{\mathcal{M}} \geq r_{\min}$ sein muss, ansonsten ist dieser Kreisbogen nicht valide und kann nicht von dem Fahrzeug mit den angegebenen Parametern befahren werden (Abb. 4.2).



(Abb. 4.2): *Motion Primitive: Kreisbogen*

Für gegebenes α ergibt sich (bei einer Linkskurve ist α und $r_{\mathcal{M}}$ negativ):

$$x_\alpha = r_{\mathcal{M}} \cdot \sin\alpha$$
$$y_\alpha = r_{\mathcal{M}} \cdot (\cos\alpha - 1)$$

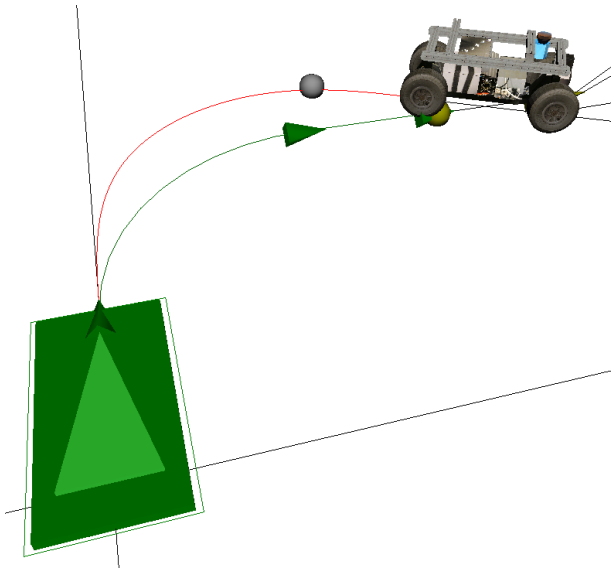
Gewünschte Position (x_α, y_α) ist gegeben:

$$r_{\mathcal{M}} = -\frac{x_\alpha^2 + y_\alpha^2}{2 \cdot y_\alpha}$$
$$\alpha = \sin^{-1}\left(\frac{x_\alpha}{r_{\mathcal{M}}}\right)$$

Dabei ist $r_{\mathcal{M}}$ der Vorzeichenbehaftete Radius, wobei negative Radien die Linkskurven sind und positive Radien die Rechtskurven sind. Selbiges gilt für den Winkel α, positive Werte sind Rechtskurven und negative Werte sind Linkskurven.

Die Übergänge zwischen zwei Kreisbögen oder Kreisbogen und Gerade sind zwar knickfrei, können jedoch nicht so leicht von einem Fahrzeug befahren werden, denn die Kreisbögen haben eine

Konstante Krümmung, die beim Übergang von der Geraden mit Krümmung $\kappa = 0$ auf die Krümmung $\kappa = \frac{1}{r_{\mathcal{M}}}$ springt. Physikalisch muss aber der Lenkwinkel über den Motor eingestellt werden, was etwas Zeit braucht, in der Zeit jedoch das Fahrzeug weiter fährt und somit nicht die Kurve kriegen kann (Abb. 4.3). Um die Kurve perfekt zu kriegen, müsste das Fahrzeug vor dem Kreisbogen anhalten, den Lenkwinkel einstellen und wieder losfahren. Man spricht dabei von einem nicht ruckfreien Übergang.



(Abb. 4.3): *Fahrzeug wechselt von einer Geraden auf eine Kreisbahn. Aufgrund physikalischer Lenkbewegung wird nicht sofort der Kreisbahnlenkwinkel eingestellt und das Fahrzeug driftet aus der Kurve. Rot ist der vom Bahnregler gefahrene Pfad und grün der berechnete Pfad.*

4.3 KLOTHOIDE

Aufgrund der Tatsache, dass Kreisbögen nicht ruckfrei verbunden werden können und somit nicht gut befahrbar sind, braucht man mathematisch komplexere Funktionen als Verbindung zwischen Start- und Endkonfiguration. Im Straßenbau werden als Kurvenübergänge Klothoiden verwendet, diese haben eine linear ansteigende Krümmung und eignen sich als ruckfreier Übergang zwischen zwei Bahnsegmenten, deren End- bzw. Startkrümmung bekannt sind. Der Normalfall ist der Übergang von einer Geraden zu einem Kreisbogen, wo die Krümmung $\kappa = 0$ auf $\kappa = \frac{1}{r_{\mathcal{M}}}$ (mit Radius $r_{\mathcal{M}}$ des Kreisbogens) linear ansteigt.

Ohne Einschränkung der Allgemeinheit, sei die Startpose im Ursprung mit Orientierung $\theta = 0$ (in Richtung der X-Achse). Somit folgt unmittelbar aus der Definition für die Krümmung in Abhängigkeit der Länge l der Bahn:

$$\kappa(l) = \sigma \cdot l \tag{4.1}$$

Dabei beschreibt σ die so genannte Schärfe der Klothoide, mit der sich die Krümmung verändert. Somit ändert sich die Orientierung des Fahrzeugs um die aktuelle Krümmung.

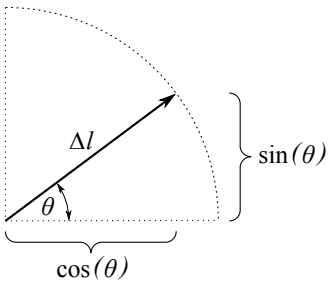
$$\dot{\theta}(l) = \kappa(l)$$

Die Orientierung beträgt dann im Verlauf der Länge:

$$\theta(l) = \int_0^l \kappa(t)\mathrm{d}t = \frac{\sigma l^2}{2} \tag{4.2}$$

Für die Änderung der Koordinaten in Abhängigkeit von der Länge gilt (Abb. 4.4):

$$\dot{x}(l) = \Delta l \cdot \cos(\theta(l))$$
$$\dot{y}(l) = \Delta l \cdot \sin(\theta(l))$$



(Abb. 4.4): *Änderung der Position, für den Winkel θ und die Distanz Δl.*

Macht man den Schritt Δl infitisimal klein, somit berechnen sich die endgültigen Koordinaten in Abhängigkeit zu der Länge l über das Integral:

$$x(l) = \int_0^l \cos(\theta(t))\mathrm{d}t \overset{(Eq.4.2)}{=} \int_0^l \cos\left(\frac{\sigma t^2}{2}\right)\mathrm{d}t$$
$$y(l) = \int_0^l \sin(\theta(t))\mathrm{d}t \overset{(Eq.4.2)}{=} \int_0^l \sin\left(\frac{\sigma t^2}{2}\right)\mathrm{d}t$$

Diese Integrale lassen sich nicht ohne weiteres berechnen. Es gibt aber die Fresnel-Integrale, für die es bereits effektive Programmbibliotheken gibt und in die sich diese Formeln mithilfe der Substitution überführen lassen können. Die Fresnel-Integrale sind folgendermaßen definiert:

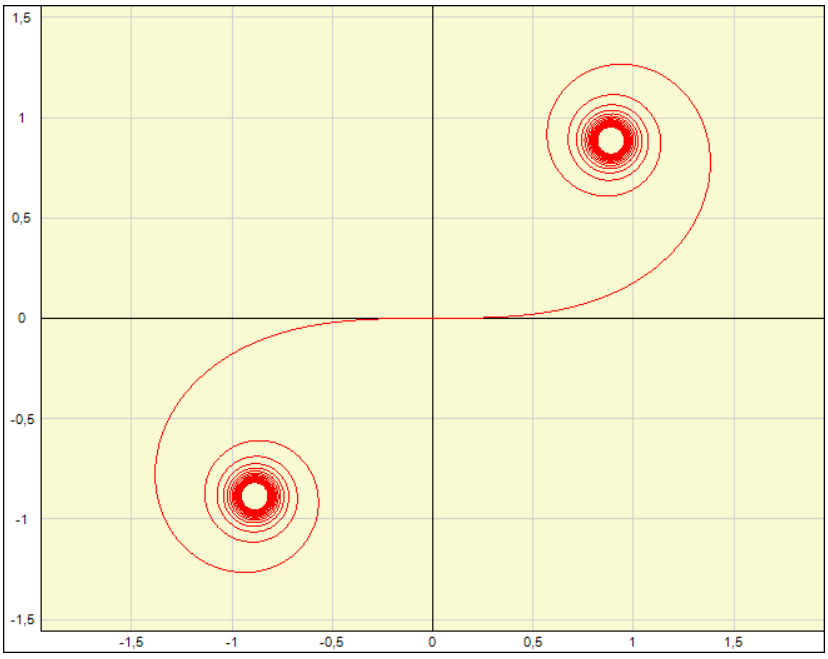
$$C_f(x) = \int_0^x \cos\left(\frac{\pi}{2} \cdot t^2\right)\mathrm{d}t \tag{4.3}$$
$$S_f(x) = \int_0^x \sin\left(\frac{\pi}{2} \cdot t^2\right)\mathrm{d}t \tag{4.4}$$

Substitution:

$$\frac{\pi u^2}{2} = \frac{\sigma t^2}{2} \quad \Rightarrow \quad u = \sqrt{\frac{\sigma}{\pi}} \cdot t \tag{4.5}$$
$$\mathrm{d}u = \sqrt{\frac{\sigma}{\pi}} \cdot \mathrm{d}t \quad \Rightarrow \quad \mathrm{d}t = \mathrm{d}u \cdot \sqrt{\frac{\pi}{\sigma}} \tag{4.6}$$

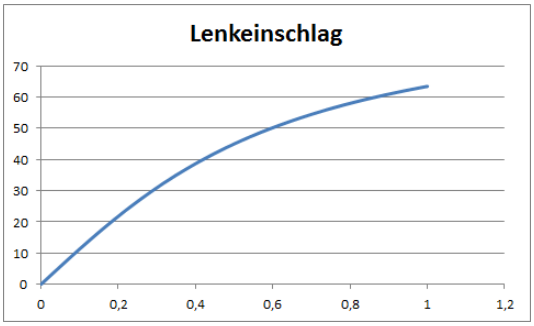
$$
\begin{aligned}
x(l) &= \int_0^l \cos\left(\frac{\sigma t^2}{2}\right) \mathrm{d}t \\
&= \int_0^{\sqrt{\frac{\sigma}{\pi}}\cdot l} \cos\left(\frac{\pi u^2}{2}\right) \mathrm{d}u \cdot \sqrt{\frac{\pi}{\sigma}} && \mid (Eq.4.5)\&(Eq.4.6) \\
&= \sqrt{\frac{\pi}{\sigma}} \cdot C_f\left(\sqrt{\frac{\sigma}{\pi}} \cdot l\right) && \mid (Eq.4.3) \\
y(l) &= \int_0^l \sin\left(\frac{\sigma t^2}{2}\right) \mathrm{d}t \\
&= \int_0^{\sqrt{\frac{\sigma}{\pi}}\cdot l} \sin\left(\frac{\pi u^2}{2}\right) \mathrm{d}u \cdot \sqrt{\frac{\pi}{\sigma}} && \mid (Eq.4.5)\&(Eq.4.6) \\
&= \sqrt{\frac{\pi}{\sigma}} \cdot S_f\left(\sqrt{\frac{\sigma}{\pi}} \cdot l\right) && \mid (Eq.4.4)
\end{aligned}
$$

Trägt man die (x, y)-Werte gegeneinander ins kartesische Koordinatensystem ein, so erkennt man die Form der Klothoide (Abb. 4.5).

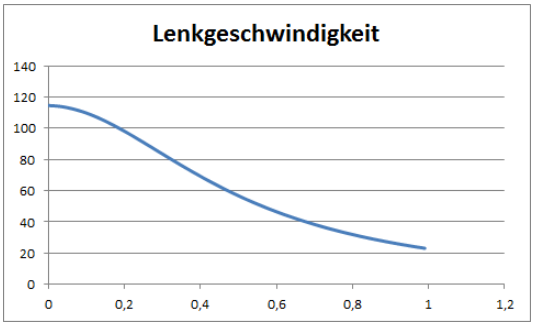


(Abb. 4.5): *Graph einer Klothoide mit* $\sigma = 1$.

Dieser Übergang ist ruckfrei und für einen Menschen bequem zu durchfahren, da man mit fortlaufender Kurvenfahrt das Lenkrad immer weiter dreht und somit der Kurve folgt. In einem ATF lenkt jedoch ein Servomotor, der eine maximale Lenkgeschwindigkeit $v_{\mathrm{steer,max}}$ hat, die nicht überschritten werden kann. Bei der Klothoide steigt die Krümmung κ linear an, trägt man aber den Lenkeinschlag φ in Abhängigkeit von der Länge l ein, so sieht man, dass dieser Verlauf nicht linear ist. Anhand der Lenkgeschwindigkeit v_{steer}, die die Ableitung des Lenkeinschlags φ ist, erkennt man, dass der Anfang einer Klothoide die schnellste Lenkbewegung nach sich zieht und im Verlauf der Kurve abnimmt (Abb. 4.6).



(a) Lenkeinschlag φ im Verlauf der Länge l.



(b) Lenkgeschwindigkeit v_{steer} im Verlauf der Länge l

(Abb. 4.6): *Funktionsgraphen für Lenkeinschlag $\varphi(l)$ und Lenkgeschwindigkeit $v_{\text{steer}}(l)$ einer Klothoide mit $\sigma = 1$.*

Möchte man also die Lenkgeschwindigkeit v_{steer} begrenzen, so muss man darauf achten, dass die Ableitung des Lenkeinschlags φ an der Stelle 0 die maximale Lenkgeschwindigkeit $v_{\text{steer,max}}$ nicht übersteigt. Aus dem Ackermannmodell (siehe Kapitel 2.1) und (Eq. 2.1) lässt sich der Lenkeinschlag φ im Verlauf der Länge l beschreiben als:

$$\begin{aligned} \varphi(l) &= \tan^{-1}(b \cdot \kappa(l)) \\ &= \tan^{-1}(b \cdot \sigma \cdot l) \qquad\qquad | \, (Eq.4.1) \end{aligned}$$

Bezieht man noch die Geschwindigkeit $v_{\max}$ des Fahrzeugs mit ein, so folgt über die physikalischen Zusammenhang der Zeit t zur Länge l:

$$l(t) = v_{\max} \cdot t \tag{4.7}$$

Somit folgt für den Lenkeinschlag φ in Abhängigkeit von der Zeit t:

$$\varphi(t) = \tan^{-1}(b \cdot \sigma \cdot v_{\max} \cdot t)$$

Die Ableitung dieser Funktion an der Stelle 0 ist die maximale Lenkgeschwindigkeit $v_{\text{steer,max}}$, die zu Beginn der Klothoide entsteht:

$$\begin{aligned} \dot{\varphi}(t) &= \frac{b \cdot \sigma \cdot v_{\max}}{b^2 \cdot \sigma^2 \cdot v_{\max}^2 \cdot t^2 + 1} \\ v_{\text{steer,max}} = \dot{\varphi}(0) &= b \cdot \sigma \cdot v_{\max} \end{aligned}$$

Möchte man also die Krümmungsänderung σ der Klothoide so begrenzen, dass das Fahrzeug diese mit all seinen kinematischen Einschränkungen durchfahren kann, so berechnet sich $\sigma_{\max}$:

$$\sigma_{\max} = \frac{v_{\text{steer,max}}}{v_{\max} \cdot b} \tag{4.8}$$

4.4 CONTINUOUS STEERING

Aufgrund der Tatsache, dass die Klothoide in ihrer Krümmungsänderung σ begrenzt werden muss, wodurch der Servomotor nur zu Beginn der Klothoide die volle Leistung gibt und den Rest der Kurve in seiner maximalen Leistung eingeschränkt wird, wird an dieser Stelle die Motion Primitive „Continuous Steering“ eingeführt. Wie der Name bereits vermuten lässt ist diese ähnlich der Klothoide, bei der der Lenkeinschlag φ linear ansteigt, im Gegensatz zu der Krümmung κ.

Aus dem Ackermannmodell (siehe Kapitel 2.1) und (Eq. 2.1) lässt sich unmittelbar der Zusammenhang zwischen Krümmung und Lenkwinkel herleiten:

$$\kappa = \frac{\tan(\varphi)}{b}$$

Nach Definition gilt für den Lenkwinkel φ im Verlauf der Zeit t:

$$\varphi(t) = v_{\text{steer}} \cdot t$$

Dabei ist v_{steer} die Lenkgeschwindigkeit. Somit gilt für den Lenkwinkel φ in Abhängigkeit von der Länge l, unter Bezug der physikalischen Formel (Eq. 4.7):

$$\varphi(l) = \frac{v_{\text{steer}}}{v} \cdot l \tag{4.9}$$

Für die Krümmung in Abhängigkeit von der Länge folgt:

$$\kappa(l) = \frac{\tan(\varphi(l))}{b} \overset{(Eq.4.9)}{=} \frac{\tan(\frac{v_{\text{steer}}}{v} \cdot l)}{b} \tag{4.10}$$

Für die Orientierung des Fahrzeugs folgt:

$$\theta(l) = \int_0^l \kappa(t)\mathrm{d}t \overset{(Eq.4.10)}{=} \frac{1}{b} \cdot \int_0^l \tan\left(\frac{v_{\text{steer}}}{v} \cdot t\right) \mathrm{d}t$$

Substituiere:

$$u = \frac{v_{\text{steer}}}{v} \cdot t \tag{4.11}$$

$$\mathrm{d}u = \frac{v_{\text{steer}}}{v} \cdot \mathrm{d}t \quad \Rightarrow \quad \mathrm{d}t = \frac{v}{v_{\text{steer}}} \cdot \mathrm{d}u \tag{4.12}$$

$$\int \tan(u)\mathrm{d}u = -\ln(\cos(u)) \tag{4.13}$$

$$\begin{aligned}
\theta(l) &= \frac{1}{b} \cdot \int_0^{\frac{v_{\text{steer}}}{v} \cdot l} \tan(u)\mathrm{d}u \cdot \frac{v}{v_{\text{steer}}} && \mid (Eq.4.11)\&(Eq.4.12) \\
&= \frac{v}{b \cdot v_{\text{steer}}} \cdot \int_0^{\frac{v_{\text{steer}}}{v} \cdot l} \tan(u)\mathrm{d}u \\
&= -\frac{v}{b \cdot v_{\text{steer}}} \cdot \ln\left(\cos\left(\frac{v_{\text{steer}}}{v} \cdot l\right)\right) && \mid (Eq.4.13)
\end{aligned}$$

Für die Änderung der Koordinaten gilt das gleiche wie für die Klothoide mit dem entsprechenden Winkel:

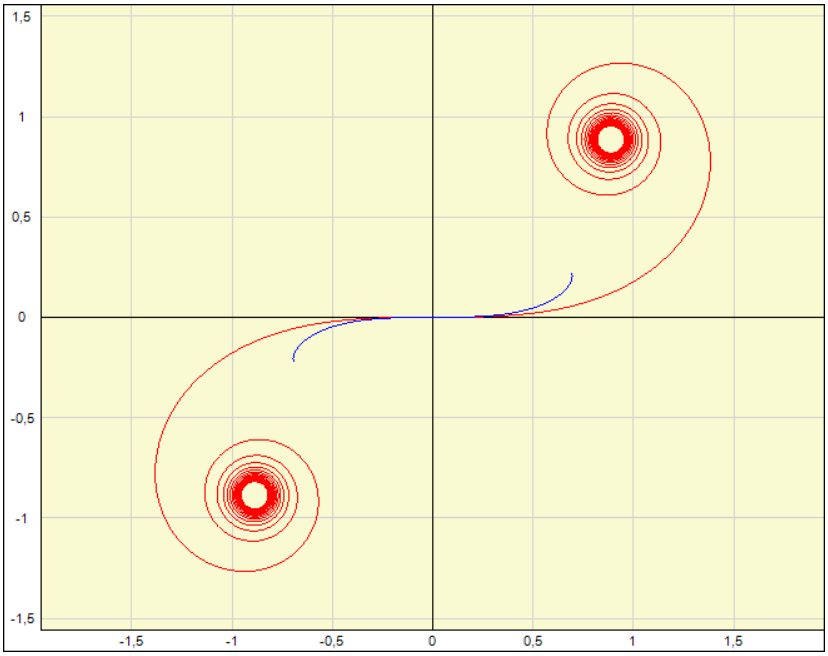
$$x(l) = \int_0^l \cos\left(-\frac{v}{b \cdot v_{\text{steer}}} \cdot \ln\left(\cos\left(\frac{v_{\text{steer}}}{v} \cdot t\right)\right)\right) \mathrm{d}t$$
$$y(l) = \int_0^l \sin\left(-\frac{v}{b \cdot v_{\text{steer}}} \cdot \ln\left(\cos\left(\frac{v_{\text{steer}}}{v} \cdot t\right)\right)\right) \mathrm{d}t$$

Diese Integrale sind leider nicht ohne weiteres lösbar und auch nicht nach den Fresnel-Integralen substituierbar. Da aber die Fresnel-Integrale auch numerisch berechnet werden, kann man die Integrale auch direkt numerisch lösen.

Da die Logarithmusfunktion nur für die positiven Zahlen definiert ist, hat auch die Continuous Steering Primitive einen eingeschränkten Definitionsbereich. Dies hat aber einen logischen Hintergrund, denn erreichen die Räder den Lenkeinschlag $\varphi = \frac{\pi}{2}$ (90°), so stehen die Räder des Fahrzeugs quer und für diesen Lenkeinschlag ist die Folgekonfiguration unbestimmt. Da aber der maximale Lenkeinschlag $\varphi_{\max}$ kleiner ist als $\frac{\pi}{2}$ muss spätestens ab dieser Stelle ein Kreisbogen als weiterer Kurvenverlauf verwendet werden. Der Definitionsbereich der Continuous Steering Primitive ist:

$$\mathbb{D} = (-\frac{v\pi}{2v_{\text{steer}}}; \frac{v\pi}{2v_{\text{steer}}})$$

Der größte Vorteil dieser Primitive ist, dass sie die Fahrzeugeinschränkungen bis ans Limit ausnutzt und somit eine sehr enge Kurve fährt, ohne die fahrdynamischen Einschränkungen zu verletzen. Eine Klothoide, die mit Krümmungsänderung $\sigma = 1$, Achsenabstand $b = 2$ (engl. wheelbase) und Geschwindigkeit $v = 1$ berechnet wurde, hat als größte Lenkgeschwindigkeit $v_{\text{steer,max}} = 2$ nach (Eq. 4.8) nur zu Beginn der Kurve. Berechnet man die Primitive Continuous Steering mit den gleichen Parametern, so verbraucht sie dramatisch weniger Platz, um den gleichen Lenkeinschlag zu erreichen (Abb. 4.7).



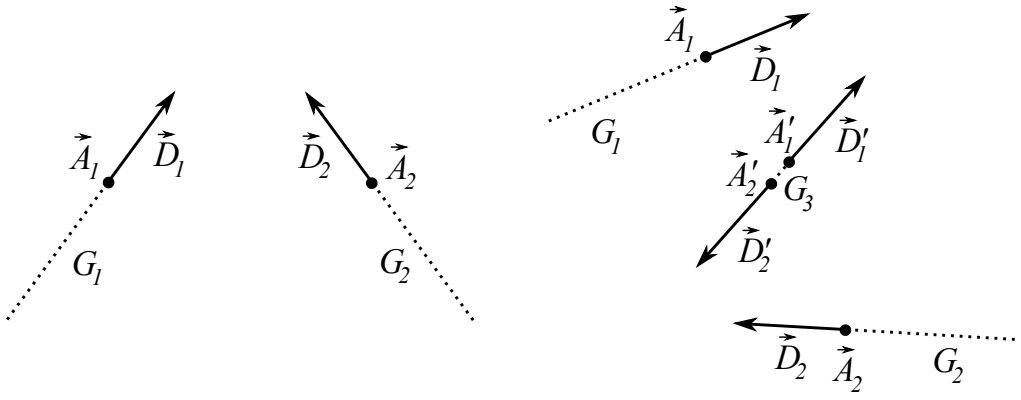
(Abb. 4.7): *Graph einer Continuous Steering Primitive (blau) mit* $v_{\text{steer}} = 2$, $v = 1$ *und* $l = \frac{\pi}{4}$, *im Vergleich zu einer Klothoide (rot) mit gleichen Fahrzeugparametern.*

4.5 VERBINDUNGEN

Es wurden viele verschiedene Primitiven vorgestellt und auch deren mathematischer Hintergrund. Es bleibt aber trotzdem keine triviale Aufgabe zwei Geraden über Klothoiden oder Continuous Steering Primitiven zu verbinden. Es gibt keine genauen Berechnungsverfahren, sondern einige wenige iterative Algorithmen. Zuallererst muss gewährleistet sein, dass sich die zu verbindenden Geraden an den Verlängerungen der Übergangspunkte schneiden. Seien die Geraden G_i definiert über den Austrittsrichtungsvektor $\vec{D_i}$ und den Stützvektor $\vec{A_i}$.

$$G_1 : \vec{X} = \vec{A_1} + \lambda_1 \cdot \vec{D_1}$$
$$G_2 : \vec{X} = \vec{A_2} + \lambda_2 \cdot \vec{D_2}$$

So muss für den Schnittpunkt, der durch λ_i definiert wird, gelten $\forall i : \lambda_i > 0$. Die Geraden müssen sich also voreinander schneiden (Abb. 4.8 a). Ist dies nicht gegeben, so muss eine Stützgerade gefunden werden, die die Geraden über eine Wendekurve verbindet und zu beiden Seiten die vorgegebenen Eigenschaften erfüllt. Der Algorithmus muss dann zu beiden Seiten ausgeführt werden (Abb. 4.8 b).



(a) Geraden schneiden sich direkt voreinander.

(b) Die Stützgerade G_3 schneidet beide Geraden nach Vorgabe.

(Abb. 4.8): *Verbindungen zwischen Geraden.*

Zuerst wird nun der Schnittpunkt beider Geraden berechnet und der längere der beiden Schenkel wird der Länge des kürzeren angepasst, indem die Gerade um die Differenz der beiden Schenkellängen verlängert wird. Es soll eine an der Winkelhalbierenden gespiegelte Kurve entstehen. Die Winkelhalbierende spielt im nächsten Schritt eine wichtige Rolle. Ausgehend von einem der Punkte $\vec{A}_i$ wird angefangen eine Continuous Steering Primitive mit maximaler Lenkgeschwindigkeit iterativ zu berechnen. In jedem Schritt lässt sich der Radius der Kurve berechnen und somit der Kreismittelpunkt $\vec{C_\Omega}$. Dieser Punkt startet in der Unendlichkeit und wandert mit zunehmendem Lenkeinschlag in Richtung der Winkelhalbierenden. Sobald dieser die Winkelhalbierende erreicht hat, hat man den richtigen Lenkeinschlag für einen Kreisbogen, den man ab dieser Stelle mit dem letzten Radius ansetzt. Das selbe gilt spiegelverkehrt für die andere Seite.

4.6 Ergebnisse und Evaluation

Es wurden mehrere Primitiven vorgestellt, aus denen sich eine Bahn zusammensetzen lässt. Nur aus Geraden kann natürlich keine Bahn aufgebaut werden, da das Fahrzeug nicht auf der Stelle wenden kann. Es ist auch nicht vorteilhaft nur Kreisbögen und Geraden zu nutzen, denn es wurde gezeigt, dass aufgrund des nicht ruckfreien Übergangs der Regler nicht damit zurechtkommt diese auf Geschwindigkeit zu durchfahren. Man hat die Wahl zwischen Klothoiden und Continous Steering Primitiven. Es wurde gezeigt, dass die Continuous Steering Primitiven viel besser die kinematischen Einschränkungen des Fahrzeugs ausnutzen, wodurch diese viel weniger Platz in Anspruch nehmen, um die gleichen Orientierungsänderungen des Fahrzeugs zu erreichen. Da die Primitiven nur einmal vor der Suche berechnet werden und für die Suche nur die Matrix der Endkonfiguration gebraucht wird und deren Komplexität nicht durch die Komplexität der Berechnung der Primitive ansteigt, kann beruhigt die Continuous Steering Primitive benutzt werden.

5 SAMPLING

Sampling beschäftigt sich mit der automatischen Generierung von Motion Primitiven. Während Control Space Sampling versucht anhand der Fahrzeugparameter zwischen den Extremen verschiedene meist gleichlange Segmente zu generieren, wird beim State Space Sampling der Raum in endlich viele gültige Konfigurationen eingeteilt, die angefahren werden können. In den folgenden Kapiteln werden diese Verfahren und einige Beispiele dazu vorgestellt.

5.1 CONTROL SPACE

Beim Sampling in Control Space geht es darum anhand von einem oder mehreren physikalischen Einschränkungen (engl. constaints) des Fahrzeugs Motion Primitiven ein mal offline vorzuberechnen, so dass die Berechnung während der Suche entfällt. Im folgenden werden zwei Möglichkeiten vorgestellt dies zu realisieren.

5.1.1 ARC-BASED SEARCH SPACE

Beim Arc-Based search space wird der Suchraum, wie der Name vermuten lässt, in Kreisbögen eingeteilt. Dies geschieht nur anhand der Fahrzeugeigenschaft Lenkwinkel, die durch den maximalen Lenkwinkel beschränkt wird. Im Normalfall entstehen dabei $2n+1$, $n \in \mathbb{N}^+$ Motion Primitiven, mit einer Geraden und n Bogensegmenten pro Lenkrichtung mit verschiedenen Lenkeinschlägen. Idealerweise hält man alle Motion Primitiven gleich lang. Für einen Radius $r_{\mathcal{M}}$, der direkt aus dem Lenkeinschlag φ folgt, kann die Länge $l_{\mathcal{M}}$ des Kreisbogens folgendermassen berechnet werden:

$$l_{\mathcal{M}} = \alpha \cdot r_{\mathcal{M}}$$

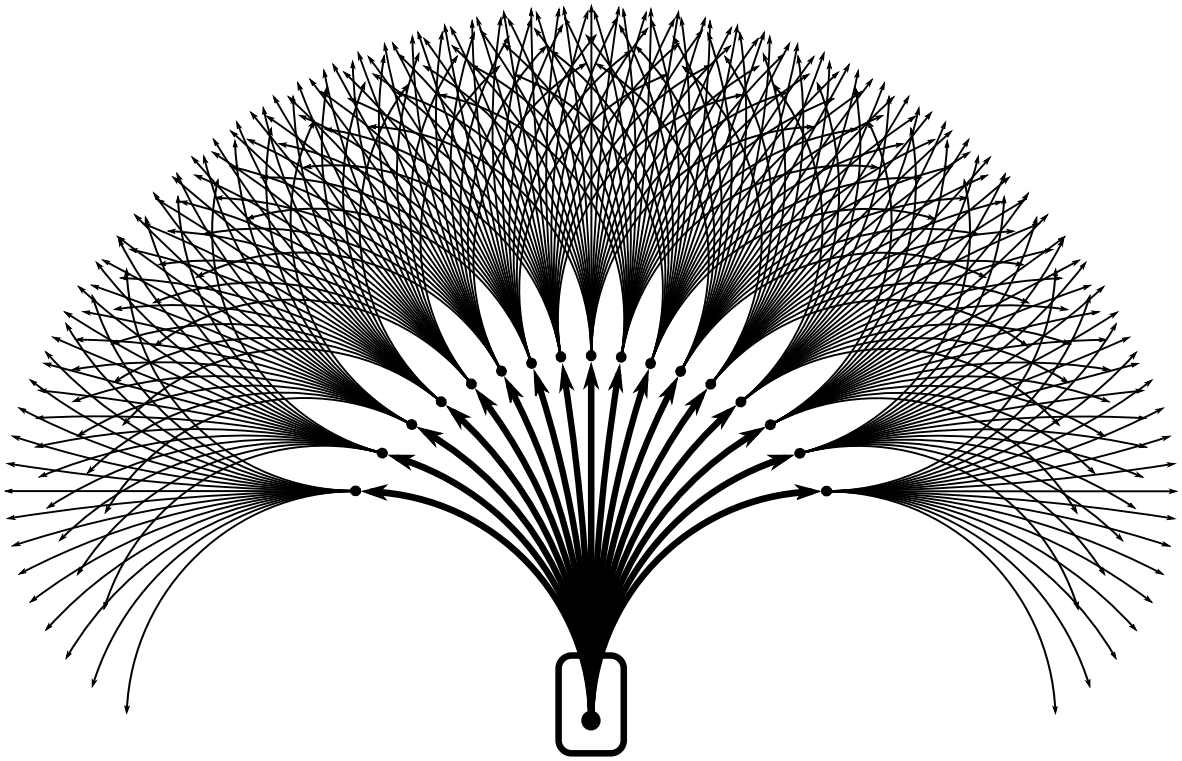
Dabei ist α der innere Winkel des Kreisbogens in Radians (siehe Kapitel 4.2). Für die verschiedenen Radien $r_{\mathcal{M},i}$ der Kreisbögen gilt:

$$r_{\mathcal{M},i} = \frac{b}{\tan\left(\frac{i}{n} \cdot \varphi_{\max}\right)} \qquad | \; i \in \mathbb{Z}, n \in \mathbb{N}^+, -n \leq i \leq n, i \neq 0$$

Entsprechend berechnen sich die inneren Winkel α_i der Kreisbögen:

$$\alpha_i = \frac{l_{\mathcal{M}}}{r_{\mathcal{M},i}}$$

Mithilfe von α_i und $r_{\mathcal{M},i}$ können anhand der Formel für x_α und y_α (siehe Kapitel 4.2) die Endkonfigurationen der Kreisbögen berechnet werden. Hinzu kommt eine Gerade der Länge $l_\mathcal{M}$ (Abb. 5.1).



(Abb. 5.1): *Control Space Sampling: Arc-Based search space der Tiefe 2 und* $n = 8$.

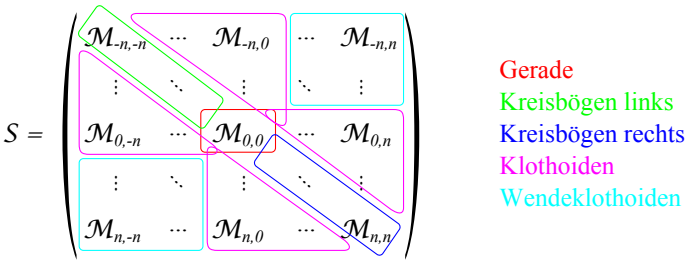
Wie bereits im Kapitel über Kreisbögen (Kapitel 4.2) angesprochen, eignen sich diese nicht besonders gut zum Befahren, aufgrund ihrer nicht ruckfreien Übergänge. Beim Übergang zwischen dem maximalen Lenkeinschlag in die eine Richtung zu dem maximalen Lenkeinschlag in die andere Richtung, driftet das Fahrzeug sehr weit aus der Kurve. Die Alternative wären Haltepunkte an den Übergängen, um den Lenkwinkel einzustellen, was jedoch dem flüssigen Fahrverhalten widerspricht.

5.1.2 ADAPTIVE SEARCH SPACE

Bei dem Adaptive search space wird zu den Lenkwinkeln, wie in Arc-Based search space, auch die Lenkgeschwindigkeit v_{steer} hinzugenommen. Es entstehen Klothoiden (siehe Kapitel 4.3) oder Continuos Steering Primitiven (siehe Kapitel 4.4). Hinzu kommt, dass man die Start- und Endlenkwinkel diskretisiert, so dass man anhand des Endleknwinkels, nur die Motion Primitiven mit dem gleichen Startlenkwinkel als Kindkonfigurationen bekommt, um für einen ruckfreien Übergang zu sorgen. Motion Primitiven, deren Start- als auch Endlenkwinkel gleich sind, sind entweder Geraden beim Lenkwinkel $\varphi = 0$ oder Kreisbögen. Eine Möglichkeit die Primitiven leicht vorzuberechnen und diese anschliessend auszuwählen, ist diese in eine $2n + 1 \times 2n + 1$ Matrix zu schreiben, wobei die Zeilen dem diskreten Startlenkwinkel und die Spalten dem diskreten Endlenkwinkel entsprechen. Anhand des Endlenkwinkels einer Motion Primitive wird die selbe Zeile der Matrix als Folgeprimitiven ausgewählt. Sei der Suchraum (engl. search space) $\mathcal{S}$ folgendermassen definiert:

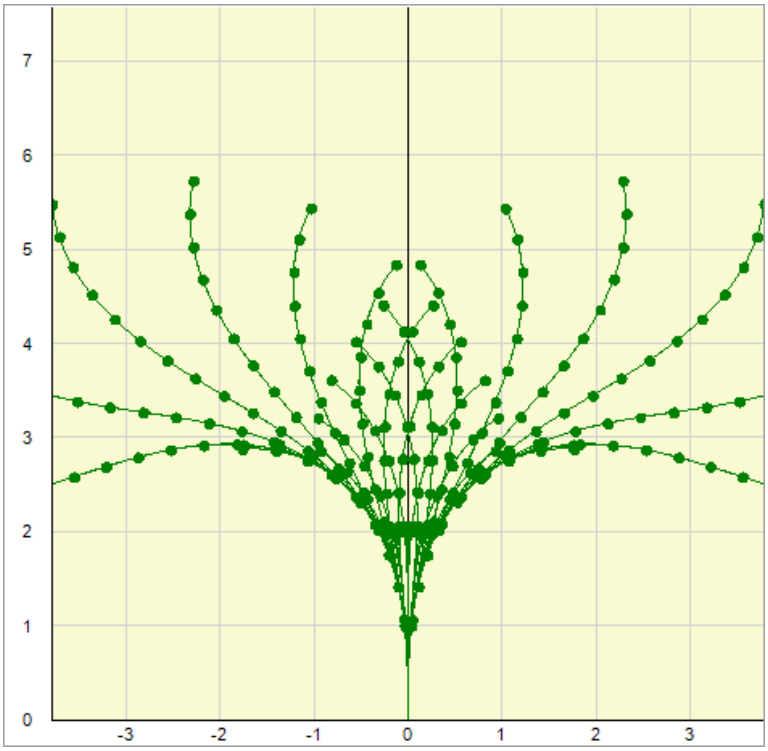
$$\mathcal{S} = \{\mathcal{M}_{i,j} \mid -n \leq i,j \leq n; n \in \mathbb{N}^+; i,j \in \mathbb{Z}\} \qquad \mid \mathcal{S} \in M^{2n+1 \times 2n+1}$$

Der Suchraum ist also eine $2n+1 \times 2n+1$ Matrix, deren Elemente aus Übersichtlichkeitsgründen von $-n$ bis n indiziert sind. Die Matrix hat die folgende Form und Elemente (Abb. 5.2):



(Abb. 5.2): *Suchraummatrix $\mathcal{S}$ des Adaptive search space und deren Elemente.*

Anstelle der Klothoiden können auch Continuous Steering Primitiven benutzt werden.



(Abb. 5.3): *Control Space Sampling: Adaptive search space der Tiefe 2, Länge $l = 1$ und $n = 8$.*

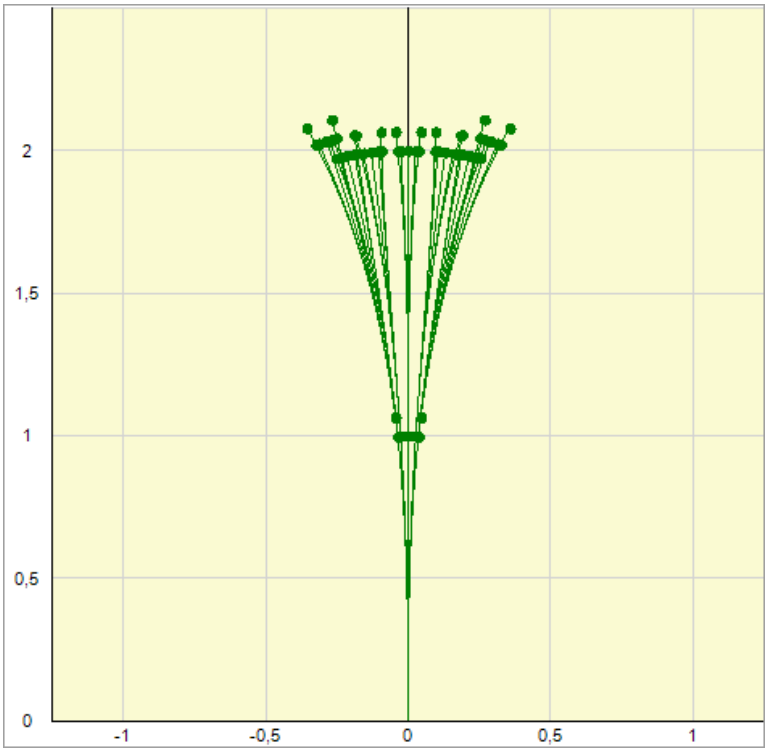
In dem Bild (Abb. 5.3) kann man die Wendekurven an den Seiten ernkennen. Man erkennt aber auch einen Nachteil der naiven Berechnung des gesamten Suchraums, denn es entstehen redundante Primitiven verschiedener Längen. Der Grund dafür ist die maximale Lenkgeschwindigkeit des Fahrzeugs und die diskreten Endlenkwinkel die eingestellt werden müssen. Da die minimale Länge $l_{\min}$ der Primitiven vorgegeben wird, errechnet sich die benötigte Lenkgeschwindigkeit v_{steer}:

$$v_{\text{steer}} = \frac{\Delta\varphi \cdot v}{l_{\min}}$$

Dabei ist $\Delta\varphi$ die Differenz zwischen End- und Startlenkwinkel. Wenn $\Delta\varphi$ aber zu groß wird, so wird die Lenkgeschwindigkeit v_{steer} größer als die maximale Lenkgeschwindigkeit $v_{\text{steer,max}}$, was physikalisch nicht möglich ist, so muss die Länge l der Bahn verlängert werden, so dass mit maximaler Lenkgeschwindigkeit die gewünschte Lenkwinkeldifferenz $\Delta\varphi$ durchgeführt werden kann.

$$l = \frac{\Delta\varphi \cdot v}{v_{\text{steer}}}$$

Dies hat zur Folge, dass die Primitiven alle mit der gleichen maximalen Lenkgeschwindigkeit $v_{\text{steer,max}}$ berechnet werden, sich aber nur anhand der Länge unterscheiden, um den gewünschten Endlenkwinkel φ zu erreichen. Diese redundanten Primitiven sind überflüssig und stören nur den Suchalgorithmus (siehe Kapitel 6), das pro Konfiguration zu viele Folgekonfigurationen zur Auswahl stehen. Entfernt man die Redundanzen, so reduziert sich die Anzahl der Folgekonfigurationen enorm (Abb. 5.4).



(Abb. 5.4): *Control Space Sampling: Adaptive search space der Tiefe 2, Länge $l = 1$ und $n = 8$, mit entfernten Redundanzen.*

5.2 State Space

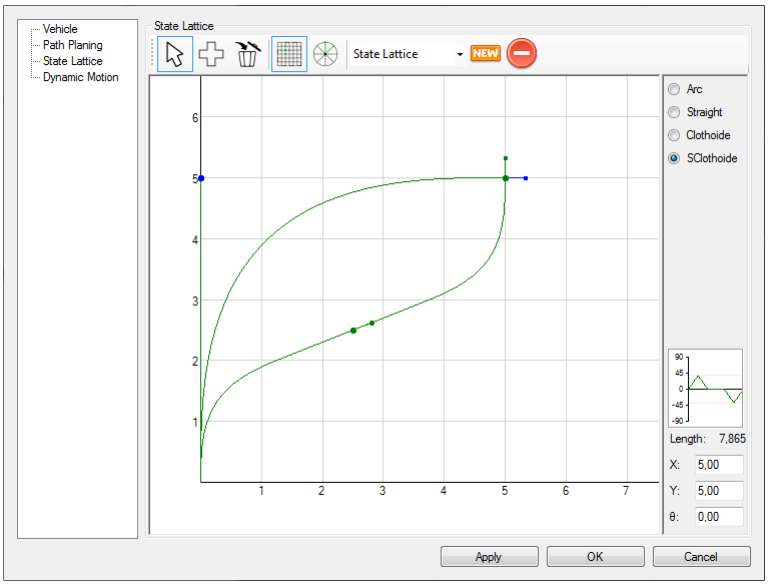
In der Literatur wird meist von dynamic State gesprochen, zu dem neben dem Zustand des Fahrzeugs (x, y, θ, φ) auch die momentanen Geschwindigkeiten zählen. Im Verlauf dieser Arbeit wird State wie

Konfiguration des Fahrzeugs, also seine Pose betrachtet. Beim State Space Sampling wird versucht möglichst bestimmte Konfigurationen im Raum anzufahren, wobei auch hier die kinematischen Eigenschaften des Fahrzeugs nicht verletzt werden dürfen. Dies kann entweder durch Diskretisierung von Raumpunkten sein, welches bei State Lattice (Kapitel 5.2.1) verfolgt wird oder indem die Konfigurationen an die Umgebung angepasst werden. Wenn ein Fahrzeug z. B. einer Fahrspur folgt, sind nur Konfigurationen interessant, die in Fahrtrichtung auf der Fahrspur und parallel dazu auf der gesamten Fahrbahnbreite liegen. Bei diesem Verfahren können aber die Motion Primitiven nicht vorberechnet werden und müssen online zur Laufzeit bestimmt werden (Kapitel 5.2.2).

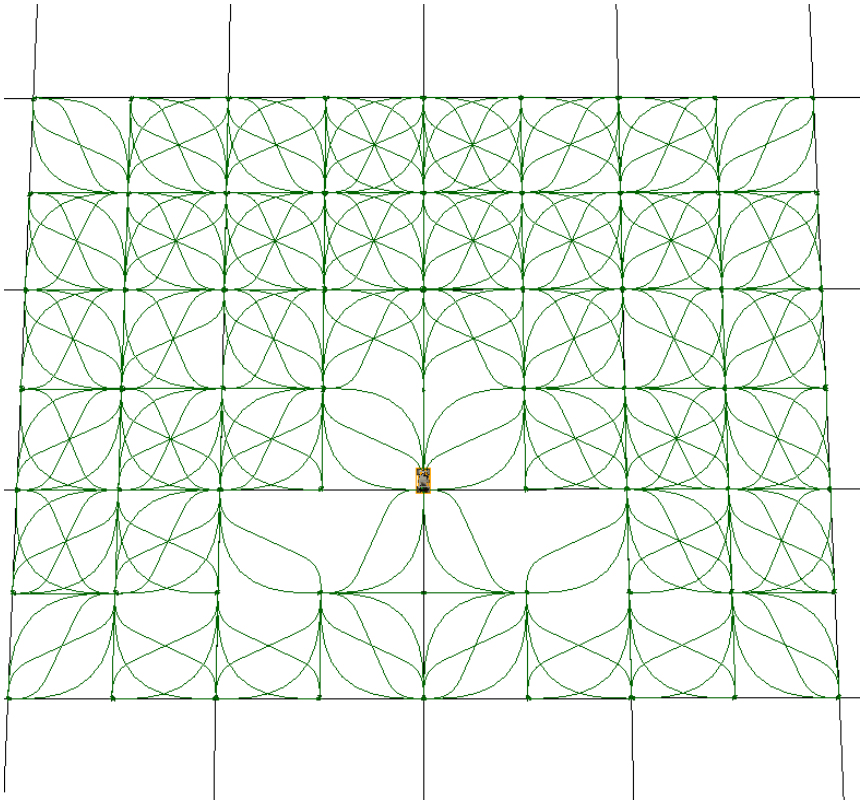
5.2.1 State Lattice

Schaut man sich noch einmal das Bild des Arc-Based search space (Abb. 5.1) an, so erkennt man dass bereits in der zweiten Stufe sehr viele unterschiedliche Endkonfigurationen entstehen. Es gibt sehr viele Endpunkte, die eng im Raum verstreut sind und unterschiedliche Orientierungen haben. Aus Sicht der Suchalgorithmen sind alles einzelne Knoten, die in Listen gespeichert werden, wodurch die Listen unnötig anwachsen und die Auswahl des kostengünstigsten Elements länger dauert (genauer in dem Kapitel 6 Suchalgorithmen). Ausserdem entstehen dabei sehr viele individuelle Bahnen, die fast nie wieder an einer anderen Stelle zusammenfinden, wodurch bereits gefundene Bahnen, beim Umplanen (z. B. mit D*) nicht wiederverwendet werden können, weil einfach die neu gefundenen Bahnen eng an der bereits gefundenen Bahn vorbeiführen, diese aber nie an bekannten Konfigurationen kreuzen.

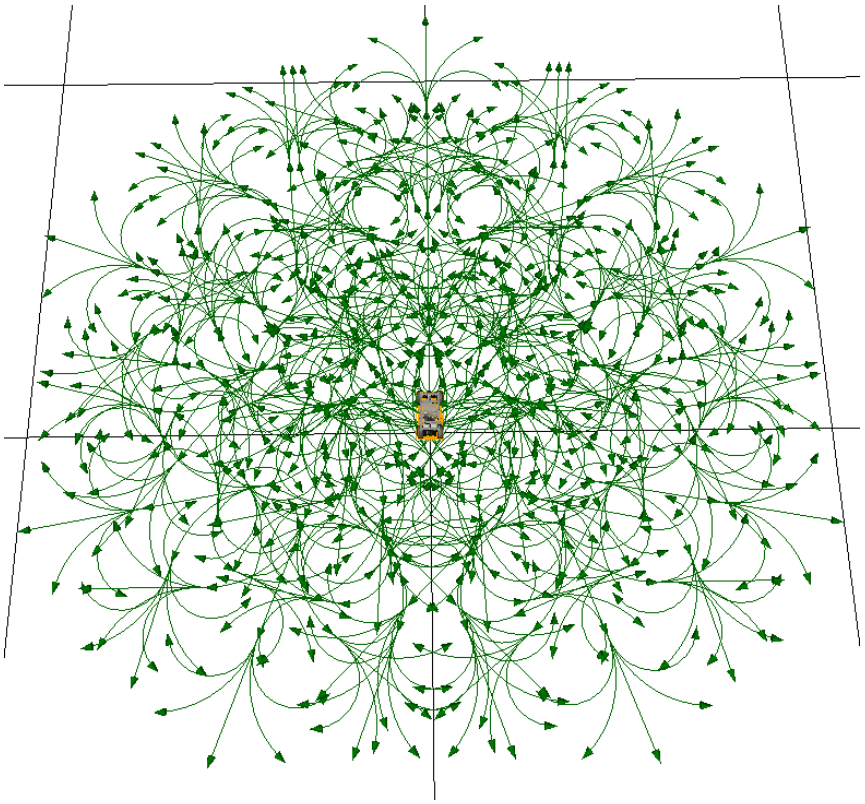
Eine gute Alternative dazu bietet die State Lattice, die den Raum in diskrete Punkte, die angefahren werden können, wie ein Grid einteilt. Die Vorlage einer State Lattice wird vom Benutzer erstellt, der darauf achten soll, dass ein möglichst homogener Baum entsteht, der den Raum möglichst gut abdeckt und sich wiederholt. Ein solches Werkzeug wurde im Rahmen dieser Arbeit erstellt (Abb. 5.5).



(Abb. 5.5): *State Lattice Editor. Hier typische Primitiven: Gerade, 90°-Kurve und Wendeklothoide.*



(a) Erreichbarkeitsbaum mit State Lattice aus (Abb. 5.5)



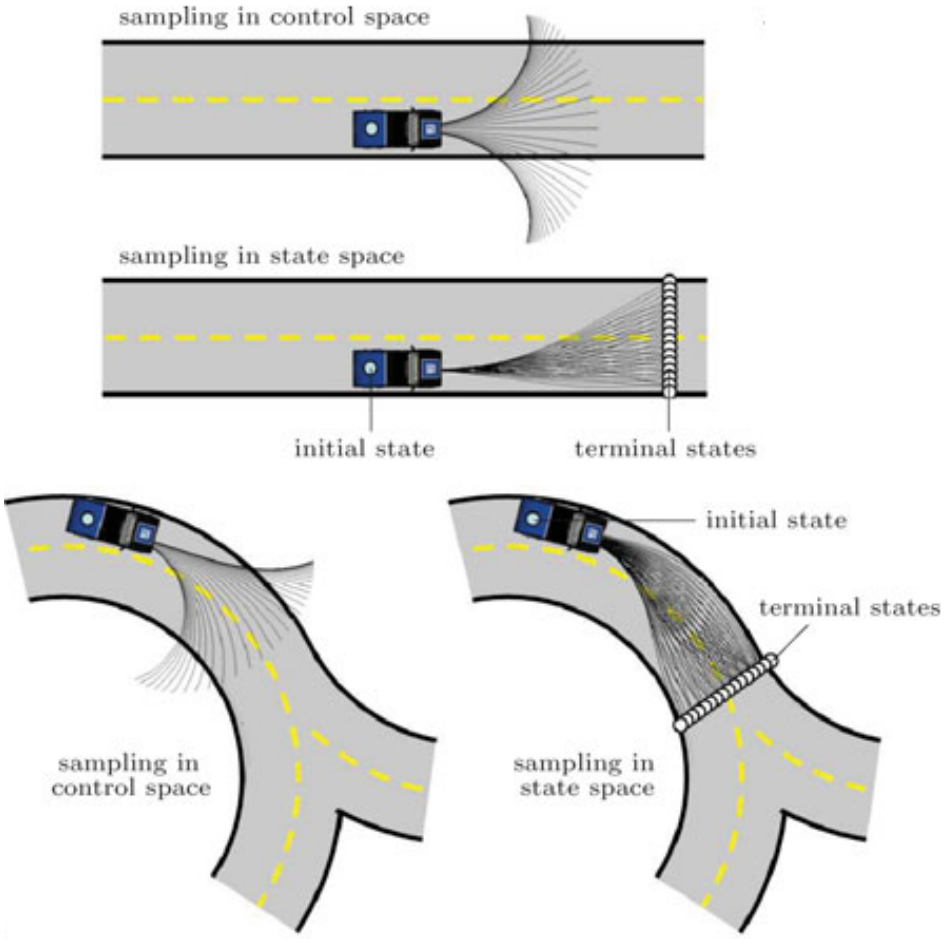
(b) Erreichbarkeitsbaum mit Adaptive Search Space.

(Abb. 5.6): *Vergleich der Erreichbarkeitsbäume der Tiefe 4 und 5 Nachfolgeelemente.*

Vergleicht man die Erreichbarkeitsbäume (engl. reachability tree) der State Lattice mit Adaptive Search Space, so fällt beim State Lattice direkt die Homogenität des Baumes auf, der sich ständig wiederholt und Punkte über mehrere Möglichkeiten angefahren werden können. Bei Adaptive Search Space entstehen sehr viele eng verstreute Konfigurationen, die alle individuell angefahren werden müssen, so dass Wiederholungen nicht ausgenutzt werden (Abb. 5.6).

5.2.2 Online Sampling

Dieses Verfahren wird angewandt, um die Motion Primitiven an die gegebene Umgebung anzupassen. Da die Umgebung stark variabel sein kann, wie z. B. Fahren auf einer Landstrasse, die beliebig komplex werden kann, ist das vorangegangene Beispiel mit State Lattices nicht ideal, da man nur diskrete Raumpunkte anfährt. Auch das Adaptive Search Space liefert aus einer Vielzahl an Möglichkeiten nur eine sinnvolle, wo der Rest von der Bahn führt und somit nicht zu gebrauchen ist. Da aber die Fahrbahn beliebig breit sein kann und auch mehrere Spuren haben kann, können nur Primitiven berechnet werden, die auf der Fahrbahn in Fahrtrichtung bleiben.



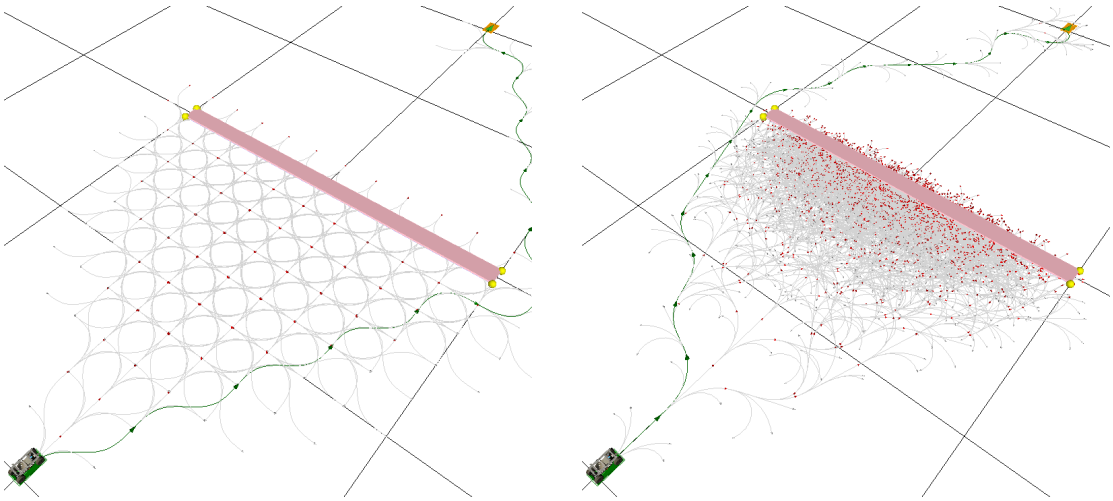
(Abb. 5.7): *Vergleich Control Space Sampling mit Online State Space Sampling.* ***Quelle:*** *Howard et al. 2008 [14]*

Dieses Verfahren wurde im Paper (Howard et al. 2008 [14]) aufgezeigt. Man erkennt, dass das Control Space nur mit wenigen Primitiven auf dem Weg bleiben würde, wo sogar die meisten spätestens im nächsten Schritt von der Fahrbahn abkommen würden (Abb. 5.7). Das Online State Space Sampling liefert aber nur gültige Primitiven, die alle auf der Fahrbahn bleiben und auch deren Folgekkonfigurationen gültig wären. Da diese Primitiven meist zur Laufzeit berechnen werden müssen, empfiehlt es sich schnell berechenbare Motion Primitiven zu benutzen, oder auch die Anzahl der Kinderprimitiven einzugrenzen auf Fahrspurhalten oder -wechseln.

Dieses Verfahren eignet sich natürlich eher für Gebiete, deren Umgebung und Fahrbahnen bekannt sind. Da aber die Umgebung im Rahmen dieser Arbeit auf statische Umgebungen ohne Fahrspuren ausgelegt ist, wurde dieses Verfahren nicht implementiert und getestet.

5.3 Ergebnisse und Evaluation

Alle Samplingmethoden haben ihre Vor- und Nachteile. Als größter Vorteil des State Space Sampling sei die Homogenität in den Erreichbarkeitsbäumen zu benennen, aufgrund derer die Suche reduziert und vereinfacht werden kann (vgl. folgendes Szenario Abb. 5.8). Diese sind auch für eine dynamische Umgebung vorteilhaft, weil so schneller umgeplant werden kann, denn es muss nur auf den bereits gefundenen Pfad zurückgefunden werden und der restliche Weg bleibt gleich. Aufgrund der diskreten Anfahrtspunkte des State Space Samplings sehen die entstandenen Pfade meist nicht aus wie ein Mensch fahren würde, denn es sieht teilweise so aus, als versuche das autonome Fahrzeug sich an den Eckpunkten eines gedachten Grids zu halten.



(a) State Space Sampling. **(b)** Adaptive Space Sampling.

(Abb. 5.8): *Szenario für die Evaluation der Samplingmethoden. Beide Verfahren haben 5 Nachfolgekonfigurationen und die Länge der Primitiven beträgt etwa 3 bis 4 Meter.*

	State Space Sampling	**Adaptive Space Sampling**
Weglänge in $[m]$	51,45	51,00
Abstand zum Ziel $[m]$	0,00	0,29
Berechnungszeit $[ms]$	24	534
Expandierte Knoten	132	796
Knoten auf dem Heap	655	3 975
Knoten in Open Set	119	1 378
Knoten in Closed Set	178	2 577

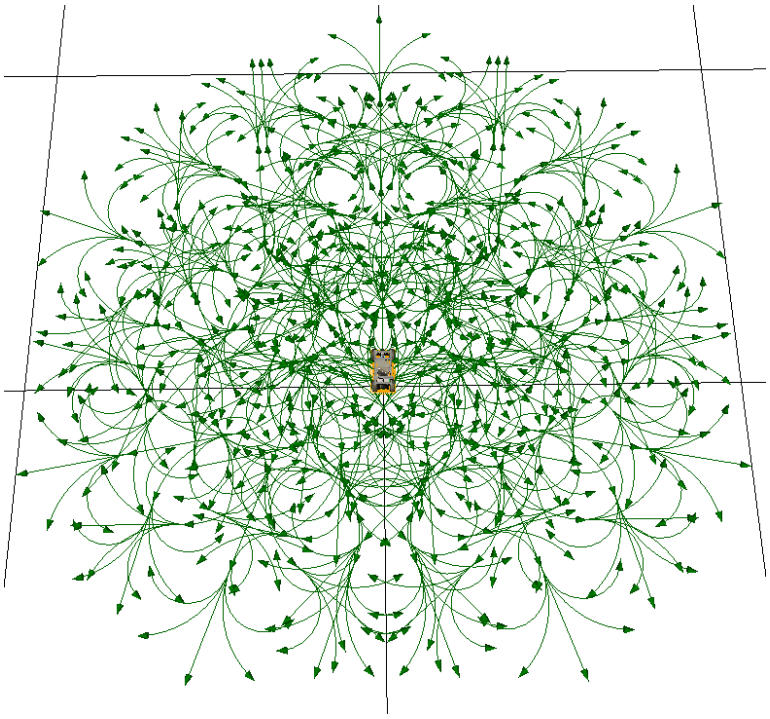
(Tab. 5.1): *Messdaten zu (Abb. 5.8).*

Anhand der Tabelle (Tab. 5.1) kann man erkennen, dass der gleiche Algorithmus (siehe Kapitel 6) mit Primitiven des State Space Sampling nahezu 20 mal schneller ist, als der Algorithmus mit Adaptive Space Sampling und auch weniger Ressourcen verbraucht. Schaut man sich die vom Suchalgorithmus aufgebauten Erreichbarkeitsbäume an, so erkennt man schnell den Grund für diesen Umstand (Abb. 5.8).

6

SUCHALGORITHMEN

Bisher wurden Motion Primitiven eingeführt, die von einem Fahrzeug mit Achsschenkellenkung abgefahren werden können. Weiterhin wurden Samplingmethoden vorgestellt, um für das Fahrzeug aus verschiedenen Konfigurationen die möglichen Motion Primitiven zu berechnen, woraus ein Erreichbarkeitsbaum entsteht. Wenn man sich noch einmal das Bild des Adaptive Search Space anschaut, so erkennt man bereits bei einer Suchtiefe von 4 und Primitivenanzahl von 5, dass der Baum viele Knoten besitzt und den Raum gut abdeckt, jedoch noch nicht so Raumgreifend ist (Abb. 6.1).



(Abb. 6.1): *Erreichbarkeitsbaum von Adaptive Search Space mit Tiefe 4 und 5 Nachfolgeelementen.*

Bereits dieser Erreichbarkeitsbaum hat $5 + 5^2 + 5^3 + 5^4 = 750$ Elemente. Man erkennt, dass dieser mit der Tiefe exponentiell wächst, was bereits nach geringen Suchtiefen den Rahmen für die Möglichkeiten eines PCs sprengen würde. Die Formel ist eine geometrische Reihe, für die allgemein gilt:

$$S_n = \sum_{i=0}^{n} q^i = 1 + q^1 + q^2 + \ldots + q^{n-1} + q^n = \frac{q^{n+1} - 1}{q - 1} \tag{6.1}$$

In diesem Fall wäre n die Tiefe des Erreichbarkeitsbaumes und q die Anzahl der Nachfolger. Da diese aber mit 1 beginnt und der Rest mit dem Beispiel übereinstimmt, muss die 1 wieder subtrahiert werden, um auf die Formel für die Anzahl Knoten in einem Erreichbarkeitsbaum zu kommen.

Unter einer Annahme, dass die Motion Primitiven eine Länge von 1 Meter haben (meistens sogar noch kürzer), kann man bei einem Erreichbarkeitsbaum der Tiefe 15, gerade einmal maximal 15 Meter radial vom Startpunkt aus entfernt sein. Berechnet man die Ressourcen, die für so einen Baum mit 5 Nachfolgeelementen benötigt werden:

$$S_{15} = -1 + \sum_{i=0}^{15} 5^i = \frac{5^{16} - 1}{5 - 1} - 1 = 38.146.972.655$$

Unter der Annahme, dass die Motion Primitiven mindestens die Start- und die Endkonfiguration besitzen, die aus je 3 Werten bestehen à 4 Byte, so hat jede Primitive im Speicher mindestens 24 Byte. Das bedeutet, dass diese Primitiven $38.146.972.655 \cdot 24\mathrm{Byte} = 915.527.343.720\mathrm{Byte} \approx 852,65\mathrm{GByte}$ Arbeitsspeicher in Anspruch nehmen. Selbst unter den heutigen Bedingungen, wo 8 GByte Arbeitsspeicher fast Standard ist, sprengt diese simple Rechnung den Rahmen, ganz zu schweigen von der Kalkulationszeit für so viele Elemente.

In diesem Kapitel werden verschiedene Suchalgorithmen vorgestellt und deren Parametriesierbarkeit, um das genannte Problem zu minimieren.

6.1 A*

A* ist eine heuristische Suche, die darauf arbeitet, dass sowohl die Start- als auch die Endkonfiguration bekannt sind und die Güte von Zwischenkonfigurationen heuristisch geschätzt und in Relation gesetzt werden kann. Dies bedeutet, dass berechnete Konfigurationen eine Information darüber liefern können, ob man in die richtige Richtung sucht. Im Normalfall ist es die Position der Konfigurationen, aus denen die Entfernung zum Endpunkt berechnet werden kann. Gleichzeitig spielt auch die bereits zurückgelegte Distanz bis zu dieser Konfiguration eine Rolle. Aus einer Liste von mehreren Konfigurationen wählt A* immer die Konfiguration mit den geringsten Kosten $f(x)$, die sich berechnen lässt über die Summe der Heuristikfunktion $h(x,e)$, die die Kosten bis zur Endkonfiguration e schätzt und der Kostenfunktion $g(s,x)$ für die bisher zurückgelegte Distanz als Kosten von der Startkonfiguration s.

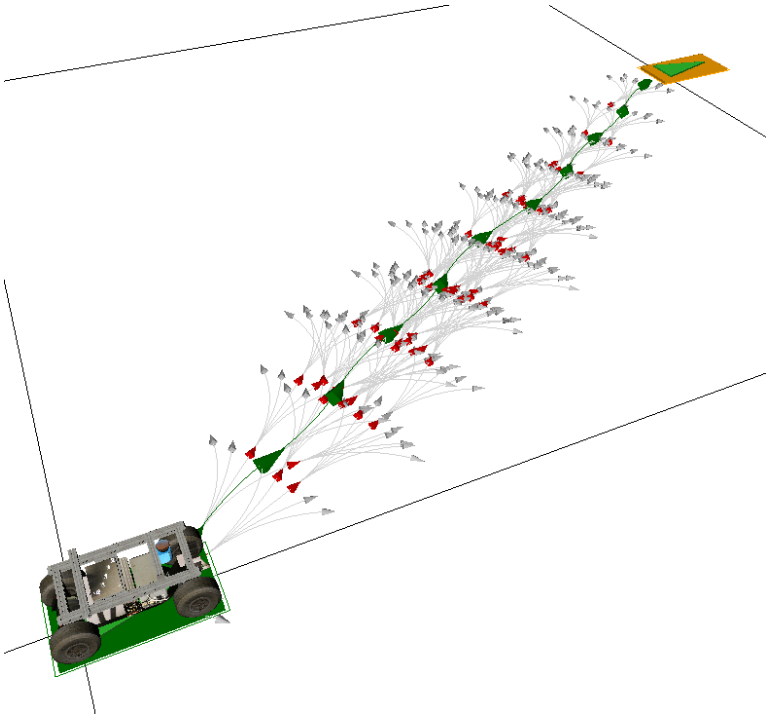
$$f(x) = g(s,x) + h(x,e)$$

Solange die Heuristikfunktion die realen Kosten nicht überschätzt, ist A* optimal (Hart, Nilsson & Raphael 1968 [9]).

$$\forall x,y : h(x,y) \leq g(x,y)$$

Auf genauere Auswirkungen und Variationen der heuristischen Funktion wird im nächsten Kapitel (6.2) eingegangen. An dieser Stelle sei die Heuristikfunktion die euklidische Distanz, die für alle Konfigurationen kleiner oder gleich der tatsächlichen Distanz ist und somit A* optimal ist.

A* operiert auf zwei Listen, die zu Beginn beide leer sind und in die offene Liste (engl. open set) kommt die Startkonfiguration. Das Open Set stellt die Menge aller noch möglichen Konfigurationen dar, die noch nicht untersucht wurden und aus denen das kostengünstigste Element ausgewählt wird. Ist das Open Set leer, so wurden alle Möglichkeiten untersucht und es existiert kein Pfad zur Endkonfiguration. Dieser Fall ist sehr unwahrscheinlich, da in der Einleitung bereits erklärt wurde, dass die Anzahl der Konfigurationen exponentiell wächst und schnell den Rahmen sprengt, es ist nur möglich wenn der befahrbare Bereich stark eingeschränkt ist und keine Lücken aufweist. In der geschlossenen Liste (engl. closed set) befinden sich Elemente, die bereits untersucht wurden oder Elemente, wo Kollisionen mit der Umgebung erkannt wurden, so dass diese nicht mehr untersuch werden. Dadurch dass immer das kostengünstigste Element aus dem Open Set ausgewählt wird, breitet sich A* zielgerichtet aus. Da aber die Anzahl der Nachfolgekonfigurationen begrenzt ist und diese nicht immer perfekt in Zielrichtung zeigen, untersucht A* mit euklidischer Distanz als Heuristikfunktion auch nahegelegene Nachbarn, da diese oft Kostengünstiger wegen den Kurven werden (Abb. 6.2).



(Abb. 6.2): *A* mit euklidischem Abstand als Heuristikfunktion. Grau (open set), rot (closed set), grün (Pfad)*

Der Algorithmus (6.1) zeigt auf, wie der A*-Algorithmus arbeitet. Dazu sei angemerkt, dass die Methode `public Node GetNode(List<Node> set, Node node)` den Knoten zurückliefert, der nach Parametereinstellungen nah genug am übergebenen Knoten ist und nahezu die gleiche Orientierung hat. Aufgrund mathematischer Ungenauigkeit und Fehlerfortpflanzung beim Rechnen werden so ähnliche Knoten unifiziert. Wird kein ähnlicher Knoten in der Menge gefunden, so wird `null` geliefert.

```
public List<Node> AStar(Node start, Node goal)
{
    // Create two empty sets
    List<Node> closedSet = new List<Node>();
    List<Node> openSet = new List<Node>();

    // Add start to open set
    start.G = 0;
    start.H = Heuristic(start, goal);
    start.F = start.G + start.H;
    openSet.Add(start);

    while (!openSet.IsEmpty)
    {
        // Find the best score node
        Node current = FindLowest(openSet);
        if (Distance(current, goal) <= minDistance)
            // Current node is close enough to the goal -> found a way
            return ReconstructPath(current);

        openSet.Remove(current);    // Remove current from open set
        closedSet.Add(current);     // And add it to the closed set

        // Create neighbors for current and calculate scores
        foreach (Node neighbor in current.Neighbors)
        {
            if (GetNode(closedSet, neighbor) != null)
                // Neighbor is already in closed set, continue with next
                continue;

            if (checkCollision(neighbor))
            {
                // Neighbor collides with the environment, add it
                // directly to closed set and continue with next
                closedSet.Add(neighbor);
                continue;
            }

            // Calculate the scores
            neighbor.G = Score(current, neighbor);
            neighbor.H = Heuristic(neighbor, end);
            neighbor.F = neighbor.G + neighbor.H;

            // Try to find other node, that's near enough to this node
            // and has almost the same orientation, so it can be
            // handled like this neighbor node
```

```
47              Node otherWay = GetNode(openSet, neighbor);
                if (otherWay == null)
49                  // No node was found, so it's new, add it to open set
                    openSet.Add(neighbor);
51              else if (neighbor.G < otherWay.G)
                {
53                  // This neighbor has better score
                    // overwrite otherWay with neighbor
55                  openSet.Remove(otherWay);
                    openSet.Add(neighbor);
57              }
            }
59      }
        return null;    // No way found
61  }

63  public List<Node> ReconstructPath(Node node)
    {
65      // The parent parameter of the node is set while creating
        // the neighbors (first time current.Neighbors is called)
67
        List<Node> res = new List<Node>();
69      // Go from the end to the parent of each node
        // until find the root, that has no parent
71      while (node.Parent != null)
        {
73          res.Insert(0, node);
            node = node.Parent;
75      }
        return res;
77  }
```

Algorithmus (6.1): *A*-Algorithmus im C# Stil.*

Die Methoden `float Score(Node from, Node to)` und `float Heuristic(Node from, Node to)`, können nach belieben angepasst werden und haben teilweise große Auswirkungen auf die Arbeitsweise des Algorithmus und sollen in den nächsten Kapiteln besser erklärt werden. Im Normalfall wird als Score die Länge der gesamten Bahn vom Startpunkt geliefert und als Heuristik der euklidische Abstand.

6.2 Heuristische Funktion

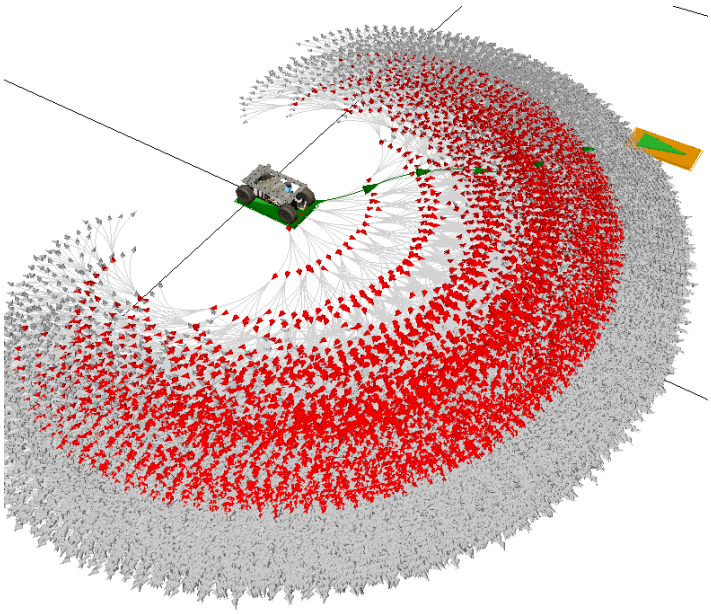
Wie bereits mehrfach angemerkt, hat die heuristische Funktion große Auswirkungen auf die Arbeitsweise des Algorithmus. Nach Definition des A*-Algorithmus soll die Heuristik immer kleiner oder gleich der tatsächlichen Bahn sein.

$$\forall x, y : h(x, y) \leq g(x, y) \tag{6.2}$$

Doch was passiert, wenn man sich nicht an diese Vorgabe hält und die Heuristik übertrieben wird? Oder was, wenn man die Heuristik weglässt? Und wie kann man die Heuristik bis ans äußerste ausreizen, so dass sie immer noch den Vorgaben entspricht? In den nächsten Unterkapiteln werden diese Punkte untersucht und beschrieben.

6.2.1 Weglassen der Heuristik

Lässt man die Heuristik weg, mit anderen Worten gilt für die Heuristik $\forall x, y : h(x, y) = 0$, so ist der Algorithmus nach Definition immer noch optimal. Nur in diesem Fall sind die Kosten $f(x)$ immer gleich dem zurückgelegten Weg $g(s, x)$ vom Startpunkt s aus. Somit verhält sich A* exakt wie der Dijkstra Algorithmus. Der Dijkstra Algorithmus wird oft als Wegfindungsalgorithmus fehlinterpretiert, denn der Algorithmus wurde dafür benutzt den minimalen Spanngraphen zu finden (Dijkstra 1959 [2]). Der Dijkstra Algorithmus breitet sich radial vom Startpunkt aus und findet somit den kürzesten Weg zu **allen** Knoten. Zu Beginn wurde dieser Algorithmus oft für Wegfindung benutzt, später aber (1968) von Peter Hart, Nils J. Nilsson und Bertram Raphael nach und nach durch hinzufügen von Heuristiken zu A* erweitert (Hart, Nilsson & Raphael 1968 [9]).



(Abb. 6.3): *Radiale Ausbreitung eines Dijkstra Algorithmus.*

Wo der Dijkstra Algorithmus noch akzeptable Ergebnisse in einem Graphen oder einem Grid liefert, sprengt die Kalkulation auf Adaptive Search Space den Rahmen. Dies liegt daran, dass nur Konfigurationen in der unmittelbaren Umgebung, mit nahezu gleicher Orientierung zu einer zusammengefasst werden, jedoch dies nicht sehr oft bei Adaptive Search Space auftritt und sich somit ein Erreichbarkeitsbaum aufbaut, deren Elementanzahl exponentiell mit seiner Tiefe wächst. Dijkstra breitet sich von der Startkonfiguration radial aus (Abb. 6.3).

6.2.2 Euklidischer Abstand

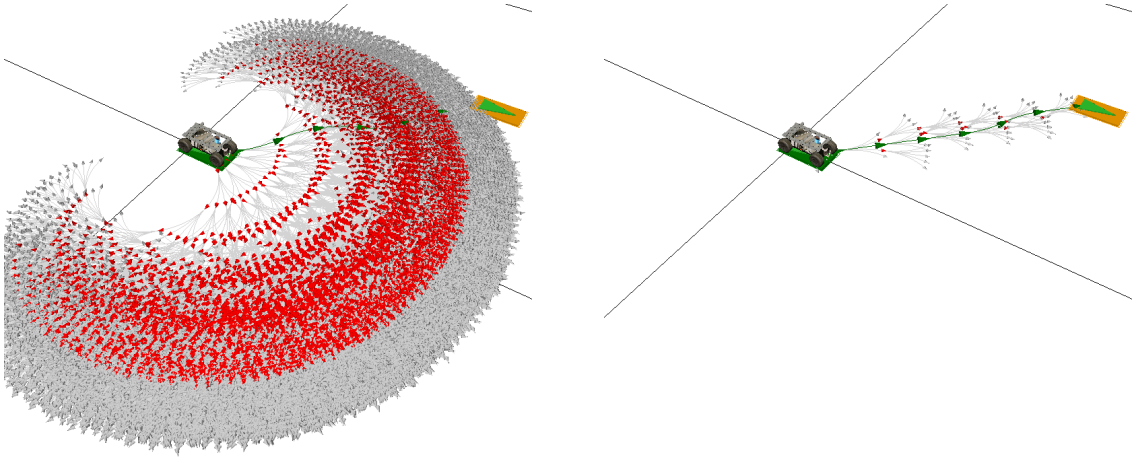
Der euklidische Abstand $d(\vec{a},\vec{b})$ ist die Standard heuristische Funktion und soll an dieser Stelle mathematisch beschrieben werden.

$$d(\vec{a},\vec{b}) = \sqrt{(a_0-b_0)^2+(a_1-b_1)^2} \qquad | \ \vec{a},\vec{b} \in \mathbb{R}^2$$

Die Formel ist nicht nur für $\mathbb{R}^2$ gültig, sondern auch allgemein auf n-Dimensionale Räume anwendbar:

$$d(\vec{a},\vec{b}) = \sqrt{\sum_{i=0}^{n-1}(a_i-b_i)^2} \qquad | \ \vec{a},\vec{b} \in \mathbb{R}^n \tag{6.3}$$

Da der euklidische Abstand die direkte, gerade Verbindung zwischen zwei Punkten ist, ist diese immer kleiner oder gleich dem tatsächlich zurückzulegenden Weg, der abweichen kann, und ist somit optimal. A* breitet sich aufgrund der Heuristik zielgerichtet aus und erreicht erhebliche Verbesserungen im Gegensatz zu Dijkstra (Abb. 6.4).



(a) Dijkstra (A* ohne Heuristik) **(b)** A* mit euklidischem Abstand als Heuristik

(Abb. 6.4): *Dijkstra und A* in direktem Vergleich*

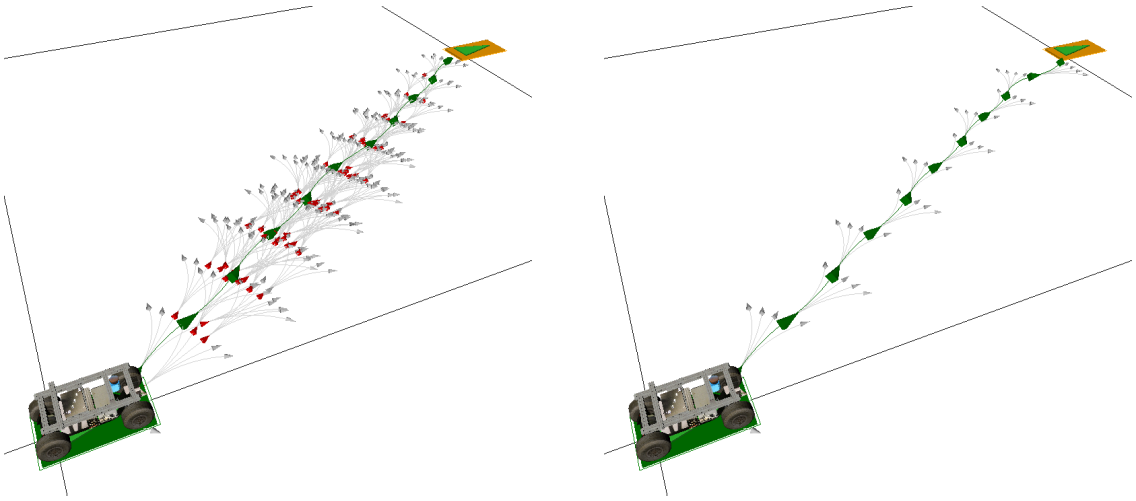
	Dijkstra Algorithmus	**A* mit euklidischem Abstand**
Weglänge in $[m]$	6,32	7,16
Abstand zum Ziel $[m]$	0,99	0,09
Berechnungszeit $[ms]$	12 707	0,9
Expandierte Knoten	8 178	20
Knoten auf dem Heap	40 885	95
Knoten in Open Set	25 806	72
Knoten in Closed Set	8 177	19

(Tab. 6.1): *Messdaten zu (Abb. 6.4).*

Anhand der Messdaten in (Tab. 6.1) erkennt man, dass A* extrem viel schneller arbeitet als Dijkstra. Wo Dijkstra für einen sehr kurzen Weg von ca $7m$ noch fast 13 Sekunden brauch, braucht A* nicht einmal $1ms$, um den optimalen Weg zu finden. Man erkennt aber auch, dass obwohl beide Algorithmen optimal sein sollten, die Weglängen um $0,06m$ (Summe des gefundenen Weges und dem Abstand zum Ziel) variieren. Dies liegt daran, dass nicht exakt zum Ziel gesucht wird, sondern nur bis auf einen Umkreis mit einem bestimmten Radius (hier $1m$), da sonst nie das Ziel erreicht werden würde. Des Weiteren werden nahegelegene Knoten zu einem zusammengefasst, was auch zu kleineren Sprüngen und Unterschieden führen kann.

6.2.3 Gewichteter Abstand

Gewichtet man den euklidischen Abstand mit einem Wert größer als 1, so kann es vorkommen, dass der tatsächliche Weg kürzer ist als die Heuristik und somit die optimalität nicht mehr gewährleistet wird. Wenn man sich jedoch das Verhalten eines solchen Algorithmus anschaut, so merkt man, dass dieser sich viel Zielgerichteter ausbreitet und nicht dazu tendiert weit nach hinten zu gehen und einen anderen Weg zu suchen (Abb. 6.5).



(a) Euklidischer Abstand **(b)** Gewichteter Abstand $(w = 1,5)$

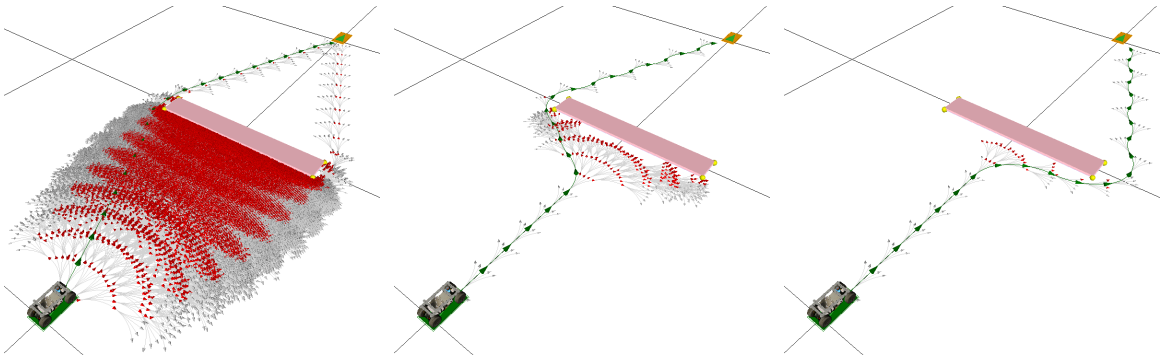
(Abb. 6.5): *A* mit euklidischem Abstand und gewichtetem Abstand im direkten Vergleich.*

	A* mit euklidischem Abstand	**A* mit gewichtetem Abstand**
Weglänge in $[m]$	11,35	11,54
Abstand zum Ziel $[m]$	0,44	0,34
Berechnungszeit $[ms]$	8,8	1,9
Expandierte Knoten	73	12
Knoten auf dem Heap	360	55
Knoten in Open Set	258	44
Knoten in Closed Set	72	11

(Tab. 6.2): *Messdaten zu (Abb. 6.5).*

Die Tabelle (Tab. 6.2) zeigt die Messdaten für A* mit euklidischem Abstand und A* mit gewichtetem Abstand, die im flachen Szenario ohne Hindernisse gemessen wurden (Abb. 6.5). Man erkennt daran, dass selbst in einem Szenario ohne Hindernisse A* mit euklidischem Abstand sehr lange braucht, da bereits expandierte Knoten mehrmals untersucht werden, weil anhand der diskreten Kurven die Wege leicht am Ziel vorbeiführen. A* mit gewichtetem Abstand bevorzugt die Primitiven, die näher an das Ziel führen viel höher, weshalb diese weiter in Richtung Ziel expandiert werden, als bereits untersuchte Knoten zu beginn noch einmal zu untersuchen. Dabei findet dieser einen kaum schlechteren Pfad, der nur $0,09m$ länger ist, dafür aber mit ca $\frac{1}{5}$ Zeit- und Ressourcenaufwand. Ähnlich der Argumentation in Kapitel 6.2.2 kann diese Wegdiskrepanz vernachlässigt und als optimal angesehen werden (Äquidistanz der Primitiven vorausgesetzt).

Wo A* mit euklidischem Abstand noch den optimalen Weg, mit etwas langsamer Kalkulationszeit geliefert hat, versagt der Algorithmus komplett, wenn bereits ein kleines Hindernis in den Weg kommt. Es entsteht wieder nahezu ein gesamter Erreichbarkeitsbaum als Tiefensuche bis zum Objekt und der Algorithmus braucht sehr lange, um einen Weg um das Hindernis zu finden. Aber bereits mit einer geringen Gewichtung wird eine Bahn sehr schnell gefunden. Mit steigendem Gewicht ist die Suche viel schneller, jedoch immer suboptimaler. Es stellt sich hier die Frage, ob man Optimalität gegen Berechnungszeit eintauschen sollte (Abb. 6.6).



(a) Euklidischer Abstand **(b)** Gewicht $w = 2$ **(c)** Gewicht $w = 10$

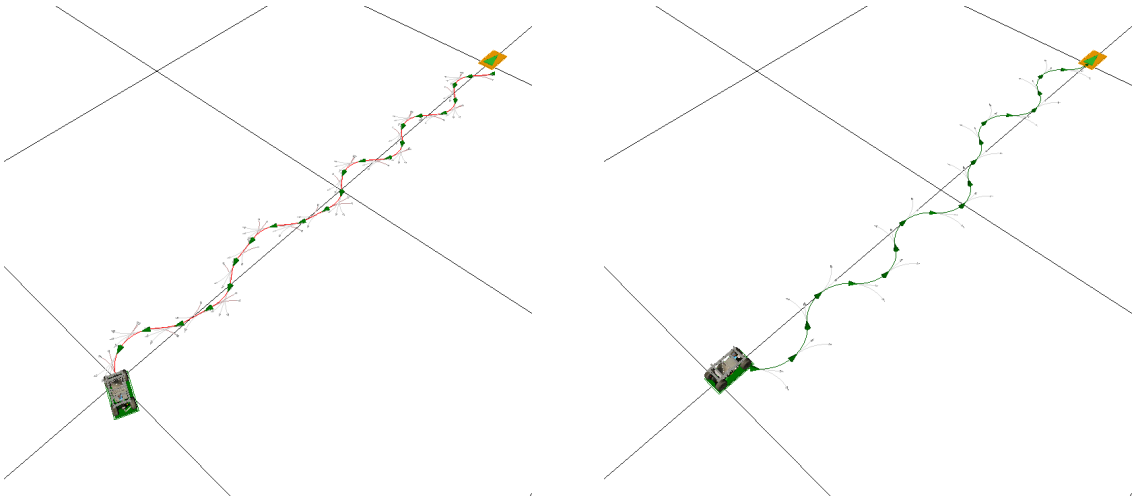
(Abb. 6.6): *A* mit unterschiedlichen Gewichtungen der Heuristik im Vergleich.*

	Euklidischer Abstand	**Gewicht** $w = 2$	**Gewicht** $w = 10$
Weglänge in $[m]$	21,21	22,30	23,30
Abstand zum Ziel $[m]$	0,45	0,88	0,93
Berechnungszeit $[ms]$	287 717	26,4	3,9
Expandierte Knoten	58 343	110	27
Knoten auf dem Heap	291 710	545	130
Knoten in Open Set	69 344	252	79
Knoten in Closed Set	105 184	277	51

(Tab. 6.3): *Messdaten zu (Abb. 6.6).*

Die Tabelle (Tab. 6.3) zeigt Messdaten zu einem Szenario mit einem Hindernis (Abb. 6.6). A* mit euklidischem Abstand würde mit fast 5 Minuten Kalkulationszeit alle Zeitschranken brechen. Wenn Kalkulationszeit keine Rolle spielt, dann liefert dieser Algorithmus den optimalsten Weg (mit den vorgegebenen Motion Primitiven) zum Ziel. Mit nur $1,52m$ längerem Weg, was etwa 7% entspricht, braucht A* mit einem Gewicht $w = 2$ nur noch $26,4ms$ und damit etwa $\frac{1}{10\ 000}$ der Zeit und Ressourcen. A* mit einem Gewicht $w = 10$ bringt weitere $1,05m$ Wegeinbuße, $2,57m$ $(\approx 11,87\%)$ zum ursprünglichem Weg und ist noch einmal um eine Faktor 10 schneller, was in zeitkritischen Systemen weitere Zeit- und Ressourceneinsparnisse bringen kann. Schaut man sich die gefundenen Pfade grafisch an (Abb. 6.6), so erkennt man, dass der optimale Weg nicht der Weg ist, den ein Mensch fahren würde, denn ein Mensch würde kurz vor einem weit entferntem Hindernis ausweichen und nicht den kürzesten Weg nehmen.

Die Gewichtung der Entfernung hat neben der Suboptimalität noch eine weitere Nebenwirkung. Dadurch dass näher am Ziel liegende Konfigurationen sehr viel besser bewertet werden, werden längere Primitiven meist bevorzugt, obwohl diese viel schlechter sind. Dadurch können Rückwärtsprimitiven den Vorwärtsprimitiven vorgezogen werden, wenn das Ziel hinter der Startkonfiguration liegt (Abb. 6.7 a). Wenn Kurven länger sind, als die Gerade, so werden diese bevorzugt, obwohl das Hindernis in direkter Fahrtrichtung liegt und auch Geraden als Primitiven existieren (Abb. 6.7 b).

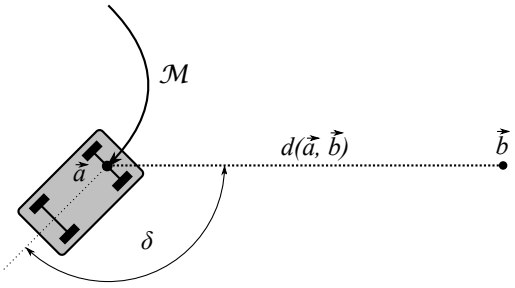


(a) Rückwärtsfahren wird bevorzugt **(b)** Kurven werden bevorzugt

(Abb. 6.7): *Hervorgehobene Nachteile an Heuristik mit gewichtetem Abstand in A*.*

6.2.4 Gewichtete Orientierung

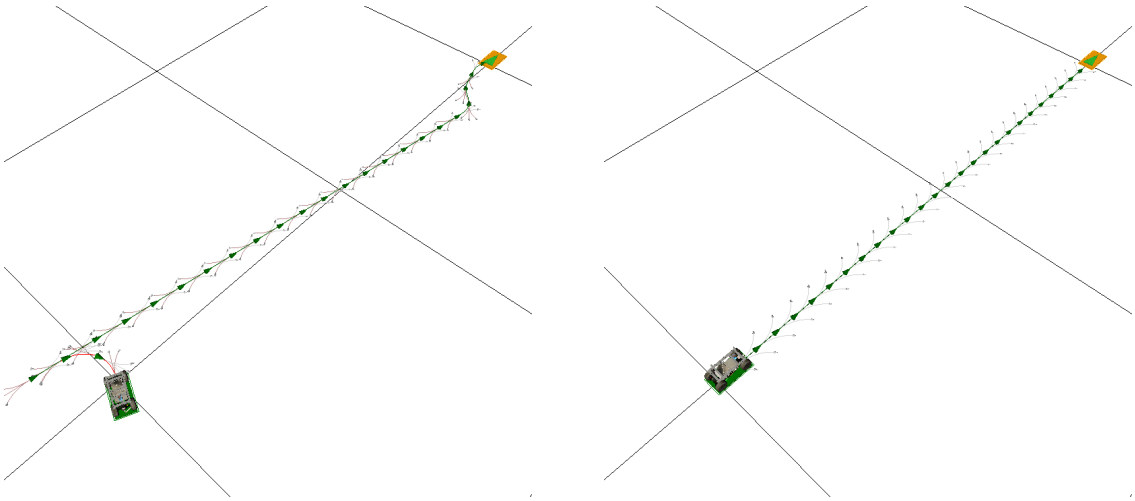
Sind die Primitiven beim gewichteten Abstand nicht äquidistant, oder werden auch Rückwärtsprimitiven benutzt, so zeigt A* mit gewichtetem Abstand ein komplettes Fehlverhalten in der Suche. Aufgrund der Tatsache, dass Längere Primitiven meist bevorzugt werden, obwohl diese nichts direkt zum Ziel führen, kann das Gewicht variabel gehalten werden. Als Gewicht eignet sich beispielsweise die Orientierung des Fahrzeugs und die Abweichung vom Ziel. Da der Algorithmus sich in Richtung des Ziels ausbreiten soll, müssen die Primitiven besser bewertet werden, deren Richtung näher zum Ziel zeigt. Sei δ der Winkel zwischen der Orientierung des Fahrzeugs im Punkt $\vec{a}$ und der Richtung zum Ziel $\vec{b}$ (Abb. 6.8).



(Abb. 6.8): *Orientierung als Gewichtung. Winkel δ zwischen Orientierung des Fahrzeugs und Ziel.*

$$h(\vec{a},\vec{b}) = (1 - w \cdot \delta) \cdot d(\vec{a},\vec{b})$$

Dabei ist w eine Gewichtungskonstante mit $w \geq 1$ und $d(\vec{a},\vec{b})$ der euklidische Abstand zwischen Punkt $\vec{a}$ und Punkt $\vec{b}$ (Eq. 6.3).

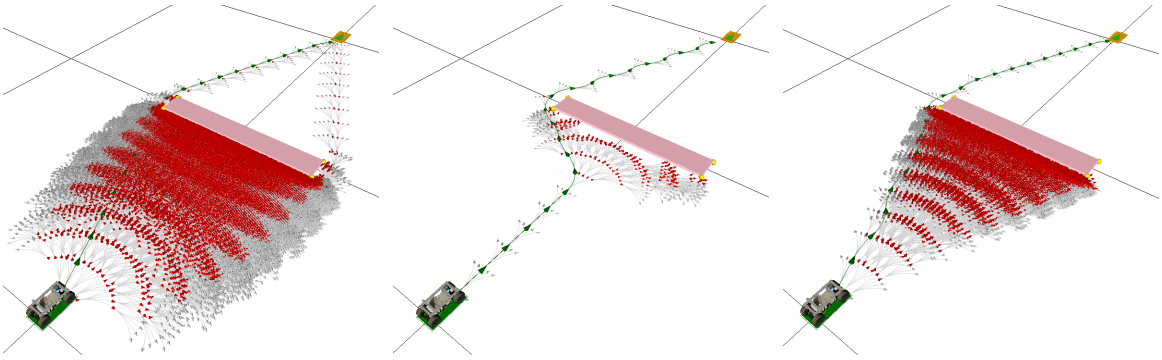


(a) Wendevorgang. **(b)** Kurven werden nicht länger bevorzugt

(Abb. 6.9): *Beseitigte Nachteile des A* mit gewichtetem Abstand, mithilfe variabler Gewichtung durch Orientierung.*

Wo noch A* mit gewichtetem Abstand ein Fehlverhalten aufwies (Abb. 6.7), berechnet A* mit gewichteter Orientierung eine korrekte Bahn, sogar mit Rangieren, um das Ziel hinter der Startkonfiguration zu erreichen. Auch die Äquidistanz der Primitiven ist nicht mehr notwendig, denn es wird die Primitive bevorzugt, die besser Richtung Ziel zeigt, um sich in diese Richtung auszubreiten (Abb. 6.9).

Wo der A* Algorithmus mit gewichtetem Abstand noch seine Vorteile hat, hat die gewichtete Orientierung leider ihren Nachteil, denn kommt ein Hindernis in den Weg zwischen Start- und Endkonfiguration, werden Primitiven mit Ausrichtung zum Ziel bevorzugt, so dass um das Hindernis nur schwer umfahren werden kann, denn dafür muss man mit der Orientierung vom Ziel abweichen. Die Suche breitet sich Konusartig aus, bis diese am Hindernis vorbeiführt. Zum Vergleich sind in (Abb. 6.10) der euklidische Abstand und der gewichtete Abstand, neben der gewichteten Orientierung als Heuristik abgebildet.



(a) Euklidischer Abstand **(b)** Gewicht $w = 2$ **(c)** Gewichtete Orientierung mit $w = 1$

(Abb. 6.10): *A* mit unterschiedlichen Heuristiken im Vergleich in einem Szenario mit Hindernis.*

	Euklidischer Abstand	**Gewicht $w = 2$**	**Gewichtete Orientierung**
Weglänge in $[m]$	21,21	22,30	21,59
Abstand zum Ziel $[m]$	0,45	0,88	0,36
Berechnungszeit $[ms]$	287 717	26,4	6 257
Expandierte Knoten	58 343	110	11 271
Knoten auf dem Heap	291 710	545	56 350
Knoten in Open Set	69 344	252	10 055
Knoten in Closed Set	105 184	277	27 966

(Tab. 6.4): *Messdaten zu (Abb. 6.10).*

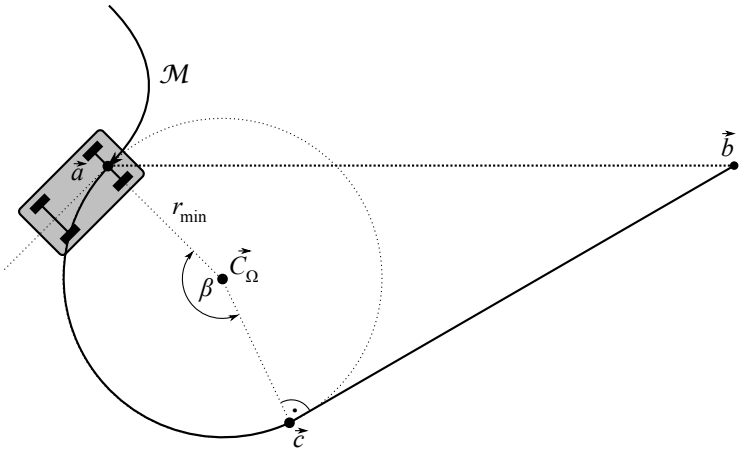
Die Tabelle (Tab. 6.4) zeigt, dass die gewichtete Orientierung bei einem Hindernis mit ≈ 6 Sekunden nicht der schnellste Algorithmus ist, der jedoch um Faktor 50 schneller ist als der naive Ansatz mit dem euklidischen Abstand. Mit $21,95m$ liefert dieser einen nur um $0,29m$ längeren Weg, der nur $\approx 1,3\%$ länger ist als der optimale.

6.2.5 KREISBOGENABSTAND

In den letzten Kapiteln wurde gezeigt, dass die Berechnungszeit des Suchalgorithmus stark von der Heuristik abhängig ist. Dabei gilt, je größer der Wert der Heuristik ist, desto schneller wird der Algorithmus. Optimal bleibt der Algorithmus jedoch solange für die Heuristik gilt:

$$\forall x, y : h(x, y) \leq g(x, y) \quad (Eq.6.2)$$

Bisher wurde dafür immer die Position des Fahrzeugs in Betracht gezogen, da aber das Fahrzeug nicht auf der Stelle wenden kann sind zwei Konfigurationen an der gleichen Position mit unterschiedlichen Orientierungen keinesfalls äquivalent, wodurch die Heuristik noch verbessert werden kann. Bei der gewichteten Orientierung wurde zwar die Orientierung des Fahrzeugs mit einbezogen, wurde jedoch als Faktor für die Distanz verwendet, so dass der minimale Weg überschätzt werden konnte und somit der Algorithmus nicht mehr optimal war. Möchte man jedoch einen optimalen Pfad garantieren, so muss man den tatsächlichen minimalen Pfad berechnen, dieser setzt sich aus einem minimalen Wendekreis bis zur Ausrichtung zum Ziel und der Geraden ab dieser Position zusammen (Abb. 6.11).

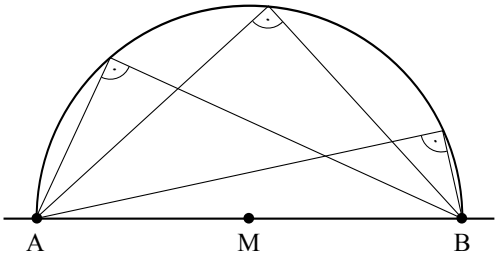


(Abb. 6.11): *Kreisbogenabstand als Heuristik. Das Fahrzeug muss minimal den Kreisbogen und die Gerade abfahren, als kürzesten Weg von $\vec{a}$ nach $\vec{b}$.*

Zunächst muss der Kreismittelpunkt $\vec{C_\Omega}$ bestimmt werden, um den das Fahrzeug fahren muss, um die Ausrichtung zum Ziel zu bekommen. Dieser ist davon abhängig, ob der Winkel links oder rechts vom Fahrzeug zum Ziel kleiner ist, um den kürzeren Weg zu haben. Sei $\vec{n}$ der normalisierte Normalenvektor zur Orientierung des Fahrzeugs, der abhängig der kürzeren Seite nach links oder nach rechts zur Orientierung des Fahrzeugs zeigt (in Abb. 6.11 links zur Fahrzeugausrichtung). Dann berechnet sich $\vec{C_\Omega}$ als:

$$\vec{C_\Omega} = \vec{a} + r_{\min} \cdot \vec{n} \qquad | \; r_{\min} \text{ nach } (Eq.2.2)$$

Um den Punkt $\vec{c}$ berechnen zu können, muss die Tangente an dem Kreis durch $\vec{b}$ berechnet werden. Dazu bedient man sich dem Satz des Thales, der zunächst besagt, dass Alle Winkel in einem Halbkreisbogen rechte Winkel sind (Abb. 6.12).



(Abb. 6.12): Satz des Thales: *Alle Winkel in einem Halbkreisbogen sind rechte Winkel.*

Es muss zunächst der Mittelpunkt $\vec{M}$ zwischen $\vec{C_\Omega}$ und $\vec{b}$ berechnet werden, der den Mittelpunkt des Thaleskreises bildet und seinen Radius r_T:

$$\vec{M} = \frac{\vec{C_\Omega} + \vec{b}}{2}$$
$$r_T = \frac{|\vec{C_\Omega} - \vec{b}|}{2}$$

Der Schnitt des Thaleskreises mit dem zuvor berechneten Kreis, gibt zwei Punkte, zu denen der Punkt $\vec{c}$ gehört. Der richtige Punkt liegt in der gleichen Richtung, wie die zuvor berechnete Normale $\vec{n}$ des Fahrzeugs, somit muss das Skalarprodukt der beiden positiv sein, um den Punkt $\vec{c}$ zu identifizieren.

$$(\vec{c} - \vec{C_\Omega}) \cdot \vec{n} \geq 0$$

Der Winkel β berechnet sich durch:

$$\beta = \cos^{-1}\left(\frac{(\vec{a} - \vec{C_\Omega}) \cdot (\vec{c} - \vec{C_\Omega})}{r_{\min}^2}\right)$$

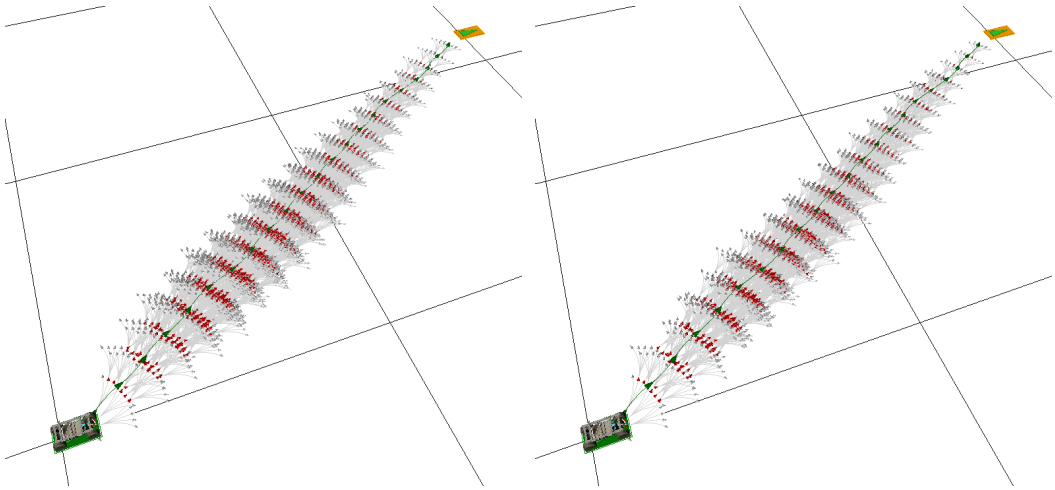
Die endgültige Länge des Weges, die zugleich die Heuristik darstellt berechnet sich schließlich:

$$h(\vec{a}, \vec{b}) = r_{\min} \cdot \beta + |\vec{b} - \vec{c}|$$

Dabei ist zu beachten, dass β in Radians vorliegen muss.

Diese Heuristik ist größer oder gleich dem euklidischen Abstand, wobei Gleichheit nur dann vorliegt, wenn das Fahrzeug exakt Richtung Ziel ausgerichtet ist. Dies bedeutet, dass man etwas besser schätzt, als mit dem euklidischen Abstand, jedoch immer noch geringer, als der tatsächliche Weg.

$$\forall x, y : d(x, y) \leq h(x, y) \leq g(x, y)$$

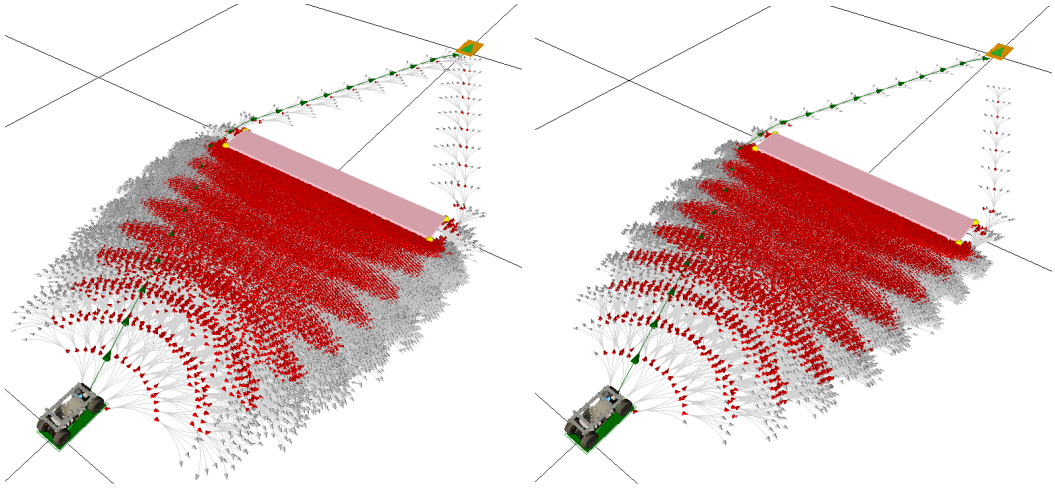


(a) Euklidischer Abstand **(b)** Kreisbogenabstand

(Abb. 6.13): *Vergleich optimaler Heuristiken in A*.*

	Euklidischer Abstand	**Kreisbogenabstand**
Weglänge in $[m]$	22,58	22,58
Abstand zum Ziel $[m]$	0,96	0,96
Berechnungszeit $[ms]$	212	150
Expandierte Knoten	1 133	866
Knoten auf dem Heap	5 660	4 325
Knoten in Open Set	3 544	2 687
Knoten in Closed Set	1 132	865

(Tab. 6.5): *Messdaten zu (Abb. 6.13).*



(a) Euklidischer Abstand **(b)** Kreisbogenabstand

(Abb. 6.14): *Vergleich optimaler Heuristiken in A* mit Hindernis.*

	Euklidischer Abstand	**Kreisbogenabstand**
Weglänge in $[m]$	21,21	21,21
Abstand zum Ziel $[m]$	0,45	0,45
Berechnungszeit $[ms]$	287 717	132 596
Expandierte Knoten	58 343	42 719
Knoten auf dem Heap	291 710	213 590
Knoten in Open Set	69 344	41 281
Knoten in Closed Set	105 184	84 537

(Tab. 6.6): *Messdaten zu (Abb. 6.14).*

In beiden Simulationen, deren Messdaten in den Tabellen (Tab. 6.5) und (Tab. 6.6) aufgezeichnet sind, liefern sowohl A* mit euklidischem Abstand, als auch A* mit Kreisbogenabstand exakt den gleichen Weg. Anhand der Bilder (Abb. 6.13) und (Abb. 6.14) erkennt man nur minimale Abweichungen in den Suchbäumen, wohingegen die Messdaten deutlich zeigen, dass A* mit Kreisbogenabstand fast um den Faktor 2 schneller ist, als A* mit euklidischem Abstand und auch Ressourcensparender ist. Der Geschwindigkeitsschub ist zwar nicht enorm, wenn aber die Optimalität des Weges gewährleistet sein soll, so bietet diese Heuristik eine gute Alternative zum naiven Ansatz mit euklidischen Abständen.

6.3 DATENSTRUKTUREN

A* arbeitet auf zwei Listen, dem **Open Set** und dem **Closed Set**. In das Open Set kommen alle neu erstellten Knoten, diese sollten nach Möglichkeit nicht redundant sein. Aufgrund mathematischer Ungenauigkeiten beim Rechnen (numerische Fehlerfortpflanzung), haben zwei Konfigurationen niemals die gleichen Werte, sondern es werden nahegelegene Konfigurationen zusammengefasst. Dafür muss zunächst die gesamte Menge durchsucht werden, ob ein ähnlicher Knoten bereits vorhanden ist und wenn dies der Fall sein sollte, so ist diese Konfiguration über mindestens zwei mögliche Pfade erreichbar, von denen der günstigste ausgewählt wird und der andere verworfen wird. Wie der Name Open Set (engl. offene Menge) bereits vermuten lässt, beinhaltet diese noch die nicht untersuchten, offenen Konfigurationen, aus denen die günstigste Konfiguration ausgewählt wird, um an dieser Stelle weiter zu expandieren. Wird eine Konfiguration expandiert, oder wird an einer neu expandierten Konfiguration eine Kollision festgestellt, so kommt diese Konfiguration in das Closed Set (engl. geschlossene Menge). Beim Expandieren einer Konfiguration werden alle neu entstandenen Konfigurationen getestet, ob diese bereits im Closed Set sind (oder ähnlich nah dran liegen). Wenn dieser Umstand erkannt wird, bedeutet dies, dass man eine bereits bekannte Konfiguration erreicht hat, die bereits expandiert wurde und an dieser Stelle nicht weiter gemacht werden soll.

Das Open Set muss eine schnelle Auswahl der günstigsten Konfiguration gewährleisten und unter anderem auch eine räumliche Sortierung vornehmen, um bereits vorhandene Konfigurationen finden zu können. Das Closed Set muss nur bereits vorhandene Konfigurationen ausschliessen, indem die Konfigurationen räumlich Sortiert werden, um diese schnell wieder zu finden. Eine schnelle Auswahl durch eine Relation sortierbarer Elemente erfüllen so genannte Prioritätswarteschlangen (engl. priority queue), im folgenden Kapitel 6.3.1 Skew Heap wird eine solche vorgestellt. Da Konfigurationen durch drei Parameter x, y und θ identifiziert werden, eignet sich für eine räumliche Sortierung und schnelle Auswahl ein Octree, der im gleichnamigen Kapitel 6.3.3 vorgestellt wird.

6.3.1 Skew Heap

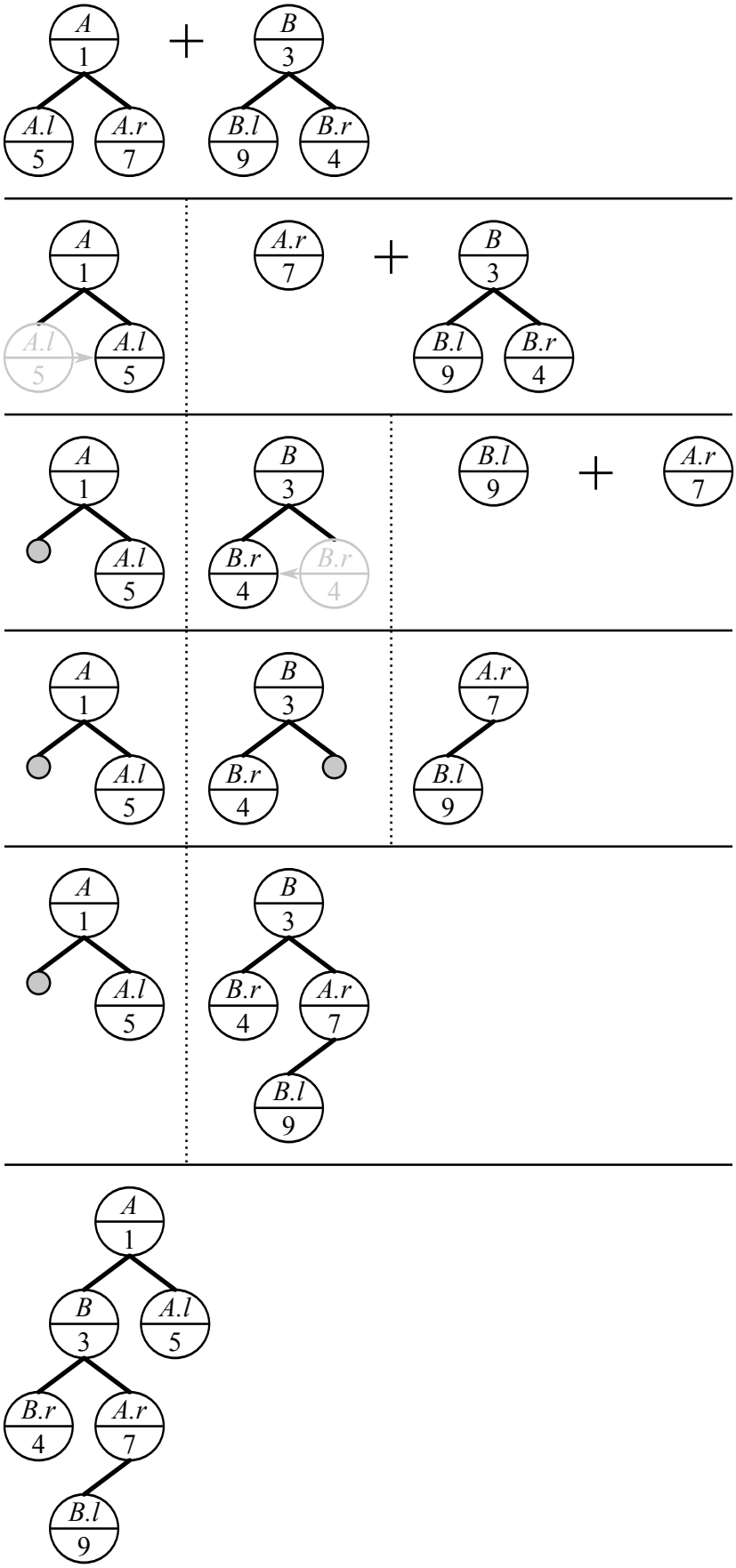
Der Skew Heap ist eine spezielle Prioritätswarteschlange (Sleator & Tarjan 1986 [36]). In erster Linie ist der Skew Heap von der Datenstruktur ein binärer Baum (engl. binary tree), an dessen Wurzel sich das kleinste (oder größte, je nach Sortierungsart) Element befindet. Mit absteigender Tiefe werden die Elemente immer größer, so dass sich an dessen Blättern die größten Elemente des Baumes befinden. Der Skew Heap stellt eine rekursive Funktion `private SkewNode merge(SkewNode left, SkewNode right)` bereit, die zwei Skew Heaps zu einem zusammenfasst. Dabei werden gleichzeitig der linke und rechte Teilbaum ständig vertauscht, wo durch diese Verwirbelung die Ausgeglichenheit des Baums zumindest im Mittel bewahrt wird. Ein generischer Algorithmus eines Skew Heaps ist im Algorithmus (6.2) beschrieben, dieser kann als Bibliothek kompiliert auch in anderen Projekten verwendet werden, dafür müssen die zu sortierenden Elemente nur das Interface `IComparable` implementieren.

```
1  public class SkewHeap<T> where T : IComparable
   {
3      public SkewNode<T> root = null;
       public int Count { private set; get; }
5
       public T Peek()
7      {
           if (root == null)
9              return default(T);
           return root.Data;
11     }

13     public T GetFirst()
       {
15         if (root == null)
               return default(T);
17
           T res = root.Data;
19         root = merge(root.Left, root.Right);
           Count--;
21         return res;
       }
23
       public void Add(T data)
25     {
           SkewNode<T> node = new SkewNode<T>(data);
27         root = merge(root, node);
           Count++;
29     }

31     private SkewNode merge(SkewNode left, SkewNode right)
       {
```

```
33          if (left == null) return right;
            if (right == null) return left;
35
            if (left.CompareTo(right) < 0)
37          {
                SkewNode swap = left.Left;
39              left.Left = merge(left.Right, right);
                left.Right = swap;
41              return left;
            }
43          else
            {
45              SkewNode swap = right.Right;
                right.Right = merge(right.Left, left);
47              right.Left = swap;
                return right;
49          }
        }
51
        public class SkewNode<T> where T : IComparable
53      {
            public T Data { set; get; }
55          public SkewNode<T> Left { set; get; }
            public SkewNode<T> Right { set; get; }
57
            public SkewNode(T data)
59          {
                this.Data = data;
61          }

63          public int CompareTo(SkewNode<T> other)
            {
65              return Data.CompareTo(other.Data);
            }
67      }
    }
```

Algorithmus (6.2): *Skew Heap im C# Stil.*

Aus den beiden Bäumen A und B wird die Wurzel mit dem geringsten Wert als neue Wurzel ausgewählt, es sei in diesem Fall der Baum A. Der linke und rechte Nachfolger $A.l$ und $A.r$ stellen ebenfalls je einen Baum dar, von denen einer so bleibt wie er ist und der andere mit Baum B rekursiv über die gleiche Methode wieder gemischt wird. Zuletzt werden die neu entstandenen Bäume $A.l$ und $A.r$ noch vertauscht. In der Abbildung (Abb. 6.15) ist dieses Verhalten der Merge-Funktion skizziert. Dabei sind vertikal einzelne Schritte von einander getrennt und horizontal die Rekursionstiefe.



(Abb. 6.15): *Arbeitsweise eines Skew Heaps.*

Das Ansehen des ersten Elements geschieht direkt in konstanter Zeit, denn es liegt an der Wurzel. Will man dieses Element gleichzeitig entfernen, so müssen die beiden Teilbäume gemischt (engl. merge) werden. Dafür geht die rekursive Funktion bis zum tiefsten Element und mischt dabei jeweils die Teilbäume, was im middle case $\log_2(n)$ Schritte braucht und im worst case, wenn der Baum entartet ist und als lineare Liste vorliegt, braucht dieser im Mittel $\frac{n}{2}$ Schritte. Zum Einfügen wird das neue Element als Baum angesehen und muss mit dem bereits bestehenden Baum gemischt werden, was wiederum bis zum tiefsten Element rekursiv mischt und ähnlich zum Entfernen im middle case $\log_2(n)$ Schritte braucht. Die Tabelle (Tab. 6.7) vergleicht die Komplexität eines Skew Heaps mit der einer naiven Liste.

	Skew Heap		**Naive Liste**	
	middle case	worst case	middle case	worst case
kleinstes Element ansehen	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
kleinstes Element entfernen	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Element hinzufügen	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Beliebiges Element entfernen	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

(Tab. 6.7): *Vergleich Skew Heap mit einer naiven Liste.*

6.3.2 SORTIERTE LISTE

Eine weitere Möglichkeit bietet eine Sortierte Liste. Dadurch dass nur einzelne Elemente eingefügt werden, kann eine Liste leicht sortiert gehalten werden, wodurch sich das Entfernen des kleinsten Elements auf einen konstanten Zugriff minimiert. Es gibt verschiedene Möglichkeiten die Liste sortiert zu halten, neben dem naiven Ansatz den Einfügeplatz von Beginn der Liste iterativ zu suchen, kann das Prinzip „Devide & Conquer“ ausgenutzt werden. Man teilt die Liste in zwei Hälften und schaut sich das mittlere Element an, ist das einzufügende Element größer als dieses, so kommt als Einfügestelle nur die hintere Hälfte in Frage, denn alle Elemente der ersten Hälfte sind kleiner. Dies kann man rekursiv weiter fortführen, bis die zu untersuchende Liste kein einziges Element mehr enthält, was die Einfügestelle für das Element ist. Der Vorteil zu dem zuvor genannten Skew Heap ist, dass die Binäre Suche immer ausgenutzt werden kann und es keinen Baum gibt der entarten kann, sondern die Suche den Bereich immer teilt und somit auch im worst case die Suche $\log_2(n)$ Schritte braucht (Tab. 6.8).

	Skew Heap		**Sortierte Liste**	
	middle case	worst case	middle case	worst case
kleinstes Element ansehen	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
kleinstes Element entfernen	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Element hinzufügen	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$
Beliebiges Element entfernen	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$

(Tab. 6.8): *Vergleich Skew Heap mit einer sortierten Liste.*

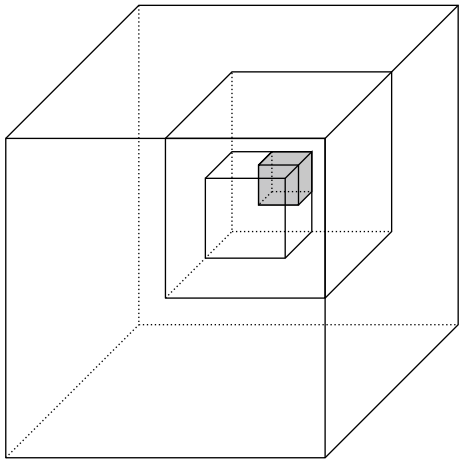
Ein generischer Algorithmus zum Einfügen in eine solche Liste könnte folgendermassen aussehen (Algorithmus 6.3):

```
public class BinarySortedList<T> where T : IComparable
{
    public List<T> list = new List<T>();

    public void Add(T data)
    {
        add(data, 0, list.Count);
    }

    private void add(T data, int from, int to)
    {
        if (from >= to)
        {
            list.Insert(from, data);
            return;
        }

        int mid = (from + to) / 2;
        if (list[mid].CompareTo(data) < 0)
            add(data, mid + 1, to);
        else
            add(data, from, mid);
    }
}
```

Algorithmus (6.3): *Funktion* `Add` *einer sortierten Liste im C# Stil.*

6.3.3 Octree

Möchte man effizient nach Konfigurationen suchen, so bietet der Octree eine gute Möglichkeit die Daten so zu organisieren, dass die Suche in logarithmischer Zeit abläuft. Da die Konfigurationen drei Dimensionen x, y und θ haben und diese nach dem Prinzip „Devide & Conquer“ zweigeteilt werden, entstehen 8 mögliche Teilbäume. Dieses Prinzip wiederholt man so lange, bis die Abstände wischen den Rändern klein genug sind, so dass man diese als gleichen Knoten zusammenfassen würde. Diese Elemente des Octrees werden Blätter genannt, da der Raum an dieser Stelle nicht mehr weiter geteilt wird. Die Blätter des Octrees enthalten dann die gewünschten Daten und sind alle in gleicher Tiefe, woraus zu folgern ist, dass der Zugriff darauf konstant ist. Dabei ist noch zu unterscheiden, ob in den Blättern eine Liste aller Konfigurationen gepflegt wird. Da im Falle des Closed Set nur darauf geachtet werden muss, ob es an dieser Stelle bereits eine Konfiguration vorhanden ist, müsste diese nichtmal selbst gespeichert werden, sondern nur der Raum markiert werden. Im Falle des Open Sets möchte man aber unter Umständen die Referenz auf die vorliegende Konfiguration haben, so dass nur eine Konfiguration in den Blättern des Octrees gespeichert werden muss. In der Abbildung (Abb. 6.16) ist der Aufbau eines Octrees der Tiefe drei Skizziert. Das ausgegraute Element wurde hier nach drei Suchschritten bereits auf $\frac{1}{8^3} = \frac{1}{512}$ des gesamten Raumes eingeschränkt.



(Abb. 6.16): *Anschauliche Ansicht eines Octrees.*

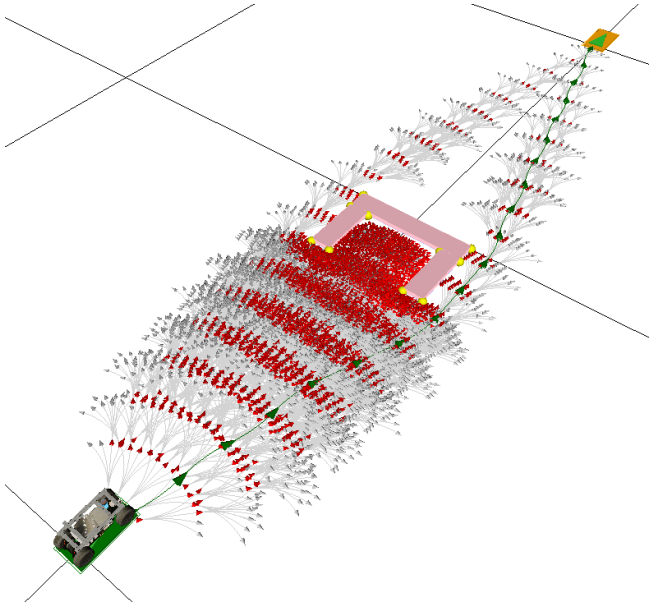
Vergleicht man den Octree mit einer naiven Liste, so ist zwar das Hinzufügen in beiden Fällen konstant, da es bei der naiven Liste nur hinten angehängt werden muss. Jedoch beim Suchen eines ähnlichen Elements muss im schlimmsten Falle die gesamte Liste durchlaufen werden, ohne dass ein Element gefunden wird, im best case jedoch ist direkt das erste Element ähnlich und die Suche kann mit diesem Ergebnis beendet werden. Im Mittel braucht die Suche aber $\frac{n}{2}$ Schritte, wenn das gesuchte Element vorhanden ist, was jedoch in den meisten Fällen nicht der Fall ist (Tab. 6.9). An dieser Stelle kann der Octree nicht mit einer sortierten Liste oder dem Skew Heap verglichen werden, denn diese beiden werden nach dem Score $f(x)$ sortiert und der Octree anhand der Konfigurationen eingeteilt.

	Octree		**Naive Liste**	
	middle case	worst case	middle case	worst case
Element hinzufügen	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Ähnliches Element suchen	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

(Tab. 6.9): *Vergleich Octree mit einer naiven Liste.*

VERSUCHSAUFBAU

Die genannten Datenstrukturen sollen nun getestet werden. Damit die Vorteile der genannten Datenstrukturen zum Einsatz kommen, müssen möglichst viele Zugriffe auf die Liste getätigt werden. Somit wird der naive A* Algorithmus mit dem euklidischen Abstand in allen Simulationsläufen verwendet, zusätzlich wird noch ein kleines Hindernis in den Weg gestellt, damit die Suche möglichst viel Zeit in Anspruch nimmt und die Listen groß anschwellen lässt. Es wurde das in Abbildung (Abb. 6.17) gezeigte Szenario gewählt.



(Abb. 6.17): *Szenario zum Testen der Datenstrukturen.*

	Naive Listen	**Skew Heap & Octree**	**Sortierte Liste & Octree**
Weglänge in $[m]$	20,42	20,42	20,42
Abstand zum Ziel $[m]$	0,16	0,16	0,16
Berechnungszeit $[ms]$	13 960	1 191	936
Zeit Open Set $[ms]$	4 142	216	52
Zeit Closed Set $[ms]$	8 974	78	75
Expandierte Knoten	1 969	2 912	2 912
Knoten auf dem Heap	9 840	14 555	14 555
Knoten in Open Set	3 022	4 053	4 053
Knoten in Closed Set	3 767	5 889	5 889

(Tab. 6.10): *Messdaten zu (Abb. 6.17).*

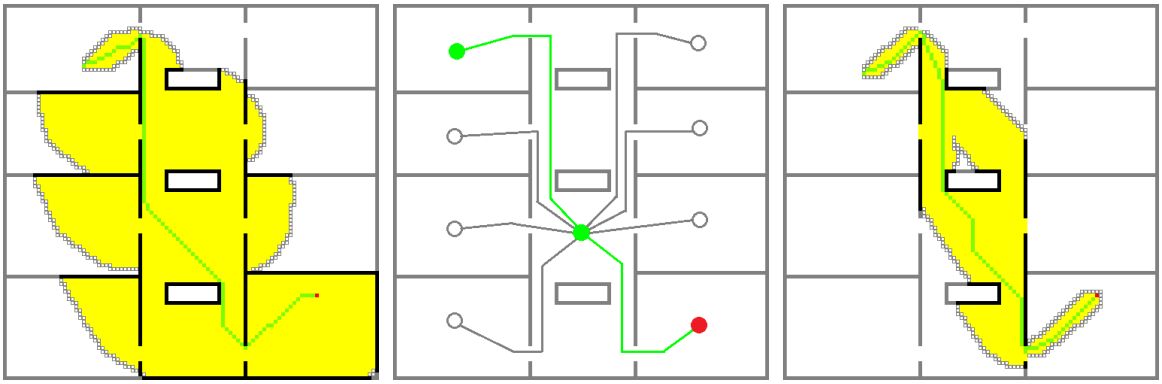
Anhand der Tabelle (Tab. 6.10), die mit verschiedenen vorgestellten Datenstrukturen im Szenario (Abb. 6.17) aufgenommen wurden, erkennt man, dass durch die Datenstrukturen bereits eine signifikante Verbesserung um mehr als den Faktor 10 erzielt werden konnte. Gleichzeitig verbrauchen die Datenstrukturen ca 50% mehr Ressourcen und expandieren 50% mehr Konfigurationen, obgleich sie sehr viel schneller arbeiten. Dies liegt daran, dass der Octree nahegelegene Konfigurationen anders zusammenfasst, als die naive Liste. Bei der naiven Liste wird eine Konfiguration im Radius zu der gesuchten Konfiguration geliefert (falls vorhanden), der Octree diskretisiert den Raum, so dass Konfigurationen auch näher aneinander liegen können als gewollt. Da die sortierte Liste nur im Open Set verwendet wird, wird dieser nur im Zugriff auf das Open Set um ca Faktor 4 schneller. Dabei ist noch zu beachten, dass für das Open Set zwei Listen gleichzeitig gepflegt werden, die jeweils für ihren optimierten Zugriff verwendet werden.

6.4 HIERARCHISCHE SUCHE

Hierarchien zeigen in vielen Bereichen einen guten Ansatz, unter anderem bereits in den Datenstrukturen (z. B. Octree) umgesetzt. Aber wie sieht es aus, wenn man die Suche hierarchisch gestalten könnte? Auf diese Fragestellung wird auch im Buch (Millington & Funge [28]) eingegangen, dazu folgt ein Gedankenexperiment als Motivation.

GEDANKENEXPERIMENT:

Gegeben sei ein Gebäude aus mehreren Räumen, möchte man aus einem Raum in einen anderen planen, so gibt es zunächst keinen Anhaltspunkt über mögliche Hindernisse und Wände. Wenn der Algorithmus naiv sich in Richtung des Ziels ausbreitet, muss er zunächst die Tür auf der anderen Seite finden. Des Weiteren plant der Algorithmus in mehrere Räume hinein, die man gar nicht untersuchen möchte (Abb. 6.18 a). Es wäre logisch nicht direkt auf der Karte zu planen, sondern eine Ebene höher, zunächst von Raum zu Raum. Es kann vorher ein Graph erstellt werden, wo die einzelnen Knoten die Räume darstellen und die Kanten Verbindungen zwischen den Räumen sind und Informationen zu den Positionen der Türen haben (Abb. 6.18 b). Es muss nicht nur von Start zum Ziel, sondern über Zwischenziele geplant werden. Dadurch spart man sich die Suche in unbeteiligten Räumen und kann zielgerichteter von Tür zu Tür planen (Abb. 6.18 c).

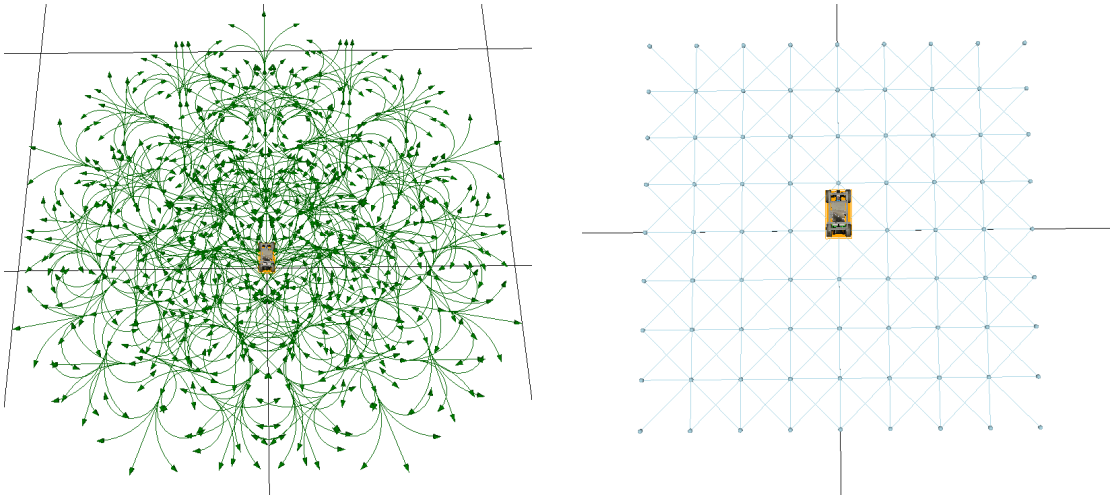


(a) Naiver Ansatz **(b)** Raumhierarchie **(c)** Hierarchische Suche

(Abb. 6.18): *Gedankenexperiment zur hierarchischen Suche. (gelb → Closed Set, weiss-grau → Open Set, grün → Pfad, rot → Startposition)*

6.4.1 GRIDSUCHE

Vergleicht man die Erreichbarkeitsbäume des Adaptive Search Space mit dem eines Grids, so erkennt man, dass das Grid sehr viel weniger Knoten hat und eine hohe Wiederholung besitzt (Abb. 6.19). Der Erreichbarkeitsbaum der Adaptive Search Space wurde hier nur aus 5 Nachfolgern aufgebaut, bei 8 Nachfolgern besäße der Baum der Tiefe 4 bereits $8^4 = 4\,096$ Knoten. Das Grid hat ebenfalls 8 Nachfolger und bei einer Tiefe von 4 hat der Baum gerade mal $9 * 9 = 81$ Knoten. Dieser Umstand kann zur hierarchische Suche beitragen, indem man zunächst grob den Weg auf einem Grid vorplant und dann die Zwischenziele anfährt.

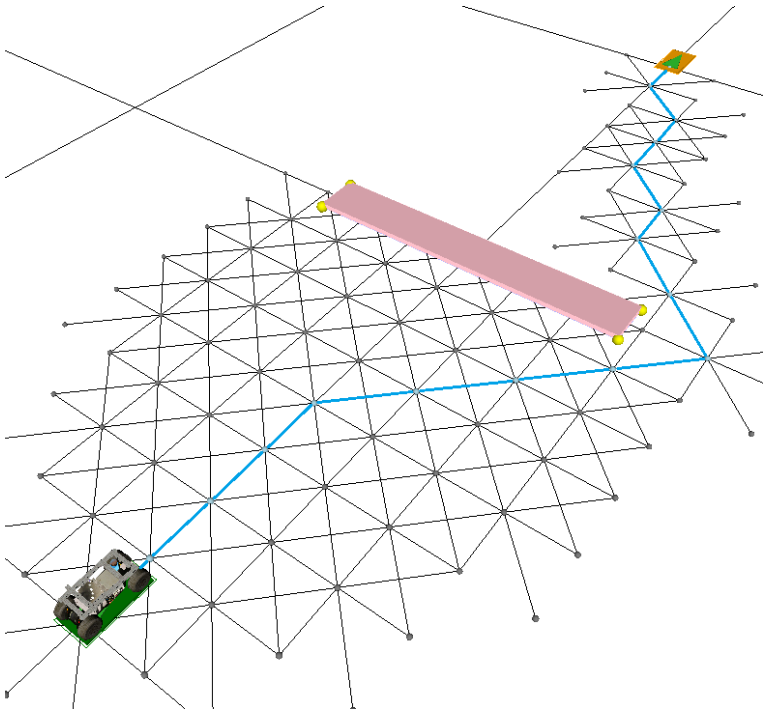


(a) Adaptive Search Space **(b)** Grid

(Abb. 6.19): *Erreichbarkeitsbäume der Tiefe 4 des Adaptive Search Space und des Grids.*

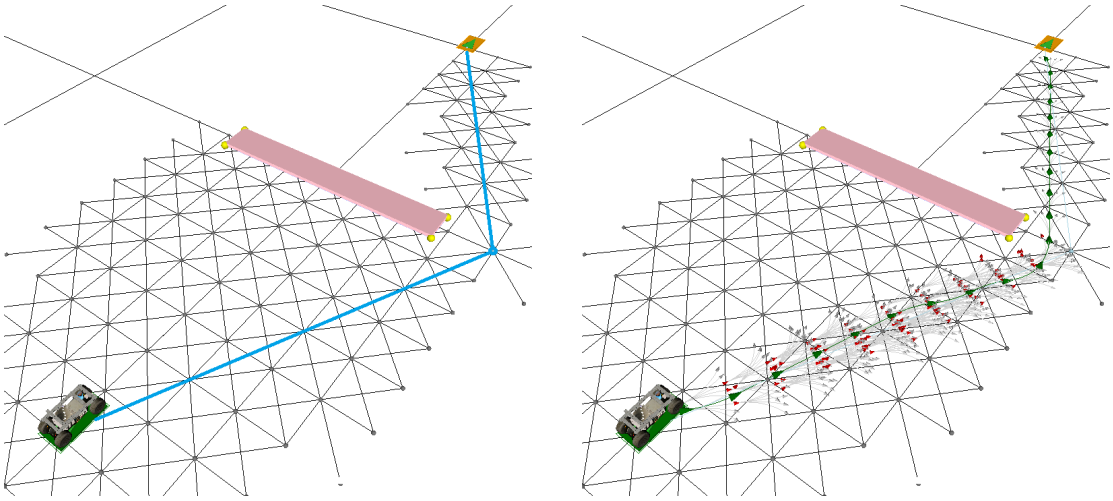
6.4.2 THETA*

Wird eine heuristische Suche auf einem Grid ausgeführt, so ist die Suche zwar sehr schnell, führt aber zu einem stark gezackten Pfad, den man auf keinen Fall fahren würde. Ausserdem sind die Punkte zu stark frequentiert, so dass man Schwierigkeiten hätte alle Punkte anzufahren (Abb. 6.20).



(Abb. 6.20): *A* Suche auf einem Grid.*

Eine Möglichkeit wäre alle nicht benötigten Knoten zu verwerfen, die auf geradem Wege ohne Kollision erreichbar wären. Der Theta* Algorithmus (Nash, Daniel & Koenig 1968 [29]) bietet eine Möglichkeit dies direkt während der Suche zu gewährleisten. Der Algorithmus ist im Grunde identisch zu A*, es wird lediglich die Kostenfunktion $g(x)$ für den zurückgelegten Weg angepasst. Wo bei A* nur die Länge des tatsächlich gefahrenen Weges gemessen wurde, wird beim Theta* ein Kollisionstest mit der Geraden zwischen dem aktuellen Knoten und dem Vorgänger seines Vaterknotens getestet. Wird eine Kollision erkannt, so werden nur die Kosten ausgerechnet, anderenfalls wird der Vorgänger des aktuellen Knotens umreferenziert auf den Vorgänger seines Vaterknotens, wodurch eine direkte Verbindung über mehrere Grideckpunkte entsteht (Abb. 6.21).



(a) Theta* Gridsuche **(b)** Hierachische A* Suche nach Theta*

(Abb. 6.21): *Hierarchische Suche: Thera* Gridsuche liefert Zwiscenziele, die über A* angefahren werden.*

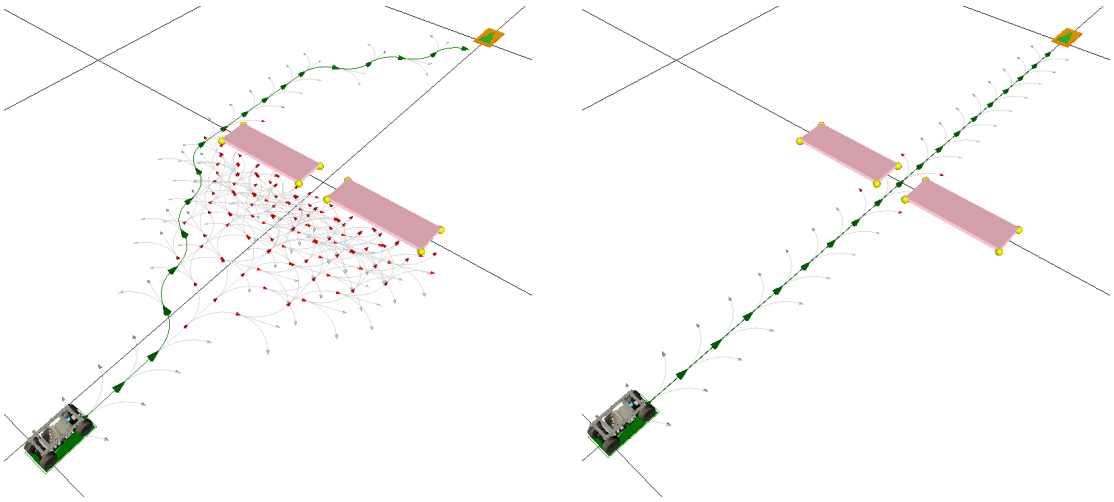
	A* mit Kreisbogenabstand	**Theta* + A* mit Kreisbogenabstand**
Weglänge in $[m]$	21,21	21,38
Abstand zum Ziel $[m]$	0,45	0,80
Berechnungszeit $[ms]$	132 596	4 + 22
Expandierte Knoten	42 719	60 + 107
Knoten auf dem Heap	213 590	472 + 525
Knoten in Open Set	41 281	52 + 386
Knoten in Closed Set	84 537	66 + 111

(Tab. 6.11): *Messdaten zu (Abb. 6.21 b) im Vergleich zu Tabelle (Tab. 6.6).*

Die Tabelle (Tab. 6.11) zeigt den Vergleich zu dem zuvor vorgestellten Algorithmus A* mit Kreisbogenabstand im Szenario (Abb. 6.14), zu dem gleichen Algorithmus im gleichen Szenario, wo vorher der Weg grob mit Theta* berechnet wurde. Der minimal längere Weg lässt sich dadurch erklären, dass Theta* auf einem diskreten Grid plant und der Punkt etwas zu sehr abseits des Hindernis geplant wurde. Dafür spricht der Geschwindigkeitsschub um den Faktor 50 000 für sich, von vorher ca 2 Minuten braucht der Algorithmus mit Theta* gerade mal $26ms$.

6.5 Ergebnisse und Evaluation

Die heuristische Suche bietet eine Möglichkeit von einer Startkonfiguration zu einer gegebenen Endkonfiguration einen Weg mit Kollisionsvermeidung zu planen. Aber einen Schwachpunkt haben alle diese genannten Algorithmen der bisher noch nicht genannt wurde. Alle Algorithmen weisen Schwierigkeiten auf, wenn sie in engen Räumen suchen müssen, oder durch einen Engpass planen müssen. In Abbildung (Abb. 6.22) ist ein Szenario mit einem Engpasshindernis gezeigt. In beiden Fällen wurden gleiche Fahrzeugparameter, der gleiche Suchalgorithmus und gleiche Primitiven benutzt, der einzige Unterschied besteht nur in der leicht versetzten Startkonfiguration, die bereits dazu führt, dass die Lücke im Hindernis nicht mehr gefunden wird.

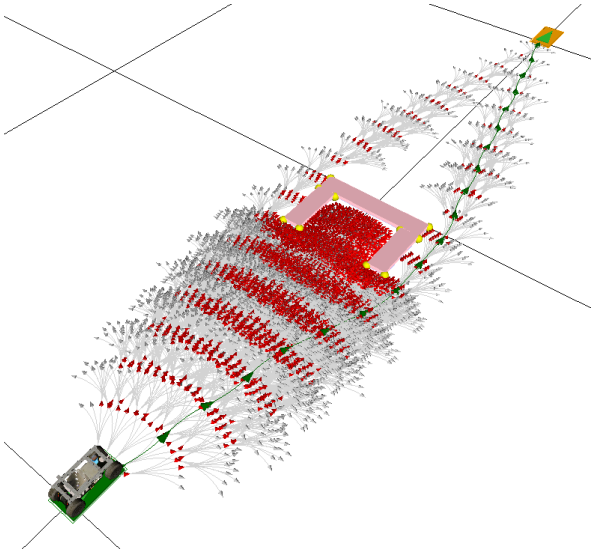


(a) Lücke wird nicht gefunden. **(b)** Lücke kann durchfahren werden.

(Abb. 6.22): *Engpässe werden von heuristischen Suchen übersehen.*

Dieses Problem weisen alle hier aufgeführten heuristischen Suchalgorithmen auf. Das Problem lässt sich theoretisch sehr leicht lösen, denn der Grund dafür sind die diskreten Punkte, die angefahren werden können. Sie entstehen durch die diskrete Berechnung der Primitiven mit den verschiedenen Samplingverfahren. Dies lässt sich dadurch beheben, dass entweder kürzere Primitiven oder eine höhere Anzahl an Nachfolgekonfigurationen berechnet werden. Beide Lösungsversuche führen aber durch die Exponentiell wachsende Erreichbarkeitsbäume zu stetig steigenden Suchzeiten. Eine andere Möglichkeit bietet die Umgebung besser zu definieren, indem z. B. zwischen den Hindernissen ein Weg definiert wird und dieser über online Sampling angefahren wird.

Zur Evaluation der gezeigten Algorithmen wurde das bereits bekannte Szenario mit einem Hindernis verwendet (Abb. 6.23). Die getesteten Algorithmen sind der naive A*, wo die naiven Datenstrukturen verwendet wurden. A* mit optimierten Datenstrukturen und die hierarchische Suche mit Theta* vor A*. Des Weiteren wurde mit jedem Algorithmus jeweils verschiedene Heuristiken verwendet. Darunter die naive Heuristik mit dem euklidischem Abstand, die optimierte Heuristik mit dem Kreisbogenabstand, dazu wurden die nicht optimalen Heuristiken mit der Orientierung und dem gewichteten Abstand genommen.

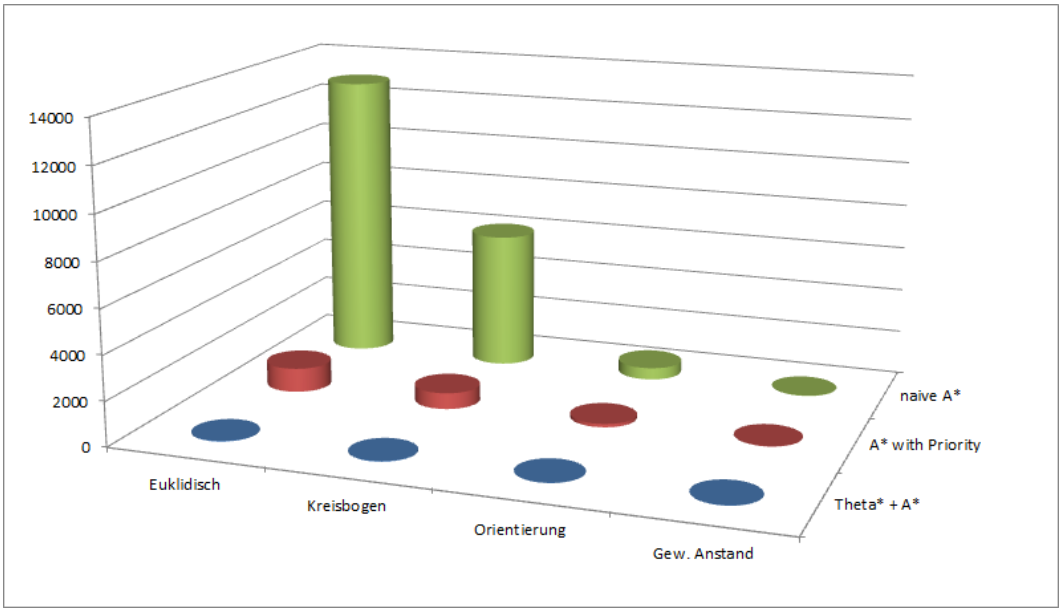


(Abb. 6.23): *Szenario zur Evaluation.*

ZEITMESSUNG

	Euklidisch	Kreisbogen	Orientierung	Gew. Abstand
naive A*	12 981	6 151	538	18
A* with Priority	1 087	730	149	13
Theta* + A*	29	26	7	8

(Tab. 6.12): *Zeitmessung in ms zum Szenario (Abb. 6.23).*



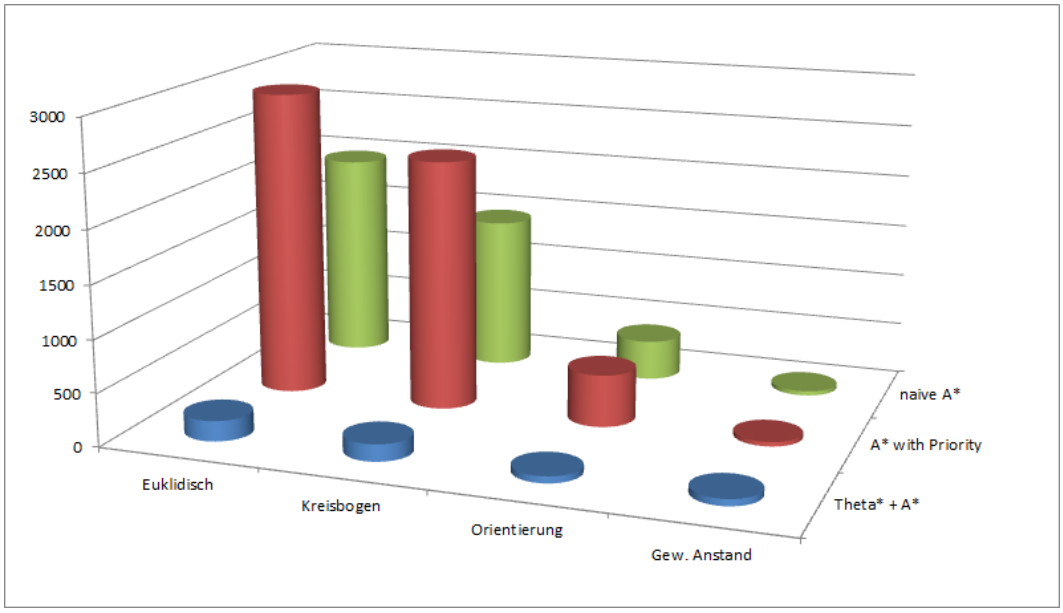
(Abb. 6.24): *Zeitdiagramm zu Tabelle (Tab. 6.12).*

Anhand der Tabelle (Tab. 6.12) oder dem Diagramm (Abb. 6.24) erkennt man sofort, dass der Algorithmus Theta* + A* die Suche dominiert und ein sehr gutes Zeitverhalten aufweist. Sehr langsam sind hierbei eigentlich nur der naive A* mit den optimalen Heuristiken.

Expandierte Knoten

	Euklidisch	**Kreisbogen**	**Orientierung**	**Gew. Abstand**
naive A*	1 969	1 456	381	41
A* with Priority	2 912	2 378	496	42
Theta* + A*	194	162	64	64

(Tab. 6.13): *Messung der expandierten Knoten zum Szenario (Abb. 6.23).*



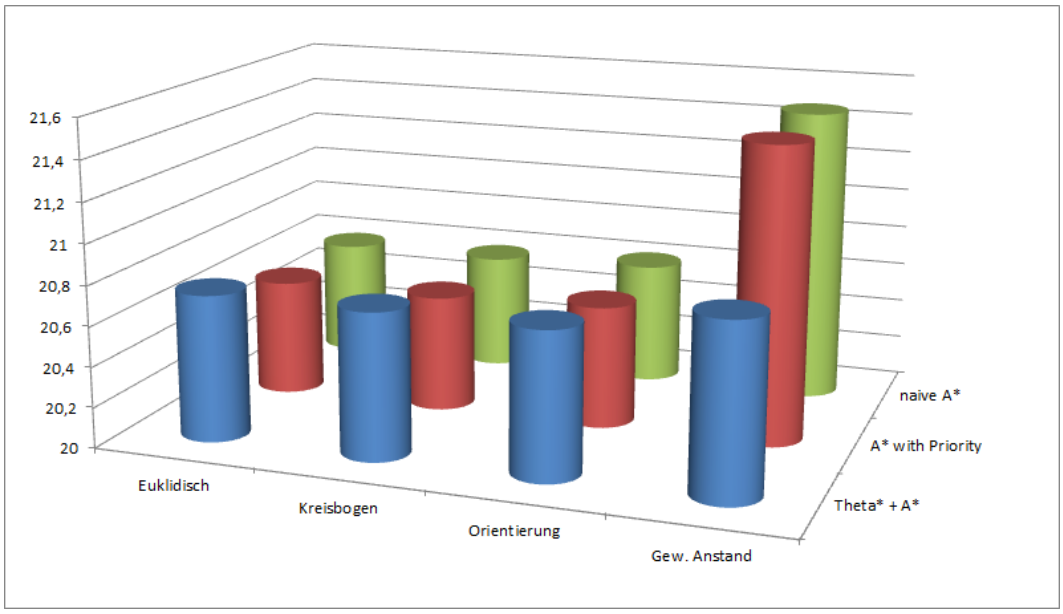
(Abb. 6.25): *Anzahl expandierter Knoten Diagramm zu Tabelle (Tab. 6.13).*

Anhand der Tabelle (Tab. 6.13) oder dem Diagramm (Abb. 6.25) erkennt man, dass Theta* mit A* auch den Ressourcenverbrauch dominiert, aber auch dass der wesentlich schnellere A*-Algorithmus mit den optimierten Datenstrukturen mehr Ressourcen verbraucht, aber nur unwesentlich. Nur die beiden A*-Algorithmen mit den optimalen Heuristiken ohne den hierarchischen Theta* verbrauchen sehr viele Ressourcen, alle anderen Algorithmen halten sich im Rahmen.

PFADLÄNGE

	Euklidisch	Kreisbogen	Orientierung	Gew. Abstand
naive A*	20,58	20,58	20,61	21,47
A* with Priority	20,58	20,58	20,61	21,47
Theta* + A*	20,73	20,73	20,73	20,86

(Tab. 6.14): *Erfassung der Weglängen in Metern zum Szenario (Abb. 6.23).*



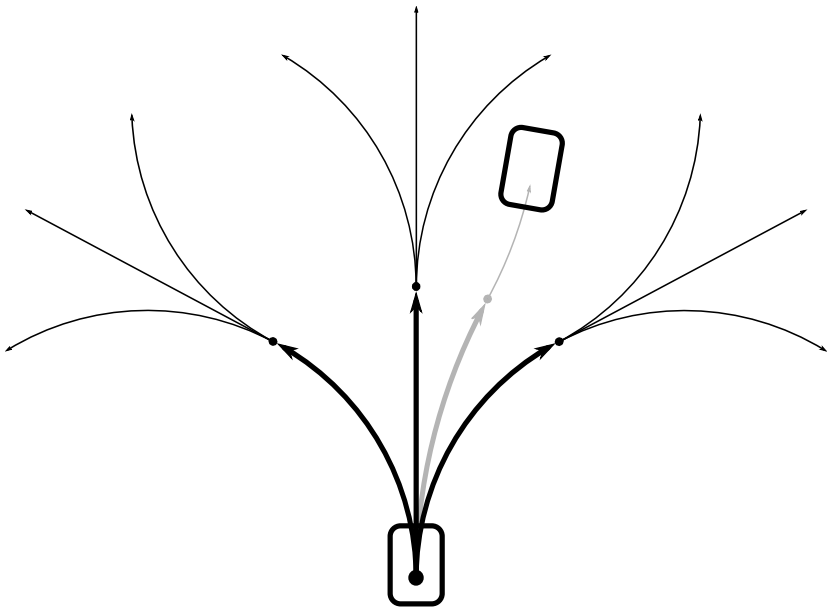
(Abb. 6.26): *Weglängendiagramm zu Tabelle (Tab. 6.14).*

Anhand der Tabelle (Tab. 6.14) oder dem Diagramm (Abb. 6.26) erkennt man, dass bis auf den gewichteten Abstand alle Algorithmen einen nahezu gleichlangen Weg liefern und kaum Abweichungen aufweisen. Der gewichtete Abstand sticht jedoch sofort raus mit einem extrem viel längerem Pfad. Interessant ist auch, dass der gewichtete Abstand bei der hierarchischen Vorsuche mit Theta* wieder einen nahezu perfekten Pfad findet.

7

REEDS SHEPP

In den ersten Kapiteln dieser Arbeit wurden verschiedene Motion Primitiven vorgestellt (Kapitel 4). Des Weiteren wurden Samplingverfahren (Kapitel 5) vorgestellt, mit denen eine Menge von Motion Primitiven vorberechnet werden kann, die den kinematischen Einschränkungen des Fahrzeugs genügen. Und schließlich wurden heuristische Suchalgorithmen (Kapitel 6) vorgestellt, um eine Position anzufahren. Aufgrund des exponentiellen Wachstums des Erreichbarkeitsbaums, gibt es nur eine endliche Anzahl an vorberechneten, möglichen Motion Primitiven, die auch eine bestimmte Länge haben. Es können also nur diskrete Raumpunkte angefahren werden. Möchte man in eine Pose dazwischen planen, so muss man sich damit zufrieden geben, dass man mit heuristischen Suchalgorithmen mit einem geringem Abstand- und Winkelfehler die Pose anfahren kann (Abb. 7.1).



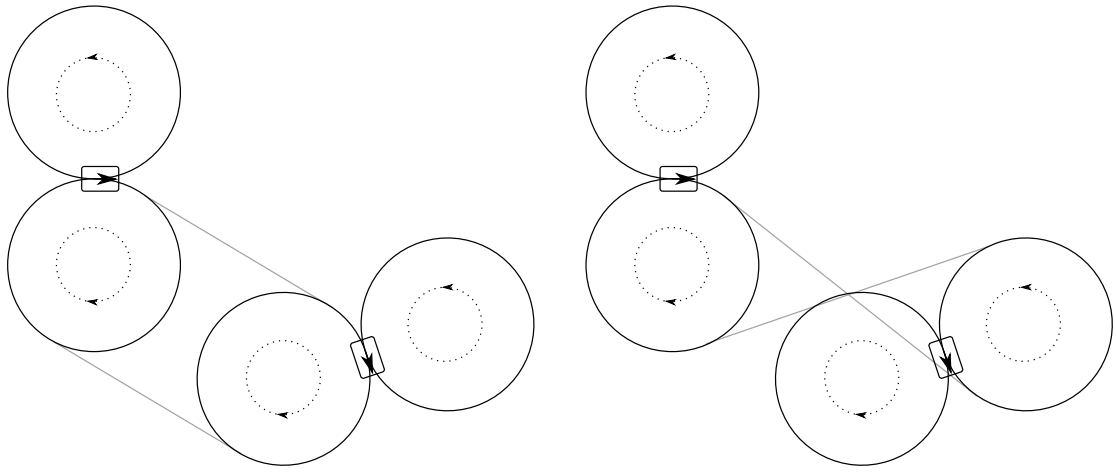
(Abb. 7.1): *Problem an heuristischen Suchalgorithmen durch Diskreditierung. (grau → valider Pfad, schwarz → vorberechnete Motion Primitiven)*

In diesem Kapitel wird ein Verfahren vorgestellt, mit dem aus einer Pose in eine andere Pose geplant werden kann. Dieser Algorithmus schließt zwar Kollisionen aus, plant aber nicht um Hindernisse herum. Trotzdem muss nicht zwangsläufig aus einer Startpose geplant werden, sondern es kann eine Konfiguration aus dem Ergebnispfad der A* Suche benutzt werden, so dass mit diesem Werkzeug der letzte Schritt gemacht werden kann um die endgültige Pose anzufahren.

7.1 REEDS SHEPP

Dieser Algorithmus ist nicht der gleiche Algorithmus, wie er im Paper (Reeds & Shepp 1990 [32]) vorgestellt wird, sondern es wird das geometrische Konstrukt aus Motion Primitiven aufgebaut, anhand von dem mit A* und Kollisionserkennung der kürzeste Weg berechnet wird. Liegt ein großes Hindernis zwischen Start und Ziel, wird kein Pfand umhergeplant, sondern der Algorithmus liefert, dass kein Pfad gefunden wurde.

Die Grundidee von diesem Algorithmus ist, dass ein Fahrzeug mit vollem Lenkeinschlag in die eine oder andere Richtung je einen Kreis mit minimalem Radius $r_{\min}$ beschreiben kann. Aus diesem Grund werden sowohl an die Start-, als auch an die Endkonfiguration je zwei Kreise angehängt. Nun kann zwischen diesen Kreisen auf einer Tangentenbahn gewechselt werden. Dabei muss die Fahrtrichtung der Kreisbahn beachtet werden. Zwei Kreisbahnen, die in die gleiche Richtung befahren werden, werden über Aussentangenten verbunden (Abb. 7.2 a). Zwei entgegengesetzte Kreise werden über Innentangenten verbunden (Abb. 7.2 b). Der Grund dafür ist, dass man nach dem Wechsel der Kreise auf dieser Kreisbahn fahren kann, aber keine Wende machen kann und somit das Anfahren der richtigen Endkonfiguration nicht mehr gewährleistet ist.



(a) Aussentangenten **(b)** Innentangenten

(Abb. 7.2): *Verbindung der Kreisbahnen mit Tangenten.*

ERREICHBARKEITSBAUM

Von den errechneten Tangenten kann je eine für die Vorwärtsfahrt benutzt werden. Die Fahrtrichtung der Geraden ist auch unabhängig der Fahrtrichtung der Kreisbahnen fest definiert. Da jeder der Startkreise mit den beiden Endkreisen auf je zwei Bahnen erreichbar ist, gibt es für Vorwärtspfade der Kreisbahnen insgesamt nur 8 Möglichkeiten. Zieht man in Betracht, dass nach dem Erreichen der Endkreise, diese sowohl vorwärts als auch rückwärts befahren werden können, so verdoppelt sich die Anzahl auf 16 Möglichkeiten. Das selbe gilt für die Startkreise, somit gilt für die Größe des Erreichbarkeitsbaum 32 mögliche Pfade der Tiefe 3, dieser ist aus insgesamt 16 Motion Primitiven aufgebaut.

7.1.1 KONSTRUKTION: ÄUSSERE KREISTANGENTEN

Die äußeren Kreistangenten werden benötigt, um zwei in gleicher Richtung verlaufende Kreisbahnen mit einer Geraden zu verbinden (Abb. 7.3).

SCHRITT 1

Gegeben seien zwei Kreise:

$$k_1 : (\vec{x} - \vec{C_1})^2 = r_1^2$$
$$k_2 : (\vec{x} - \vec{C_2})^2 = r_2^2$$

SCHRITT 2

Berechne Thaleskreis k_T mit Mittelpunkt $\vec{M}$ und Radius r_T:

$$\vec{M} = \frac{\vec{C_1} + \vec{C_2}}{2}$$
$$r_T = \frac{|\vec{C_1} - \vec{C_2}|}{2}$$
$$k_T : (\vec{x} - \vec{M})^2 = r_T^2$$

Berechne den Hilfskreis $k_2^{'}$ mit dem Mittelpunkt des größeren Kreises (hier $\vec{C_2}$) und Differenz der Radien als Radius:

$$k_2^{'} : (\vec{x} - \vec{C_2})^2 = (r_2 - r_1)^2$$

SCHRITT 3

Berechne Schnittmenge S vom Thaleskreis k_T und Hilfskreis $k_2^{'}$ (Schnittmenge S habe in diesem Fall zwei Schnittpunkte $\vec{S}$ und $\vec{S'}$):

$$S = \{\vec{S}, \vec{S'}\} = k_T \cap k_2^{'}$$

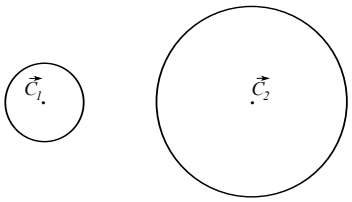
Bestimme Gerade g durch $\vec{S}$ (und $\vec{S'}$) und $\vec{C_1}$:

$$g : \vec{x} = \vec{C_1} + \lambda \cdot (\vec{S} - \vec{C_1})$$

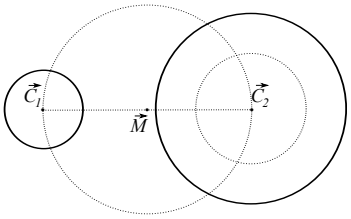
SCHRITT 4

Verschiebe die Gerade um Vektor $\vec{D}$, die die Tangentengerade g_T ergibt:

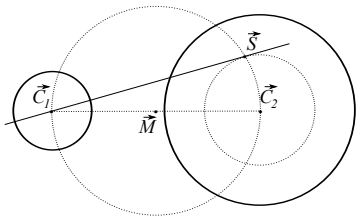
$$\vec{D} = r_1 \cdot \frac{\vec{S} - \vec{C_2}}{|\vec{S} - \vec{C_2}|}$$
$$g_T : \vec{x} = \vec{C_1} + \vec{D} + \lambda \cdot (\vec{S} - \vec{C_1})$$



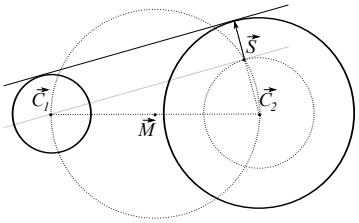
(a) 1. Schritt: Ausgangszustand. Zwei Kreise mit $\vec{C_1}$ und $\vec{C_2}$ als Mittelpunkte und Radien r_1 und r_2



(b) 2. Schritt: Thaleskreis mit Mittelpunkt $\vec{M}$ und Hilfskreis mit Mittelpunkt $\vec{C_2}$ mit Radius $r = r_2 - r_1$ konstruieren.



(c) 3. Schritt: Gerade durch Schnittpunkt des Thaleskreis mit Hilfskreis und Zentrum des anderen Kreises $\vec{C_1}$.



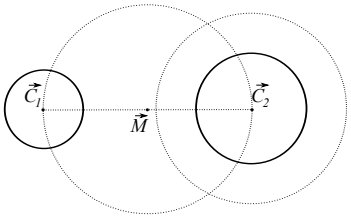
(d) 4. Schritt: Gerade in Richtung des Zentrums $\vec{C_2}$ um Länge $-r_1$ verschieben.

(Abb. 7.3): *Konstruktion der äußeren Kreistangenten.*

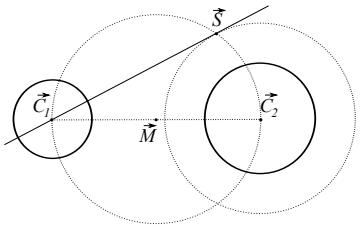
7.1.2 KONSTRUKTION: INNERE KREISTANGENTEN

Die inneren Kreistangenten werden benötigt, um zwei in entgegengesetzter Richtung verlaufende Kreisbahnen mit einer Geraden zu verbinden (Abb. 7.4).

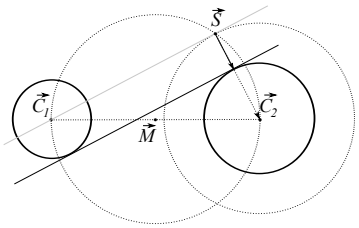
(a) 1. Schritt: Ausgangszustand. Zwei Kreise mit $\vec{C_1}$ und $\vec{C_2}$ als Mittelpunkte und Radien r_1 und r_2



(b) 2. Schritt: Thaleskreis mit Mittelpunkt $\vec{M}$ und Hilfskreis mit Mittelpunkt $\vec{C_2}$ mit Radius $r = r_1 + r_2$ konstruieren.



(c) 3. Schritt: Gerade durch Schnittpunkt des Thaleskreis mit Hilfskreis und Zentrum des anderen Kreises $\vec{C_1}$.



(d) 4. Schritt: Gerade in Richtung des Zentrums $\vec{C_2}$ um Länge r_1 verschieben.

(Abb. 7.4): *Konstruktion der inneren Kreistangenten.*

Die Konstruktion dieser ist analog zur Konstruktion der äußeren Tangenten, bis auf die Tatsache, dass der Radius des Hilfskreises über die Summe beider Radien berechnet wird und somit die Formel für den Hilfskreis $k_2^{'}$ lautet:

$$k_2^{'} : (\vec{x} - \vec{C_2})^2 = (r_1 + r_2)^2$$

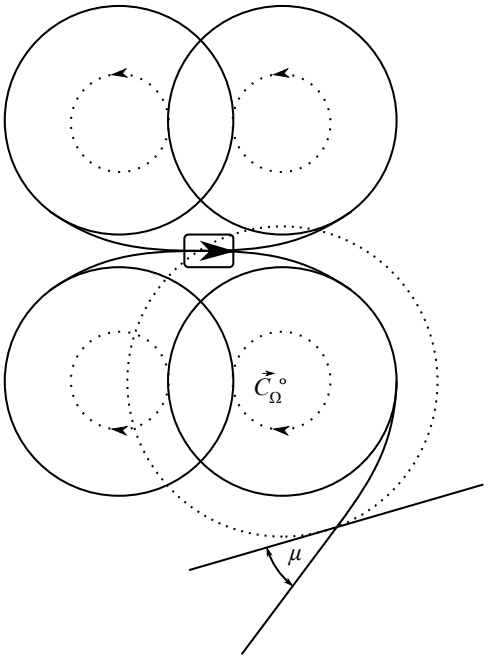
Des Weiteren muss die Gerade in Richtung des Kreismittelpunktes $\vec{C_2}$ verschoben werden und nicht von dem weg, somit wird der Vektor $\vec{D}$ berechnet über:

$$\vec{D} = -r_1 \cdot \frac{\vec{S} - \vec{C_2}}{|\vec{S} - \vec{C_2}|}$$

$$\vec{D} = r_1 \cdot \frac{\vec{C_2} - \vec{S}}{|\vec{C_2} - \vec{S}|}$$

7.2 CONTINUOUS STEERING

Wie bereits in den Motion Primitives (Kapitel 4) behandelt, ist die Verbindung zwischen Gerade und Kreisbahn nicht ruckfrei durchfahrbar und somit nicht optimal für Regler geeignet. Eine bessere Möglichkeit bieten Klothoiden oder Continuous Steering Primitiven, dieses wird im Paper (Fraichard & Scheuer 2004 [7]) behandelt. Die Idee dabei ist, dass ein Fahrzeug mit maximaler Lenkgeschwindigkeit v_{steer} den maximalen Lenkwinkel $\varphi_{\max}$ erreicht und somit, auf die Kreisbahn kommt. Somit werden die Kreise nicht direkt links und rechts an eine Konfiguration angelegt, sondern an das Ende der Continuous Steering Primitive. Da das Erreichen des maximalen Lenkwinkels immer in der gleichen Zeit abläuft ist diese Primitive konstant und muss nur einmal vorberechnet werden (Abb. 7.5).



(Abb. 7.5): *Reeds Shepp zu Continuous Steering.*

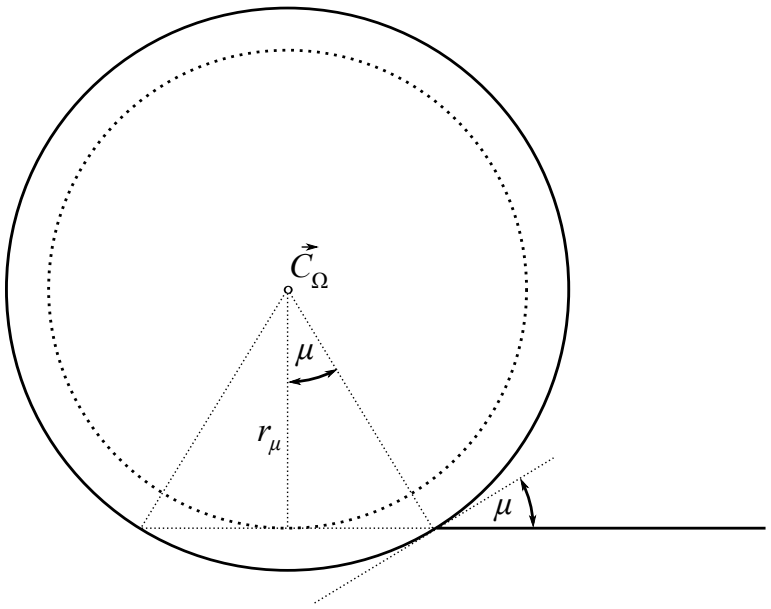
Hat man die Continuous Steering Primitive berechnet und einen Kreis mit dem Radius $r_{\min}$ angehängt, so bekommt man den Mittelpunkt $\vec{C_\Omega}$. Der Abstand von $\vec{C_\Omega}$ zu der Startposition ergibt den Radius r_Ω für den Übergangskreis (hier gestrichelt), an dem der Übergang von der Geraden zu den Continuous Steering Primitiven stattfindet. Der Eintritt in und der Austritt aus dem Übergangskreis geschehen über so genannte μ-Tangenten, weil sie den Winkel μ mit den Tangenten einschließen. Dieser Winkel taucht noch einmal zwischen der y-Achse und der Geraden von Startpunkt zu $\vec{C_\Omega}$ auf.

$$\vec{C_\Omega} = \begin{pmatrix} x_\Omega \\ y_\Omega \end{pmatrix}$$
$$r_\Omega = \sqrt{x_\Omega^2 + y_\Omega^2}$$
$$\mu = \tan^{-1}\left(\frac{x_\Omega}{y_\Omega}\right)$$

7.2.1 KONSTRUKTION: μ-TANGENTEN

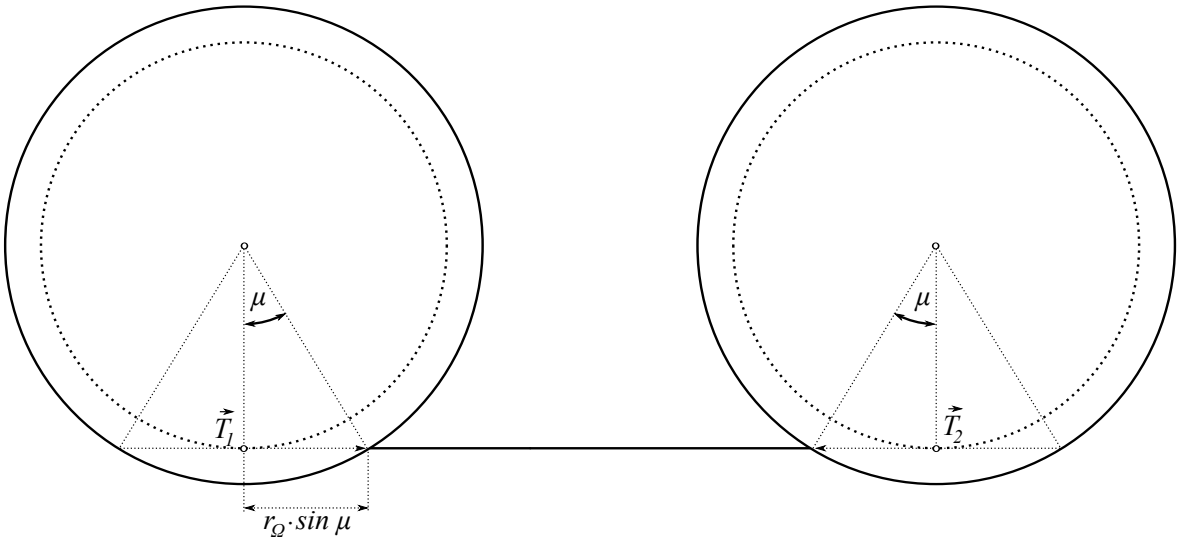
Auf die Berechnung der μ-Tangenten wird im Paper (Fraichard & Scheuer 2004 [7]) leider nicht eingegangen, so dass diese hier geschehen soll. Schaut man sich die folgende Konstruktion an, so erkennt man, dass alle Tangenten des inneren μ-Kreises mit dem Radius r_μ den äußeren Kreis (Radius r_Ω) mit dem Winkel μ verlassen (Abb. 7.6). Dieser Radius berechnet sich über:

$$r_\mu = r_\Omega \cdot \cos\mu$$



(Abb. 7.6): *Konstruktion der μ-Tangente.*

Somit müssen die äußeren und inneren Tangenten mit den μ-Kreisen nach dem bereits beschriebenen Prinzip (Kapitel 7) berechnet werden und nur deren Punkte angepasst werden. Seien $\vec{T_1}$ und $\vec{T_2}$ die gefundenen Schnittpunkte der Tangente mit den μ-Kreisen, so müssen diese Punkte jeweils bis auf den äußeren Kreis verschoben werden. Dies kann entweder über den Schnitt mit den Kreisen geschehen, oder viel einfacher, indem man die Punkte jeweils um den Wert $r_\Omega \cdot \sin \mu$ verschiebt (Abb. 7.7).



(Abb. 7.7): *Anpassung der μ-Tangente an den äußeren Kreis.*

8

FAZIT UND AUSBLICK

Das Bahnplanungsframework bietet eine Möglichkeit mehrere Algorithmen auch kombinatorisch auszuprobieren und deren Verhalten auf Änderung verschiedener Parameter zu untersuchen. Es wurden einige Algorithmen implementiert und diese wurden auf ihre Parametriesierbarkeit untersucht. Eine große Rolle spielten dabei die Heuristikfunktion und die hierarchische Suche, durch die die Kalkulationszeit der Algorithmen verbessert werden konnte. Die Statistikauswertung zeigt Verbesserungsmöglichkeiten im Programm, weshalb auch optimierte Datenstrukturen für das Open und Closed Set eingeführt wurden.

Nun flogt noch ein Ausblick auf die einzelnen Kapitel, was interessante Aussichten und eventuell Verbesserungen bringt, die jedoch im Rahmen dieser Arbeit nicht mehr bearbeitet werden konnten und somit als Denkanstöße für weitere Arbeiten dienen können.

MOTION PRIMITIVES

In Gesprächen mit Herrn Lehr von Götting KG, die sich schon sehr lange mit diesem Thema befasst, kam heraus, dass er nie die kinematischen Eigenschaften des Fahrzeugs voll ausreize, sondern bis nur etwa $\frac{2}{3}$ des Maximums gehe. Dabei bezog er sich vor allen Dingen auf die Lenkgeschwindigkeit. Der Grund dafür sei, dass die Lenkbewegung nicht linear verlaufe, sondern gleichförmig durch einen Motor beschleunigt und später auch wieder abgebremst werde. Bei kleinen Änderungen sei der Motor ständig am Beschleunigen und am Bremsen und erreiche niemals die maximale Lenkgeschwindigkeit, weshalb man etwas geringer beim Berechnen ansetzen solle. Da diese Tatsache aber die Primitiven unnötig groß anwachsen lässt, kann man noch einen Schritt weiter gehen und nicht Continuous Steering Primitiven berechnen, sondern Accelerated Steering Primitiven. Dieses Verfahren wurde bisher aber auch von Herrn Lehr nicht praktiziert.

SAMPLING

Aufgrund der detailarmen Modellierung in der Simulation und den vorgegebenen statischen Szenarien wurde das vorgestellte online Sampling nicht implementiert. Die Hindernisse werden im Format RNDF gespeichert, welches auch die Möglichkeit für Straßen und Verbindungen bietet. Doch es reicht nicht nur die Umgebung zu modellieren, viel wichtiger ist der Algorithmus, der die Primitiven an die Umgebung anpassen muss. Interessante Ansätze liefert (Kelly & Nagy 2003 [17]).

SUCHALGORITHMEN

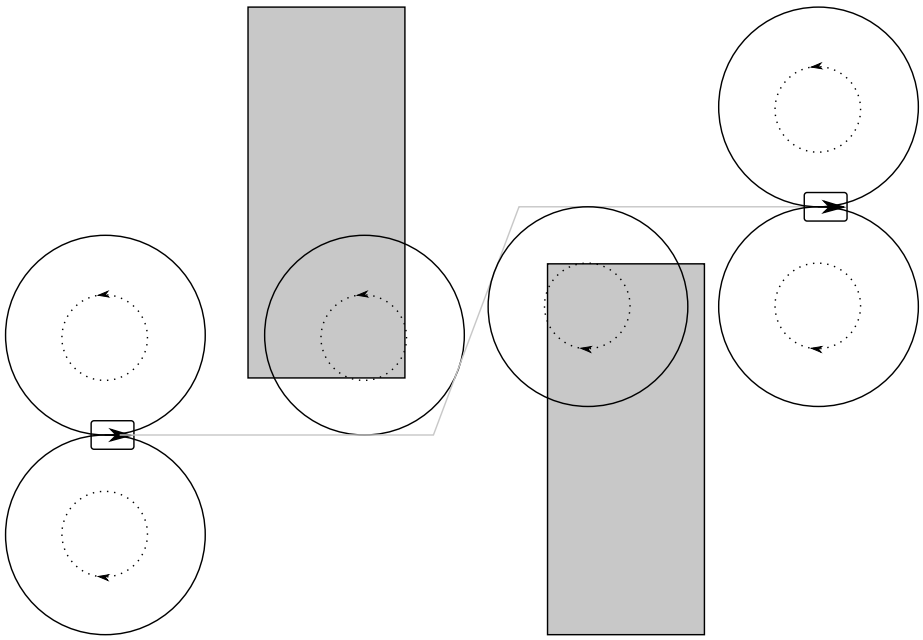
Alle Szenarien fanden in einer statischen Umgebung statt. Erweitert man die Umgebung um Dynamik, so kommen viele weitere Algorithmen in Frage. D* ist einer der bekanntesten dynamischen Algorithmen, der Teile der bereits gefundenen Bahn wiederverwenden kann. D* wird meistens in

unbekannten Gebieten eingesetzt, wo die Hindernisse nicht alle bekannt sind, sondern nur nach und nach erkannt werden und umgeplant werden muss. Aber es muss auch nicht zwangsweise so sein, dass der D* Algorithmus bessere Ergebnisse liefert, im Paper (Hernández 2005 [11]) wurden D* Lite mit A* verglichen und festgestellt, dass die Umgebung zu 50% mit Hindernissen zugestellt werden müsse, damit D* Lite schneller werde als A* und bei sehr kleinen Szenarien D* Lite immer langsamer sei. Es reicht nicht nur das Bahnplanungsframework um Dynamik zu erweitern, es muss einstellbar sein, wann welche Hindernisse neu erkannt werden und die Algorithmen müssen während der Fahrt neu ausgeführt werden. Da die Algorithmen mehrmals während einer Fahrt ausgeführt werden zählen alle Ausführungen zu einer Statistik, so dass die Statistik so erweitert werden muss, dass mehrmalige Ausführungen erfasst werden können.

Auch der Ansatz mit hierarchischer Suche klingt interessant, wenn man wie zuvor angemerkt Straßennetze in die Umgebung modelliert. Wenn vorher auf dem Straßennetz gesucht werden kann, muss ein Algorithmus geschrieben werden, der dem Straßenverlauf folgt und es muss nicht die gesamte Umgebung abgesucht werden.

Reeds Shepp

Reeds Shepp liefert eine gute Möglichkeit zwei Konfigurationen miteinander zu verbinden, kann aber nicht um Hindernisse planen. Im Paper (Fraichard & Scheuer 2004 [7]) wird zum Schluss ein Algorithmus angesprochen, der es erlaubt mit Reeds Shepp eine vollständige Suche um Hindernisse auszuführen. Der Ansatz wäre also mit Theta* den Weg grob vorzuplanen und danach die Punkte miteinander nach Reeds Shepp über Kreise zu verbinden. Die Verbindungen zwischen den einzelnen Theta* Konfigurationen wurden bereits auf Kollisionen untersucht, trotzdem kann das naive hinlegen eines Kreises zur Kollision am Kreis führen. Es gäbe mehrere Möglichkeiten zum durchfahren einer Kurve, diese können alle angeboten werden, wenn die naive Methode nicht klappt (Abb. 8.1).



(Abb. 8.1): *Reeds Shepp Suche.*

GLOSSAR

Nachfolgend sind noch einmal wesentliche Begriffe dieser Arbeit zusammengefasst und erläutert. Eine ausführliche Erklärung findet sich jeweils in den einführenden Abschnitten sowie der jeweils darin angegebenen Literatur. Das im Folgenden im Rahmen der Erläuterung verwendete Symbol $\sim$ bezieht sich jeweils auf den im Einzelnen vorgestellten Begriff, das Symbol $\uparrow$ verweist auf einen ebenfalls innerhalb dieses Glossars erklärten Begriff.

Ablage Als $\sim$ wird der Abstand von der aktuellen $\uparrow$Pose des Fahrzeugs zu dem nähesten Punkt des aktuellen Bahnsegments bezeichnet.

Bahn Eine $\sim$ ist eine Liste von Bewegungsprimitiven, die für ein Fahrzeug abfahrbar sind. Diese kann auch in Form von $\uparrow$Posen vorliegen

dGPS $\sim$ steht für differential Global Positioning System. Ähnlich wie GPS funktioniert $\sim$, hat aber zusätzlich die Information einer Referenzstation, wodurch die Position Centimetergenau berechnet werden kann.

Odometrie Durch $\sim$ ist es möglich über Sensoren, die den Lenkwinkel und die Radumdrehung messen die $\uparrow$Pose des Fahrzeugs zu berechnen. Die $\sim$ ist inkrementell und muss zu Beginn mit einer gültigen $\uparrow$Pose initialisiert werden.

Pfad Ein $\sim$ ist eine Liste von Wegpunkten, die in Form von $\uparrow$Positionen vorliegt.

Pose Eine $\sim$ ist die Kombination von $\uparrow$Position und Orientierung des Fahrzeugs. Diese wird entweder als Tupel (x, y, θ) oder als affine Transformationsmatrix $M \in \mathbb{R}^{3\times3}$ angegeben.

Position Eine $\sim$ beschreibt in der Bahnplanung ein Koordinatentupel (x, y), als Entfernung des Fahrzeugmittelpunkts zum Weltkoordinatenmittelpunkt.

RGBD-Kamera Die Bilder einer $\sim$ beinhalten neben den drei Hautfarben rot, grün und blau auch die Tiefe (engl. depth) für räumliches Sehen.

ABKÜRZUNGEN

ATF	Autonomes Transportfahrzeug
DGPS	Differential Global Positioning System
FTF	Fahrerloses Transportfahrzeug
GPS	Global Positioning System
LGS	Lineares Gleichungssystem
RGB	Red-Green-Blue (Farpkomponenten)
RGBD	Red-Green-Blue-Depth
RNDF	Route Network Definition File
RS	Reeds Shepp
TCP	Transmission Control Protocoll

ABBILDUNGEN

1.1 Testfahrzeuge. 3
1.2 Screenshot der Bahnplanung. 4

2.1 Modell eines Fahrzeugs mit Achsschenkellenkung. 6
2.2 Zustandsquadrupel (x, y, θ, φ) des Ackermannmodells und die Kreisbahn $\mathcal{M}$, bei sich nicht änderndem Lenkwinkel φ. 7
2.3 2D-Laserscanner von SICK 8
2.4 Schematisches Modell der Laserscanner. 8
2.5 Typische Ansichten von Laserscanns. 9
2.6 Struktogramm des gesamten Kollisionstestprozesses 11
2.7 Abstand Punkt Linie über das Kreuzprodukt. 12
2.8 Punkt P_i' der D-Hülle eines Polygons 13
2.9 Struktogramm für einen Bounding Sphere Schnittest 14
2.10 Bounding Sphere eines Polygons. 14
2.11 Vergleich der Bounding Spheres vom naiven Ansatz und dem gewichteten Ansatz. . . 15
2.12 Bounding Spheres einer Motion Primitive. 16
2.13 Bei Bogensegmenten, deren innerer Winkel größer als π ist, muss die Bounding Sphere anders berechnet werden. Sonst entsteht eine Bounding Sphere, die nicht die gesamte Bahn umfasst (rechts). 16
2.14 Schnitttest Gerade mit Polygon. Vereinfacht auf Schnittest zweier Geradenesgmente L und $L_{\mathcal{P}_i}$. 17
2.15 Bahnsegment $\mathcal{M}$ schneidet die D-Hülle (gestrichelt) im Punkt $\vec{K}$, jedoch berührt das Fahrzeug mit der Konfiguration $\mathcal{A}$ das Polygon $\mathcal{P}$ nicht. 18
2.16 Anschauliche Werte zur Herleitung der affinen Positionsmatrix. 18
2.17 Closest point $\vec{P_{\mathcal{A}}}$ und Tangente $\vec{D_{\mathcal{A}}}$ auf der Bahn $\mathcal{M}$, zu der Konfiguration $\mathcal{A}$ 19
2.18 Lenkwinkelkorrektur γ, als Winkel zwischen Vorderachse und dem Vorhaltepunkt $\vec{F_{\mathcal{A}}}$ 20
2.19 Asymptotische Annäherung einer gefahrenen Bahn (rot) zu einer berechneten Bahn (grün). 21
2.20 Messung der Bremsbeschleunigung a. Geschwindigkeit v im Verlauf der Zeit t. . . . 22
2.21 Diskrete Messungen von Fahrzeugkonfigurationen $\mathcal{A}_1$ und $\mathcal{A}_2$ zwischen zwei Bahnsegmenten $\mathcal{M}_1$ und $\mathcal{M}_2$. 24
2.22 Simulationsversuch mit verschiedenen Bahnreglern. Gefahrene Bahn (rot) auf einer vorberechneten Testbahn (grün). 24
2.23 Aufgezeichnete Ablagedaten $dis(l)$ im Verlauf der Länge l in dem Szenario (Abb. 2.22). 25
2.24 Integral über die Ablagefunktion $\int dis(l)dl$ in dem Szenario (Abb. 2.22). 25

3.1 Framework Architektur. 27
3.2 Entwickeltes Framework. 28
3.3 Parameter verschiedener Elemente. 29

3.4 Vehicle Parameters. 29
3.5 Statistikauswertung. 31

4.1 Motion Primitive: Gerade 34
4.2 Motion Primitive: Kreisbogen 34
4.3 Fahrzeug wechselt von einer Geraden auf eine Kreisbahn. Aufgrund physikalischer Lenkbewegung wird nicht sofort der Kreisbahnlenkwinkel eingestellt und das Fahrzeug driftet aus der Kurve. Rot ist der vom Bahnregler gefahrene Pfad und grün der berechnete Pfad. 35
4.4 Änderung der Position, für den Winkel θ und die Distanz Δl. 36
4.5 Graph einer Klothoide mit $\sigma = 1$. 37
4.6 Funktionsgraphen für Lenkeinschlag $\varphi(l)$ und Lenkgeschwindigkeit $v_{\mathrm{steer}}(l)$ einer Klothoide mit $\sigma = 1$. 38
4.7 Graph einer Continuous Steering Primitive (blau) mit $v_{\mathrm{steer}} = 2$, $v = 1$ und $l = \frac{\pi}{4}$, im Vergleich zu einer Klothoide (rot) mit gleichen Fahrzeugparametern. 41
4.8 Verbindungen zwischen Geraden. 42

5.1 Control Space Sampling: Arc-Based search space der Tiefe 2 und $n = 8$. 44
5.2 Suchraummatrix $\mathcal{S}$ des Adaptive search space und deren Elemente. 45
5.3 Control Space Sampling: Adaptive search space der Tiefe 2, Länge $l = 1$ und $n = 8$. 45
5.4 Control Space Sampling: Adaptive search space der Tiefe 2, Länge $l = 1$ und $n = 8$, mit entfernten Redundanzen. 46
5.5 State Lattice Editor. Hier typische Primitiven: Gerade, 90°-Kurve und Wendeklothoide. 47
5.6 Vergleich der Erreichbarkeitsbäume der Tiefe 4 und 5 Nachfolgeelemente. 48
5.7 Vergleich Control Space Sampling mit Online State Space Sampling. **Quelle:** Howard et al. 2008 [14] 49
5.8 Szenario für die Evaluation der Samplingmethoden. Beide Verfahren haben 5 Nachfolgekonfigurationen und die Länge der Primitiven beträgt etwa 3 bis 4 Meter. 50

6.1 Erreichbarkeitsbaum von Adaptive Search Space mit Tiefe 4 und 5 Nachfolgeelementen. 53
6.2 A* mit euklidischem Abstand als Heuristikfunktion. Grau (open set), rot (closed set), grün (Pfad) 55
6.3 Radiale Ausbreitung eines Dijkstra Algorithmus. 58
6.4 Dijkstra und A* in direktem Vergleich 59
6.5 A* mit euklidischem Abstand und gewichtetem Abstand im direkten Vergleich. . . . 60
6.6 A* mit unterschiedlichen Gewichtungen der Heuristik im Vergleich. 61
6.7 Hervorgehobene Nachteile an Heuristik mit gewichtetem Abstand in A*. 62
6.8 Orientierung als Gewichtung. Winkel δ zwischen Orientierung des Fahrzeugs und Ziel. 63
6.9 Beseitigte Nachteile des A* mit gewichtetem Abstand, mithilfe variabler Gewichtung durch Orientierung. 63
6.10 A* mit unterschiedlichen Heuristiken im Vergleich in einem Szenario mit Hindernis. 64
6.11 Kreisbogenabstand als Heuristik. Das Fahrzeug muss minimal den Kreisbogen und die Gerade abfahren, als kürzesten Weg von $\vec{a}$ nach $\vec{b}$. 65

6.12 **Satz des Thales:** Alle Winkel in einem Halbkreisbogen sind rechte Winkel. 66
6.13 Vergleich optimaler Heuristiken in A*. 67
6.14 Vergleich optimaler Heuristiken in A* mit Hindernis. 67
6.15 Arbeitsweise eines Skew Heaps. 71
6.16 Anschauliche Ansicht eines Octrees. 74
6.17 Szenario zum Testen der Datenstrukturen. 75
6.18 Gedankenexperiment zur hierarchischen Suche. (gelb → Closed Set, weiss-grau → Open Set, grün → Pfad, rot → Startposition) 76
6.19 Erreichbarkeitsbäume der Tiefe 4 des Adaptive Search Space und des Grids. 77
6.20 A* Suche auf einem Grid. 77
6.21 Hierarchische Suche: Thera* Gridsuche liefert Zwiscenziele, die über A* angefahren werden. 78
6.22 Engpässe werden von heuristischen Suchen übersehen. 79
6.23 Szenario zur Evaluation. 80
6.24 Zeitdiagramm zu Tabelle (Tab. 6.12). 80
6.25 Anzahl expandierter Knoten Diagramm zu Tabelle (Tab. 6.13). 81
6.26 Weglängendiagramm zu Tabelle (Tab. 6.14). 82

7.1 Problem an heuristischen Suchalgorithmen durch Diskreditierung. (grau → valider Pfad, schwarz → vorberechnete Motion Primitiven) 83
7.2 Verbindung der Kreisbahnen mit Tangenten. 84
7.3 Konstruktion der äußeren Kreistangenten. 86
7.4 Konstruktion der inneren Kreistangenten. 86
7.5 Reeds Shepp zu Continuous Steering. 87
7.6 Konstruktion der μ-Tangente. 88
7.7 Anpassung der μ-Tangente an den äußeren Kreis. 89

8.1 Reeds Shepp Suche. 92

TABELLENVERZEICHNIS

5.1 Messdaten zu (Abb. 5.8). 51

6.1 Messdaten zu (Abb. 6.4). 60
6.2 Messdaten zu (Abb. 6.5). 61
6.3 Messdaten zu (Abb. 6.6). 62
6.4 Messdaten zu (Abb. 6.10). 64
6.5 Messdaten zu (Abb. 6.13). 67
6.6 Messdaten zu (Abb. 6.14). 68
6.7 Vergleich Skew Heap mit einer naiven Liste. 72
6.8 Vergleich Skew Heap mit einer sortierten Liste. 72
6.9 Vergleich Octree mit einer naiven Liste. 74
6.10 Messdaten zu (Abb. 6.17). 75
6.11 Messdaten zu (Abb. 6.21 b) im Vergleich zu Tabelle (Tab. 6.6). 78
6.12 Zeitmessung in *ms* zum Szenario (Abb. 6.23). 80
6.13 Messung der expandierten Knoten zum Szenario (Abb. 6.23). 81
6.14 Erfassung der Weglängen in Metern zum Szenario (Abb. 6.23). 82

LITERATUR

[1] DEAN, T., AND BODDY, M. An Analysis of Time-Dependent Planning. *AAAI-88 Proceedings* (1988), 49–54.

[2] DIJKSTRA, E. W. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1* (1959), 269–271.

[3] DOUGLAS, D. H., AND PEUCKER, T. K. Algorithms for the Reduction of the Number of Points required to represent a digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (Oct. 1973), 112–122.

[4] DUBINS, L. E. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics 79*, 3 (1957), 497–516.

[5] ECKERMANN, E. *Die Achsschenkellenkung und andere Fahrzeug-Lenksysteme*. Deutsches Museum, 1998.

[6] FERGUSON, D., LIKHACHEV, M., AND STENTZ, A. A Guide to Heuristic-based Path Planning. *American Association for Artificial Intelligence* (2005).

[7] FRAICHARD, T., AND SCHEUER, A. From Reeds and Shepp's to Continuous-Curvature Paths. *IEEE Transactions on Robotics 20*, 6 (Dec. 2004), 1025–1035.

[8] FURTUNA, A. A., BALKCOM, D. J., CHITSAZ, H., AND KAVATHEKAR, P. Generalizing the dubins and reeds-shepp cars: Fastest paths for bounded-velocity mobile robots. *2008 IEEE International Conference on Robotics and Automation* (May 2008), 2533–2539.

[9] HART, P. E., NILSSON, N. J., AND RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics ssc-4* (1968), 100–107.

[10] HE, H., ZHANG, Y., AND LIU, H. A Method Used To Dynamic Scene Management Based On Octree. *Computer Graphics, Imaging and Visualisation (CGIV 2007)*, Cgiv (Aug. 2007), 16–21.

[11] HERNÁNDEZ, C., BAIER, J., URAS, T., AND KOENIG, S. Position Paper : Incremental Search Algorithms Considered Poorly Understood. *IEEE Transaction on Robotics and Automation 21*, 3 (2005).

[12] HORN, B. K. P. The Curve of Least Energy. *ACM Transactions on Mathematical Software 9*, 4 (1983), 441–460.

[13] HORVITZ, E. Problem-Solving Design: Reasoning about Computational Value, Tradeoffs, and Resources. *Proceedings of the Second Annual NASA Research Forum* (1987), 1–18.

[14] HOWARD, T. M., GREEN, C. J., KELLY, A., AND FERGUSON, D. State Space Sampling of Feasible Motions for High-Performance Mobile Robot Navigation in Complex Environments. *Journal of Field Robotics 25 25*, 1 (2008), 325–345.

[15] HRBÁČEK, J., RIPEL, T., AND KREJSA, J. Ackermann mobile robot chassis with independent rear wheel drives. *EPE-PEMC 2010* (2010), 46–51.

[16] KANAYAMA, Y., AND HARTMAN, B. I. Smooth local path planning for autonomous vehicles. In *Proceedings, 1989 International Conference on Robotics and Automation* (1989), IEEE Comput. Soc. Press, pp. 1265–1270.

[17] KELLY, A., AND NAGY, B. Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control. *The International Journal of Robotics Research 22*, 7-8 (July 2003), 583–601.

[18] KIRCH, T. 3D-Lasertriangulation, 2007.

[19] KOENIG, S., AND LIKHACHEV, M. Improved Fast Replanning for Robot Navigation in Unknown Terrain. *Proceed- ings of the IEEE International Conference on Robotics and Automation (ICRA)* (2002), 1–8.

[20] KOMORIYA, K., TACHI, S., AND TANIE, K. A method of autonomous locomotion for mobile robots. *Advanced Robotics 1*, 1 (1986), 3–19.

[21] KOWALK, W. Animierte Algorithmen. In *Skript zu Lehrveranstaltung Algorithmen und Datenstrukturen* (2005), Uni Oldenburg, pp. 1–292.

[22] KOWALK, W. Vektorrechnung. In *Skript zu Lehrveranstaltungen über Computer Graphik* (2010), Uni Oldenburg, pp. 1–27.

[23] LAMIRAUX, F., BONNAFOUS, D., AND LEFEBVRE, O. Reactive Path Deformation for Nonholonomic Mobile Robots. *IEEE Transactions on Robotics 20*, 6 (Dec. 2004), 967–977.

[24] LAVALLE, S. M. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.

[25] LIKHACHEV, M., FERGUSON, D., GORDON, G., STENTZ, A., AND THRUN, S. Anytime Dynamic A*: An Anytime, Replanning Algorithm. *Proceedings of the Interna- tional Conference on Automated Planning and Scheduling (ICAPS)* (2005), 1–10.

[26] LIKHACHEV, M., GORDON, G., AND THRUN, S. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. *Advances in Neural Information Processing Systems* (2003), 1–8.

[27] MCNAUGHTON, M., URMSON, C., DOLAN, J. M., AND LEE, J.-W. Motion planning for autonomous driving with a conformal spatiotemporal lattice. *2011 IEEE International Conference on Robotics and Automation 1* (May 2011), 4889–4895.

[28] MILLINGTON, I., AND FUNGE, J. *Artificial Intelligence for Games*. Elsevier, 2009.

[29] NASH, A., DANIEL, K., AND KOENIG, S. Theta*: Any-Angle Path Planning on Grids, 1968.

[30] PIVTORAIKO, M., AND KELLY, A. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Sept. 2011), IEEE, pp. 2172–2179.

[31] PIVTORAIKO, M. N. *Differentially Constrained Motion Planning with State Lattice Motion Primitives*. No. February in 1. The Robotics Institute, 2012.

[32] REEDS, J. A., AND SHEPP, L. A. Optimal Paths for a Car that goes both Forwards and Backwards. *Pacific Journal of Mathematics 145*, 2 (1990), 367–394.

[33] RUFLI, M., AND SIEGWART, R. On the Design of Deformable Input- / State-Lattice Graphs. *2010 IEEE International Conference on Robotics and Automation* (May 2010), 3071–3077.

[34] SHIN, D. H., AND SINGH, S. Path Generation for Robot Vehicles Using Composite Clothoide Segments. Tech. rep., Carnegie Mellon University, 1990.

[35] SHKEL, A. M., AND LUMELSKY, V. J. On Calculation of Optimal Paths with Constrained Curvature : The Case of Long Paths *. *International Conference on Robotics and Automation* (1996), 3578–3583.

[36] SLEATOR, D. D., AND TARJAN, R. E. Self-Adjusting Heaps. *SIAM Journal on Computing 15*, 1 (Feb. 1986), 52–69.

[37] STENTZ, A. The Focussed D* Algorithm for Real-Time Replanning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August (1995), 1–8.

[38] THRUN, S., AND BÜCKEN, A. Integrating Grid-Based and Topological Maps for Mobile Robot Navigation. *Proceedings of the Thirteenth National Conference of Artificial Intelligence AAAI*, August (1996), 944–950.

[39] TOUNSI, M., AND LE CORRE, J. Trajectory generation for mobile robots. *Mathematics and Computers in Simulation 41*, 3-4 (July 1996), 367–376.

[40] TSUMURA, T., FUJIWARA, N., SHIRAKAWA, T., AND HASHIMOTO, M. Automatic guidance of vehicles following the route stored in the memory. *Advanced Robotics 1*, 3 (1986), 201–210.

[41] ZILBERSTEIN, S., AND RUSSEL, S. Approximate reasoning using anytime algorithms. *Imprecise and Approximate Computation* (1995), 1–20.

VERSICHERUNG

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 27. Februar 2013

Jürgen Boger