Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Bachelorarbeit

Pfadplanung in dynamischer Umgebung bei mobilen autonomen Robotern

Vorgelegt von

Philipp Freick

Leipzig, Februar 2014

Betreuender Professor: Prof. Dr. rer. nat. habil. Siegfried Schönherr

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Leipzig, den 25.02.2014

_____

Kurzfassung

Partizipanten des RoboCup Wettbewerbs sehen sich mit einem breiten Spektrum von Herausforderungen in den Bereichen der Künstlichen Intelligenz und der Robotik konfrontiert. Die flexible autonome Bewegungsfähigkeit ist dabei eine grundlegende Fertigkeit um unterschiedlichste Aufgabenstellungen bewältigen zu können. In dieser Arbeit soll die Funktionsweise eines Pfadplaners beschrieben werden, der auf dem Konzept des erweiterten Sichtbarkeitsgraphen basiert. Schwerpunkte sind hierbei die Steigerung der Laufzeiteffizienz und die Behandlung nicht vorgesehener Spielkonstellationen. Die Funktions- und Leistungsfähigkeit wird durch eine Demo-Applikation präsentiert. Abschließend werden die Resultate diskutiert und Möglichkeiten zur Erweiterung und Verbesserung des Verfahrens aufgezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Prolog

Die durch den technologischen Fortschritt bewerkstelligte Übertragung der Arbeit vom Menschen auf die Maschine ist ein seit der Antike andauernder und noch nicht absehbar endender Prozess. Mit der Entwicklung flexibel programmierbarer elektronischer Schaltkreise scheint die Automatisierung in immer schnellerer Geschwindigkeit stattzufinden. So ist die Summe an Innovationen in den letzten 100 Jahren wohl weitaus größer als in den 1000 Jahren davor. Aktuelle Bestrebungen zielen auf die vollständige und allumfassende Automatisierung in allen Lebensbereichen ab, um den Arbeitsbereich des Menschen von der Produktion, welche oft gefährliche und monotone Arbeiten erfordert, auf die Planung, Administration und Wartung zu lenken. Um dies zu erreichen, ist die Entwicklung intelligenter Roboter- und Drohnensysteme nötig. Im Status Quo ist eine Welt vorzufinden, die auf die Bedürfnisse und Physis der Menschen angepasst wurde. Neu entwickelte Systeme sollen sich dabei in diesem Dogma einordnen, um eine möglichst hohe Akzeptanz der Nutzer zu erfahren. Die Förderung einer für den Roboter angepassten Umgebung wäre dahingegen für den Menschen kontraintuitiv und zudem (finanziell) äußerst aufwändig. So werden Roboter vorerst in leicht überschaubaren, gut kontrollierbaren Umgebungen entwickelt und getestet, um dann mit zunehmender Anpassungsfähigkeit der Systeme für weitaus komplexere Anforderungen eingesetzt zu werden. Auf dem Weg hin zur Schaffung eines intelligenten Systems, welches sich in der Lage sieht, verschieden modellierte Lösungsprozesse selbstständig anwenden zu können und ggf. selbst neue Lösungsmöglichkeiten zu entwickeln, stellt die Pfadplanung einen elementaren Bestandteil dieser Entwicklung dar.

Erst durch eine ausgereifte Navigationsfähigkeit wird es in naher Zukunft möglich sein, das Einsatzspektrum der Systemklasse von autonomen mobilen Systemen im erheblichen Maße auszuweiten. So werden schon in wenigen Jahrzehnten aus ökonomischen Gründen Reinigungsdrohnen, autonome Kraftfahrzeuge sowie Drohnen in der Produktionstechnik zum Transport von Waren ein gewohntes Bild sein[1]. Darüber hinaus existieren schon heute Problemdomänen, in denen ausschließlich autonom handelnde Systeme arbeiten können. Ein Beispiel stellt die Raumsonde Curiosity dar, die durch ihre enorme Distanz

[1] Schon heute werden im begrenzten Maße Zulieferaufträge innerhalb von Produktions- und Lagerstätten von autonomen Dronen übernommen. Beispiele hierfür sind Amazons Warenlager in Seattle und das Leipziger BMW-Werk.

zur Erde und der damit einhergehenden Verzögerung der Funkübertragungen nicht ferngesteuert werden kann.

Ein partieller Aspekt der Schöpfung selbstständig agierender Systeme ist die Konstruktion humanoider Roboter. Inspiriert vom menschlichen Abbild, zeichnen sich diese meist durch das Vorhandensein zweier Beine, einem Torso und einer Vielzahl von Gelenken, die in ihrer Anordnung dem menschlichen Organismus ähneln, aus.

Humanoide Roboter erscheinen in vielfältigen Gebieten einsetzbar und so verwundert es kaum, dass auch global agierende Unternehmen diesem Trend folgen und mit massivem Aufwand Forschung betreiben. Darüber hinaus hat sich eine wissenschaftliche Gemeinde herausgebildet, die sich für eine kooperative Entwicklung und dem Austausch von Forschungsergebnissen verpflichtet sieht: die RoboCup-Plattform. Seite 2009 ist auch das Nao-Team der HTWK Leipzig ein Teil der RoboCup-Gemeinschaft und agiert als Partizipant der Standard Plattform Liga, um somit ebenfalls einen beständigen Anteil am technischen Fortschritt der Liga beizutragen. Die vorliegende Arbeit soll einen Teil dieser Bestrebungen darstellen.

1.1 Motivation

Die Erfahrungen der letzten Jahre haben deutlich zum Ausdruck gebracht, dass eine fehlende Pfadplanung ein erhebliches Defizit im spielerischen Vermögen der Roboter darstellt. Zum jetzigen Zeitpunkt ist keine vergleichbare Implementierung im Code des Nao-Team HTWK Leipzig vorzufinden. Dies ist damit zu begründen, dass erst im Zuge der Entwickelung weitreichender Grundlagen und der einhergehenden Implementierung erweiterter Softwarekonzepte die Möglichkeiten dafür geschaffen wurden. So ist die Erkennung und Lokalisierung[2] gegnerischer Roboter erst seit dem Jahr 2013 ein Bestandteil der Roboter-Software. Durch eine fehlende Pfadplanung ergeben sich im Detail folgende spielerische Defizite:

- In der Initialisierungsphase[3] – welche vor der eigentlichen Spielphase stattfindet - haben die Roboter 45 Sekunden lang die Möglichkeit, zu vorher selbst ausgewählten Anstoßpositionen zu laufen. Ist ein Roboter aufgrund der Behinderung durch einen Mitspieler nicht in der Lage den Zielpunkt zu erreichen, wirkt sich dies nachteilig auf das Spielgeschehen aus. So können sich zum Beispiel unbefriedigend große Distanzen zum Ball, ein erheblicher Abstand zu einer beliebig anzustrebenden

[2] Lokalisierung bezeichnet die Fähigkeit eines autonomen mobilen Roboters, seine eigene bzw. die Position anderer Roboter in der unmittelbaren Umgebung festzustellen. Kann ein Roboter keine präzise Ortsbestimmung durchführen bzw. sind die angenommenen Koordinaten fehlerhaft, so wird der Roboter als delokalisiert bezeichnet.

[3] Im englischen Regelwerk als „Ready State" bezeichnet [Rob13].

Position in der gegnerischen Spielhälfte oder eine mangelhafte Deckungsmöglichkeit des Tors ergeben.

- Schon das kurzweilige Beobachten des Spielgeschehens macht deutlich, dass die meisten Stürze auf die Kollisionen mit anderen Robotern zurückzuführen sind. Für die Zeit des Wiederaufstehens und der anschließenden Orientierungs- und Lokalisierungsphase ist der Roboter praktisch vom Spiel ausgeschlossen[4]. Vor allem im zentralen Mittelfeld kommt es zu Zweikämpfen und Zusammenstößen der Roboter. Dies ist im besonderen Maße hinderlich, da ein im Mittelkreis gestürzter Roboter zur Delokalisierung neigt, wodurch er im Anschluss gegen das eigene Team spielt.
- Die Destabilisierung gegnerischer Roboter durch direkten Kontakt kann als Regelverstoß geahndet werden[5]. Als Sanktion ist vorgesehen, dass der Roboter für 45 Sekunden vom Spielgeschehen entfernt wird. Darüber hinaus wird beim vierten, sechsten, achten, zehnten und zwölften Regelverstoß wegen „Pushings" der entsprechende Roboter bis zum Ende der Halbzeit von der Partie disqualifiziert. Dies kann somit dazu führen, dass eine Mannschaft komplett vom Spiel entfernt werden muss.
- Es erscheint als logisch und offensichtlich, dass die Selektion günstiger Routen generell zu einem schnelleren Erreichen von Zielpositionen führen kann. Durch den Gewinn einer höheren Spieldynamik sind damit (situationsbedingt) auch flexiblere Spielzüge realisierbar, wie etwa das Freilaufen zur Passannahme oder ein Wechsel von Verteidigungs- in Angriffspositionen und umgekehrt.

1.2 Zielsetzung und Abgrenzung

Ziel ist die Implementierung eines Java basierten Pfadplanungskonzepts. Der Software-Prototyp soll einen Roboter dazu befähigen, sich zu jedem beliebigen Zielpunkt auf dem Spielfeld zu bewegen, ohne dabei mit anderen Robotern zu kollidieren.

Die Softwarelösung soll möglichst effizient sein, da nur begrenzte Rechenkapazitäten zur Verfügung stehen und die Roboter innerhalb kürzester Zeit auf Änderungen in der Spielfeldumgebung reagieren müssen. Es ist zudem gewünscht, dass die erzeugten Pfade einen harmonische Form aufweisen. Die Validität der Routenberechnung soll durch eine Visualisierung der Ergebnisse demonstriert werden.

[4] Das Aufrichten des Roboters aus einer waagerechten Position dauert in etwa sieben Sekunden. Des Öfteren wird ein Roboter in dieser Bewegung von Mitspielern oder zu heiß gewordenen Motoren gehemmt, was eine Vervielfachung der Zeit um den Faktor zwei bis drei bewirken kann.

[5] Im Regelwerk wird dies als „Player Pushing" bezeichnet.

Nicht Bestandteil der Arbeit wird die Integration der Software auf die Roboterplattform sein, da die Portierung der essentiellen Multi-Target Tracking Softwarekomponente zum gegenwertigen Zeitpunkt noch aussteht. Ebenfalls keine Beachtung findet die Koordination der verschiedenen Roboter-Routen, was in gängiger Literatur auch als Multidimensionale-Pfadplanung bezeichnet wird. Somit ist es nicht möglich, Pfade auszuschließen, die zwei oder mehr Roboter zur gleichen Zeit an die gleiche Position führen[6]. Das meiden des eigenen Strafraums[7] und die Navigation in Zweikämpfen, werden ebenfalls nicht von Interesse dieser Arbeit sein, da bereits andere Softwaremodule dieses Verhalten steuern. Ferner sind schon Komponenten für die Sensorfilterung, der Positionsfindung und der Kartenerstellung vorhanden und brauchen demnach nicht weiter berücksichtig werden.

1.3 Aufbau der Arbeit

In diesem Abschnitt soll eine kurze Inhaltsangabe der Kapitel für einen umfassenden Überblick der Bachelorarbeit sorgen. Kernpunkte der Arbeit sollen hierbei erklärt und in Zusammenhang mit der Aufgabenstellung gesetzt werden.

Im zweiten Kapitel werden die theoretischen Grundlagen dieser Arbeit erörtert. Zunächst wird der RoboCup mit seinen verschiedenen Ligen und deren Eigenheiten vorgestellt, um einen Einblick in das wissenschaftliche Umfeld zu erhalten. Es folgen Details zu den verwendeten Robotern und zur Standard Plattform Liga, um einen Ausgangspunkt für die spätere Konzeption eines autonomen Pfadplanungsagenten zu schaffen. Um die unterschiedlichen Aspekte der in dieser Arbeit favorisierten Lösung besser einordnen und beurteilen zu können, sollen zudem weitere Verfahren zur Navigation und signifikante Bewertungskriterien vorgestellt werden.

Im dritten Kapitel wird das grundlegende Verfahren des erweiterten Sichtbarkeitsgraphen detaillierter erörtert und in verschiedene Teilprobleme aufgelöst. Diese Teilprobleme stellen Schlüsselkomponenten des Verfahrens dar. Das konkrete Design dieser Komponenten soll hierbei erarbeitet und abgegrenzt werden. Um bewusste Entwurfsentscheidungen zu offenbaren, sollen relevante Lösungsmöglichkeiten festgehalten und gegenübergestellt

[6] In [Our04] werden diverse Algorithmen zur Multidimensionalen-Pfadplanung beschrieben. Die Autorin verzichtet dort ebenfalls auf die Implementierung solch eines Algorithmus und weist als Begründung darauf hin, dass Lösungen für dieses Entscheidungsproblem als PSPACE-Komplex zu klassifizieren sind. Eine Entscheidung kann somit durch polynomiellem Speicherverbrauch getroffen werden, abhängig von der Anzahl an Robotern. Die Lösung für dieses spezielle Problem ist somit als durchaus diffiziel einzustufen, wobei das Leistungsspektrum der Roboter nicht im angemessenen Maße erweitert wird.

[7] Laut Regelwerk stellt das Betreten des eigenen Strafraums einen Regelverstoß dar, welcher als „Illegal Defending“ bezeichnet wird. Ausgenommen sind hiervon lediglich der Torwart und Spieler der gegnerischen Mannschaft.

werden, wenngleich diese nicht alle in die finale Implementierung einfließen. Das Kapitel enthält zudem eine theoretische Laufzeitabschätzung und wird durch praktische Laufzeitmessungen komplettiert.

Den Abschluss dieser Arbeit stellt das vierte Kapitel mit einer Zusammenstellung und Bewertung der erzielten Ergebnisse dar. Die entwickelte Lösung soll den im ersten Kapitel erhobenen Anforderungen gegenübergestellt und validiert werden. Hierfür sollen nochmals Vor- und Nachteile der erzielten Implementierung herangezogen werden. Zudem findet eine Auseinandersetzung mit denkbaren Erweiterungsmöglichkeiten statt.

2 Grundlagen

2.1 Robocup

Das Propagieren eines Standardproblems in der wissenschaftlichen Gemeinde bietet zumeist vielerlei Vorteile. So ist nicht nur der kompetitive Charakter zum Entwickeln leistungsfähigerer Methoden erheblich innovationsfördernd. Vielmehr wird somit eine Plattform zum wissenschaftlichen Austausch geschaffen, was verschiedenste Lösungsansätze besser vergleichbar macht.

Bis ins Jahr 1997 galt das Schachspiel als Standardproblem im Bereich der Künstlichen Intelligenz. Forschungsbemühungen zielten zum damaligen Zeitpunkt auf die Entwicklung eines Systems ab, das den amtierenden Schachweltmeister unter Wettkampfbedingungen schlagen kann. Der Entwicklungsstand Künstlicher Intelligenz wurde zumeist mit den Fähigkeiten moderner Schachcomputer assoziiert. Doch nicht allein aus Gründen der Popularität wurde Schach als Standardproblem gewählt. Vielmehr bietet das Schachspiel die Eigenschaft, als isoliertes deterministisches System modelliert werden zu können. Unvorhersehbare Geschehnisse können somit ausgeschlossen werden. Wenngleich die enorm große Anzahl zulässiger Züge genug Komplexität bietet, um intelligentes Verhalten imitieren zu können.

Als Computerschach Mitte der 90er Jahre seinen Höhepunkt mit dem Sieg gegen den amtierenden Schachweltmeister Garri Kimowitsch Kasparow[1] erlebte, gewann nun der Roboterfußball als neues Standardproblem schnell an Bedeutung. Als Forschungsgegenstand kombiniert er nicht nur verschiedenste Bausteine aus den Bereichen der Künstlichen Intelligenz und der Robotik miteinander, sondern bietet auch durch sein dynamisches Umfeld eine Komplexität, die in weiten Teilen der realen Welt entspricht. Hier erzielte Forschungsergebnisse können somit auch mit hoher Wahrscheinlichkeit in anderen Bereichen wiederverwendet werden. Eine signifikante Charakteristik des Roboterfußballs ist die interdisziplinäre Zusammenarbeit verschiedenster Forschungsgebiete[2].

[1] Garri Kimowitsch Kasparow wurde von dem von IBM entwickelten Schachcomputer Deep Blue geschlagen. Deep Blue war kein lernendes System und konnte sich somit weder live dem gegnerischen Spieler anpassen, noch seine Strategie im Spielverlauf ändern. Anpassungen am strategischen Verhalten mussten immer noch eigenhändig in das System eingepflegt werden. Der Rechner zog seine Spielstärke vielmehr aus dem ihn zur Verfügung stehenden, enorm großen Rechenkapazitäten.

[2] So ist es erforderlich, dass Lösungen für folgende Punkte erarbeitet werden: Motorik, Sensorik, Planen,

Die Idee der Fußball spielenden Roboter wurde erstmals 1992 von Alan Mackworth[3] in der Veröffentlichung *On Steering Robots* thematisiert. Bereits 1997 fand dann die erste Weltmeisterschaft in Nagoya, Japan als Teil der *International Joint Conference on Artificial Intelligence* (IJCAI) KI-Konferenz statt. Seitdem finden sich Jahr für Jahr verschiedenste Forschergruppen aus allen Teilen der Welt zusammen, um sich in den unterschiedlichen Ligen zu messen und um auf den lokalen Kongressen Erfahrungen und Erkenntnisse aus den Bereichen der Künstlichen Intelligenz und der Robotik auszutauschen. Das ambitionierte Ziel der RoboCup Initiatoren ist bis zum Jahr 2050 den amtierenden Fußball-Weltmeister besiegen zu können. Die Zeitspanne ist dabei vergleichbar mit dem Aufkommen des Computerschachs als Standardproblem, bis zum Sieg von Deep Blue gegen Garri Kimowitsch Kasparow.

2.1.1 Robocup Ligen

Innerhalb des RoboCup Wettbewerbs existieren verschiedene Ligen, die durch ihre unterschiedlichen Charakteristika zur Arbeit in disparaten Forschungsschwerpunkten motivieren. Die zum gegenwärtigen Zeitpunkt bestehenden Ligen sollen im Folgenden erläutert werden.

2D- und 3D-Simulation

In den Simulationsligen treten virtuelle Fußballspieler[4] auf einem simulierten Fußballfeld gegeneinander an. Die einzelnen Spieler agieren dabei autonom und kommunizieren über einen zentralen Soccer-Server. Dieser Soccer-Server verwaltet den Spielablauf, stellt Sensordaten zur Verfügung und nimmt Handlungsentscheidungen der einzelnen Spieler entgegen. Die Weltmodellierung berücksichtigt dabei unter anderem den eingeschränkten Sichtbereich der Spieler und das ein Spieler nur über kurze Distanzen mit voller Geschwindigkeit laufen kann. Teams der 2D-Simulationsliga können sich aufgrund fehlender mechanischer Problemstellungen und der hohen Anzahl an teilnehmenden Agenten vollends auf die Konstruktion ausgeklügelter Multi-Agenten-Systeme konzentrieren (siehe Abbildung 2.1a). Teilnehmer der 3D-Simulationsliga müssen zusätzlich eine stabile Roboter-Kinematik für die simulierte Physik-Engine entwickeln (siehe Abbildung 2.1b). Ein Vorteil dieser beiden Klassen ist, dass durch den Entfall jeglicher Roboter-Hardware die finanziellen Einstiegshürden sehr gering sind.

Lernen, reaktives Verhalten, Agenten-Architektur, Multi-Agenten-Verwaltung, Selbstlokalisierung und Pfadplanung.

[3] Alan Mackworth gilt als Gründungsvater des RoboCup und war ehemals Präsident der Association for the Advancement of Artificial Intelligence.

[4] Die Spieler werden auch als Agenten bezeichnet.

(a) 2D-Simulationsliga [R2Shr] (b) 3D-Simulationsliga [RI2hr]

Abb. 2.1: Zwei Screenshots aus den Simulationsligen

Small Size

Die Small Size Liga zeichnet sich durch ein äußerst dynamisches Spielgeschehen aus. Teams treten dabei mit selbst entwickelten Robotern an, die in ihrer Größe auf einen Durchmesser von 18 cm und eine Höhe von 15 cm beschränkt sind (siehe Abbildung 2.2). Die Agilität der Roboter wird überwiegend durch die Verwendung von omnidirektionalen Antrieben bewerkstelligt. Sensordaten werden dabei zentral von einem Server bereitgestellt, der das Spielgeschehen über zwei aufgestellte Kameras perzipiert. Die Steuerungs- und die Handlungsanweisungen beziehen die einzelnen Roboter zum größten Teil von einem dedizierten System per WiFi, welches die Sensordaten auswertet und daraus Schlussfolgerungen zieht. Forschungsschwerpunkte in dieser Liga sind die Verwendung von Multi-Agenten-Systemen und die Erstellung eines robusten und wendigen Roboter-Designs.

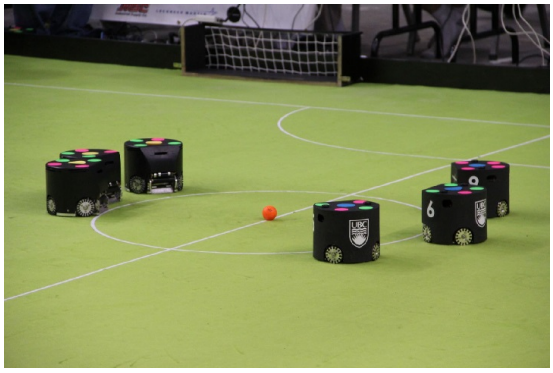


Abb. 2.2: Small Size Liga [ATFhr]

Middle Size

Die Roboter in dieser Liga sind Eigenentwicklungen der jeweiligen Teams (siehe Abbildung 2.3). Die Abmaße der Roboter sind dabei mit einer Höhe von maximal 80 cm und

einem Durchmesser von bis zu 52 cm um einiges üppiger als die Roboter der Small Size Liga. Der Antrieb erfolgt zumeist omnidirektional, wobei die Datenverarbeitung und Sensorik auf den jeweiligen Robotern stattfindet. Die Roboter handeln somit im Gegensatz zur Small Size Liga vollständig autonom. Spielzüge mit anderen Teamteilnehmern können dabei untereinander per WiFi abgestimmt werden. Typisch für diese Liga ist die Verwendung von omnidirektionalen Kameras, um eine 360° Sicht zu ermöglichen. Die Schwerpunkte der Entwicklung sind identisch mit denen aus der Small Size Liga, mit dem Unterschied, dass in der Middle Size Liga das Multi-Agenten-System anhand einer verteilten Wissensbasis arbeitet.

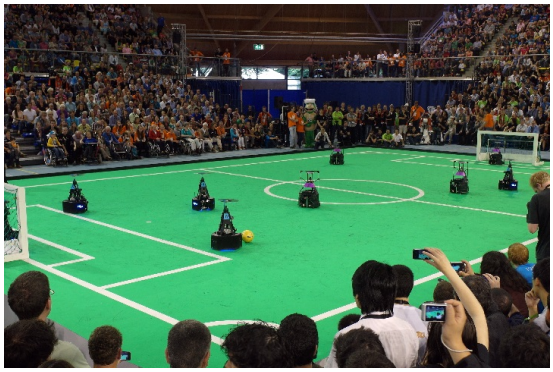


Abb. 2.3: Middle Siza Liga [MTUhr]

Standard Plattform

Anders als in den bisher vorgestellten Ligen treten hier humanoide Roboter gegeneinander an. Als besonderes Merkmal dieser Liga sei der Einsatz baugleicher Roboter des Models Nao genannt (siehe Abbildung 2.4). Hardwareseitige Modifikationen sind untersagt. Durch den Gebrauch identischer Hardware lassen sich spielerische Vorteile nicht mehr mittels höherwertiger und zumeist auch teurer Hardware erkaufen. Die Teams können sich somit gänzlich auf die Entwicklung leistungsfähiger Softwarelösungen fokussieren. Die Verwendung identischer Roboter erleichtert zudem den direkten Vergleich dieser Softwarelösungen und senkt darüber hinaus die Einstiegshürde in die Liga. Auch in dieser Liga agieren die Roboter vollständig autonom und müssen somit Handlungsentscheidungen selbstständig treffen. Eine Kommunikation unter den Robotern ist per Funk möglich. Kern der untersuchten Forschungsfragen liegen hier in der bipeden Fortbewegung[5], der Ballmanipulation und der strategischen Koordination im Team.

[5]Bipedie ist die Fortbewegung durch Gehen oder Hüpfen auf zwei Beinen.

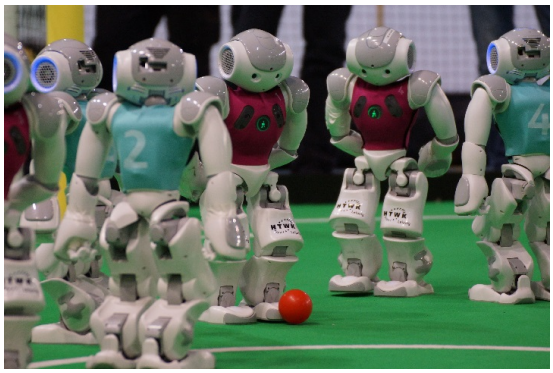


Abb. 2.4: Standard Plattform Liga

Humanoid

In den Humanoid Ligen treten ebenfalls Roboter mit menschenähnlicher Körperform an. Auch diese Roboter sind wie in der Small- und Middle Size Liga Eigenentwicklungen, wobei eine Einschränkung bezüglich der zu verwendeten Sensoren stattfindet. So ist es nur gestattet Sensoren zu implantieren, die in ihrer Art der Wahrnehmung und der Positionierung im Körper dem menschlichen Körper nachempfunden sind. Die Installation von Ultraschall- oder Laserentfernungsmesser ist demzufolge nicht legitim. Roboter der Humanoid Liga sind entweder komplette Selbstentwicklungen oder basieren auf Modifikationen von Standardrobotern. Die Steuerung erfolgt hierbei wieder komplett autonom. Die Wettbewerbe lassen sich in drei Größenklassen einordnen. Es gibt die Kid Size Liga (siehe Abbildung 2.5a) mit Robotern kleiner 60 cm, die Teen Size Liga (siehe Abbildung 2.5b) mit Robotern zwischen 100 und 120 cm und die Adult Size Liga (siehe Abbildung 2.5c) mit Robotern größer 130 cm. Die Entwicklungsschwerpunkte liegen in der cleveren Konstruktion eines agilen Roboter-Designs, der Generierung stabiler, bipeder Fortbewegungsabläufe, der Ballmanipulation und der Koordination des Mannschaftsspiels.

(a) Kid Size Liga [R20hr] (b) Teen Size Liga [R2Thr] (c) Adult Size Liga [VRrhr]

Abb. 2.5: Momentaufnahmen von RoboCup Spielen aus diversen Humanoid Ligen

Neben dem Fußball gibt es noch weitere Forschungsdomänen innerhalb des RoboCups, welche hier noch kurz der Vollständigkeit halber erwähnt werden sollen. So beschäftigen

sich die RoboCup Rescue Teilnehmer mit der Entwicklung von Robotern, die in Katastrophengebieten bzw. für Notfälle eingesetzt werden können. RoboCup@Home widmet sich dem Einsatz von Robotern im alltäglichen Gebrauch, wobei hier die Interaktion zwischen Mensch und Maschine im Vordergrund steht. Zur Nachwuchsförderung wurde für Partizipanten unter 20 Jahre die Klasse RoboCup Junior etabliert. In dieser können ausschließlich Jugendliche in den drei Subligen Junior Dance, Junior Rescue und Junior Soccer gegeneinander konkurrieren.

2.1.2 Standard Plattform Liga (SPL)

Die in der vorliegenden Arbeit enthaltenen Lösungsansätze zur Pfadplanung wurden auf Eigenheiten der Liga und Basis des SPL Reglements entwickelt und abgestimmt. Im Folgenden sollen daher die Spielfeldumgebung und der Nao-Roboter als Entwicklungsplattform genauer beschrieben werden.

Um zu gewährleisten, dass an der SPL beteiligte Teams identische Voraussetzungen haben, erfolgt sowohl der Spielablauf als auch die Gestaltung des Spielfeldes nach fest definierten Richtlinien. Alljährliche Anpassungen des Regelwerkes führen zu einer gezielten Weiterentwicklung in verschiedenen Forschungsschwerpunkten der SPL und stellen teilhabende Mannschaften immer wieder vor neuen Herausforderungen. Als Beispiel sei hier die kontinuierliche Vergrößerung der Spielfeldgröße und der zunehmenden Anzahl teilnehmender Roboter genannt. Da sich unter diesen Regeländerungen ein allzu „egoistischer“ Spielstil der Roboter mehr und mehr als nachteilig erwies, führte dies unweigerlich zu Verbesserungen des Pass- und Teamspiels. Ein weiteres Beispiel ist das Entfernen wesentlicher Orientierungshilfen. So gab es in den frühen Jahren des RoboCups noch farbig codierte Spielfeldmarkierungen und unterschiedliche Torfarben an denen sich die Roboter leicht orientieren konnten. Mit der Simplifizierung der Spielfeldumgebung wurden zunehmend Arbeiten im Bereich der Selbstlokalisierung und der visuellen Wahrnehmung fokussiert.

Als gegenwärtige Spielfeldgrundlage wird ein 9 x 6 m großer grüner Teppich verwendet (siehe Abbildung 2.6 in Zusammenhang mit Tabelle 2.1). Im Regelwerk nicht genau spezifiziert sind Farbton und Textur des Teppichs. Durch regionale Unterschiede ergeben sich somit Anpassungen der Parameter für visuelle Wahrnehmungs- und Laufalgorithmen. Als Spielball dient ein rotfarbener Streethockeyball mit einem Durchmesser von circa 65 mm [Rob13].

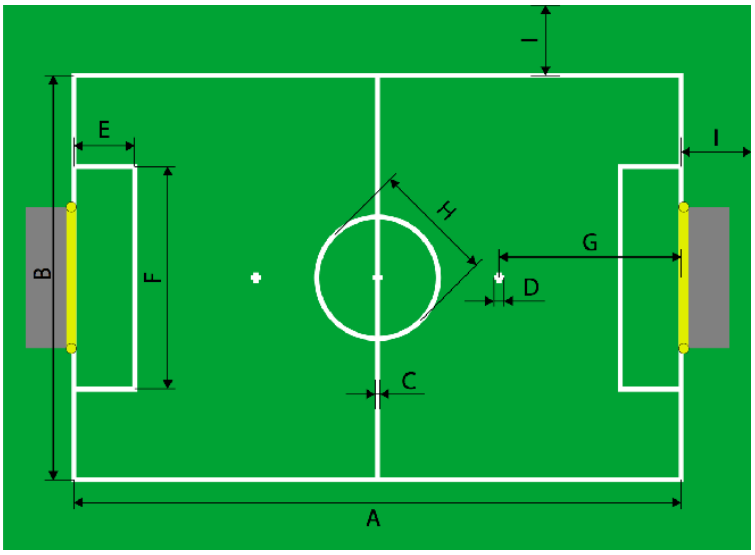


Abb. 2.6: Graphische Darstellung des Spielfelds aus der Standard Plattform Liga [Rob13]

ID	Beschreibung	Länge (in mm)
A	Feld Länge	9000
B	Feld Breite	6000
C	Linien Breite	50
D	Elfmeterpunkt Größe	100
E	Strafraum Länge	600
F	Strafraum Breite	2200
G	Elfmeterpunkt Entfernung	1800
H	Mittelkreis Durchmesser	1500
I	Grenzstreifen Breite	700

Tab. 2.1: Abmessungen des Spielfelds der Standard Plattform Liga

Der Nao-Roboter ist seit dem Jahr 2008 in der SPL in Verwendung und wird von dem französischen Hersteller Aldebaran Robotics gefertigt. Eine Mannschaft besteht aus jeweils fünf Naos. Es sind verschiedene Ausführungen des Roboters erhältlich, die sich in der Anzahl der zur Verfügung stehenden Freiheitsgrade und der Sensorik unterscheiden. Die gegenwertig von dem Nao-Team HTWK Leipzig verwendete „RoboCup Edition“ ist 58 cm groß, wiegt 4,3 kg, verfügt über 21 Freiheitsgrade und stellt für anfallende Kalkulationen einen Intel Atom @ 1.6 GHz Z530 Prozessor zur Verfügung. Als Betriebssystem wird eine Linux-Distribution verwendet, dessen Kernel auf Echtzeitberechnungen optimiert wurde. Die Sensorik reicht von zwei Kameras mit einer Auflösung von 1280 x 960 Bildpunkten, über Ultraschallsensoren in der Brust, vier Mikrophonen die zur Richtungslokalisierung geeignet sind, Gyro- und Winkelsensoren zur Bestimmung der Körperlage, bis hin zu Tastsensoren in den Füßen und einem Berührungssensor auf dem Kopf. Eine Kommunikation mit der Außenwelt ist über die eingebaute Ethernet-,

WiFi- oder Infrarot-Verbindung möglich. Befehle des Schiedsrichter-Computers werden generell per WiFi übermittelt. Der Hersteller gibt eine Laufzeit von 60 Minuten unter Last und 90 Minuten unter Normalbedingungen an.

Abb. 2.7: Der Nao-Roboter aus der Standard Plattform Liga, hier zu sehen mit blauen Sichtkomponenten [PKRhr]

2.2 Pfadplanung

Für die autonome Handlungsfähigkeit eines Roboters ist es notwendig, dass dieser selbstständig Handlungsaufträge planen und ausführen kann. Soll die Aufgabe eines mobilen Roboters darin bestehen, sich von einem Start- zu einem Zielpunkt zu bewegen, ist dafür eine räumliche Navigation unablässig. Voraussetzung für die erfolgreiche Generierung einer kollisionsfreien Bewegungssequenz, ist die Verfügbarkeit eines Abbildes der Umwelt[6], welches alle relevanten Merkmale erfasst. Die Erstellung solch eines Weltmodells umfasst dabei zwei wichtige Teilgebiete der Navigation autonomer mobiler Systeme: die Positionsfindung[7] und die Kartenerstellung[8]. Für beide Vorgänge ist die Wahrnehmung der Umgebung durch Sensoren wie Farbkameras[9], Drucksensoren und Ultraschall enorm wichtig. Erst diese erlauben es dem System in Echtzeit auf Geschehnisse reagieren zu können. Andernfalls wären diese nur wenig flexibel und nur in abgeschirmten Bereichen einsetzbar. Im Anschluss daran lassen sich auf Basis der Sensordaten durch Anwendung entsprechender Filter Rückschlüsse auf die eigene Position ziehen. Das Weltmodell wird im letzten Schritt durch das Erkunden der Umgebung graduell verfeinert und vervoll-

[6] Fortwährend als Weltmodell bezeichnet.
[7] Auch als Lokalisierung bezeichnet.
[8] Auch als Exploration oder Mapping bezeichnet.
[9] Zum aktuellen Zeitpunkt wird im NAO-Team HTWK Leipzig die gesamte Selbstlokalisierung und Exploration durch Video-Sensoren bewerkstelligt.

ständigt[10]. Einen guten Einblick in die Verwendung verschiedener Lokalisierungsfilter und Mapping-Verfahren bietet hierbei der Autor Sebastian Thrun mit seinem Buch *Probabilistic Robotics* [TBF05].

Eine planbasierte Kontrolle muss darauf achten, dass Informationen zum Teil auch unvollständig oder fehlerbehaftet sein können und die Umwelt einem permanenten Änderungsprozess unterliegt. Ein erstellter Plan kann somit schnell an Gültigkeit verlieren. Um diesem Aspekt in die Gesamtproblematik einzubeziehen, wird die Umgebung für einen kurzen Zeitraum als statisch betrachtet. Wird nun ein Plan auf Grundlage dieser Daten erstellt, so ist dieser nur in diesem Zeitabschnitt valide. In jedem darauf folgenden Zeitabschnitt werden neue Pläne anhand aktuellerer Umweltdaten erstellt und somit auf eventuelle Geschehnisse reagiert (siehe Abbildung 2.8).

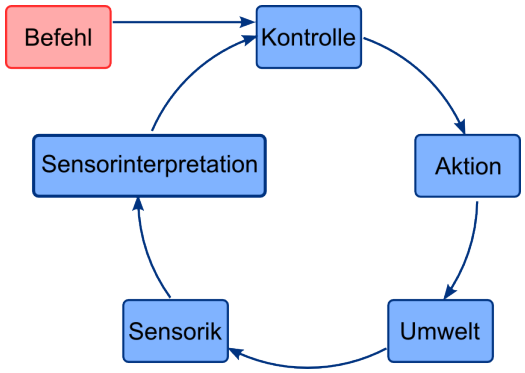


Abb. 2.8: Schematische Darstellung der Komponenten einer planbasierten Kontrolle, in Anlehnung an: [Her12]

Denkbar ist an dieser Stelle auch ein alternativer Ansatz, der einen bereits aufgestellten Plan nicht bei kleinsten Änderungen verwirft und neu entwickelt, sondern lediglich Anpassungen vornimmt. Dieses Konzept wird in [Her12] genauer erläutert. Die planbasierte Kontrolle wird hierbei in diverse Teilmodule gegliedert (siehe Abbildung 2.9). Die Summe der einzelnen Module bilden eine Modulkette. Jedes Modul ergibt sich durch seine Abstraktionsebene und dem Zeitintervall, in dem es arbeitet. Die Granularität der Teilmodule in Abbildung 2.9 wird von oben nach unten immer feiner. Benachbarte Module können sich gegenseitig beeinflussen. Pläne gelten langfristig und werden nur selten aktualisiert, wohingegen Reaktionen und Reflexe kurzfristig auf Ereignisse folgen sollen und somit in einer höheren Frequenz ausgeführt und erneuert werden. Je gröber die Zeitauflösung ist, je höher ist der Abstraktionsgrad. Die Schwierigkeit besteht hierbei darin, durch Anpassungen auf den unteren Ebenen den Gesamtplan nicht aus den Augen zu verlieren.

[10] Da beim Fußballspielen die Umgebung schon im Vorfeld bekannt ist, verzichtet das implementierte Mapping-Verfahren des Nao-Team HTWK Leipzig auf eine vollständige Erfassung aller Umgebungsparameter. Das Mapping besteht lediglich aus der Lokalisierung und dem Tracking von gegnerischen Spielern und des Spielballs.

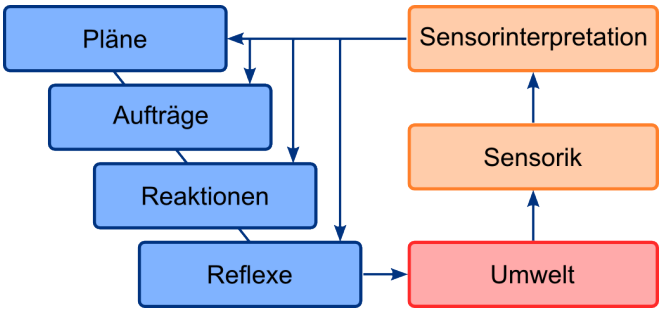


Abb. 2.9: Schema einer Roboterkontrolle mittels vierteiliger Modulkette, in Anlehnung an: [Her12]

2.2.1 Grundlagen und Terminologie

In der Fachwelt ist das Konzept des Arbeits- und Konfigurationsraums[11] weit verbreitet. So stellt der Arbeitsraum ein Abbild des physischen Raums dar, den der Roboter umgibt. Geometrische Daten werden dabei in ein kartesisches Koordinatensystem übertragen. Die Gesamtheit aller im Arbeitsraum möglichen Posen des Roboters wird als Konfigurationsraum bezeichnet. Eine Konfiguration[12] entspricht somit einer spezifischen Position und Ausrichtung eines Roboters. Die Gesamtheit aller im Konfigurationsraum zulässigen Posen wird als freier Konfigurationsraum bezeichnet. Unzulässige Konfigurationen des Roboters werden hingegen als Hindernisraum bezeichnet. Der Konfigurationsraum ist maßgeblich von den Abmaßen und Bewegungsmöglichkeiten des Roboters abhängig. Aus diesem Grund wird der Roboter zumeist als holonomer[13] orientierungsloser Punkt modelliert. Dadurch beschränkt sich die Pfadplanung auf die Planung eines Punktes im Konfigurationsraum, unabhängig von der Kinematik des genutzten Roboters. Bei dieser Darstellungsweise kann es allerdings dazu kommen, dass ein Areal im Konfigurationsraum als zulässig erscheint, dieses im Arbeitsraum aber nicht zugänglich ist. Beispielhaft dafür ist das Erreichen eines Areals durch eine schmale Zufahrt. Im Konfigurationsraum scheint dies noch möglich, da der Roboter hier durch einen holonomen Punkt repräsentiert wird. Im Arbeitsraum können die Abmaße des Roboters zum Passieren der Durchfahrt allerdings schon zu groß sein. Um diese Areale als unzugänglich zu modellieren, werden die Hindernisse um die Hälfte des Roboterradius vergrößert.

Ein Großteil der existierenden Pfadplanungsverfahren führen die Kalkulation des Pfades auf ein Suchproblem zwischen einem Start- und Zielknoten in einem Graphen zurück,

[11] Das Konzept des Konfigurationsraums (Engl. Configuration Space) wurde von Shriram M. Udupa in seiner Dissertationsarbeit [Udu77] vorgeschlagen.

[12] Beschrieben durch x-, y- und z-Position sowie Roll-, Nick-, Gierwinkel. Da sich der Roboter auf einer Ebene bewegt und Roll- und Nickwinkel zumeist nicht relevant sind, kann die Pose auch vereinfacht durch x-, y-Position und Gierwinkel bzw. Orientierung beschrieben werden.

[13] Holonome Systeme sind in der Lage von einer beliebigen Konfiguration in jede andere Konfiguration zu gelangen. So ist ein holonomer Roboter dazu fähig, in jede beliebige Richtung zu fahren ohne vorher eine Drehung durchführen zu müssen.

da sich dieses Problem leicht durch vielseitig erprobte Verfahren wie A* oder Dijkstra lösen lässt. Beispielhaft dafür sind Wegkarten- und Zellzerlegungsverfahren. Einen alternativen Ansatz bietet das Potentialfeldverfahren, welches das gesamte Gebiet mit einem Gradienten überzieht, aus dem jeweils lokal die Fahrtrichtung ersichtlich wird. Eine Übersicht der gängigen Verfahren ist in Abbildung 2.10 zu finden.

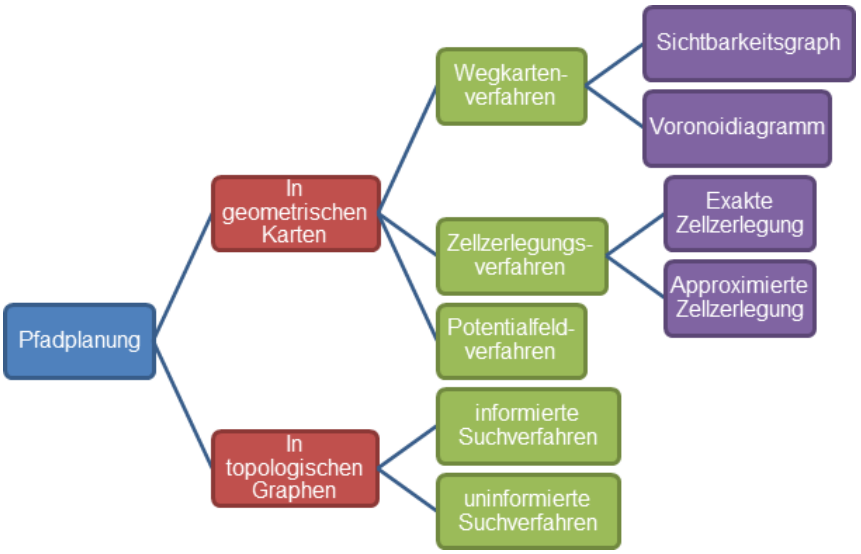


Abb. 2.10: Übersicht gängiger Pfadplanungsverfahren

2.2.2 Bewertungskriterien

Die verschiedenen Verfahren zur Pfadplanung zeichnen sich durch ihre Optimierungsfähigkeit in einzelnen Charakteristiken aus. Bei der Implementierung einer Mehrzieloptimierung kann es durchaus dazu kommen, dass sich die einzelnen Ziele widersprechen und zwischen diesen abgewogen werden muss.

Vollständigkeit

Ein Pfadplaner wird als vollständig bezeichnet, wenn dieser immer einen Pfad von einem Start- zu einem Zielpunkt findet, insofern dieser existiert. Andernfalls gilt der Pfadplaner als unvollständig. Im Zusammenhang mit einem Verfahren, welches auf der Diskretisierung des Raums beruht, wird auch von der Vollständigkeit bezüglich der gewählten Diskretisierung gesprochen. Von einer wahrscheinlichen Vollständigkeit, innerhalb eines bestimmten Zeitintervalls, spricht man bei der Verwendung von Algorithmen, die den Pfad durch das Zufallsprinzip explorieren. Diese Verfahren zeichnen sich durch eine gute Effizienz in höherdimensionalen Konfigurationsräumen aus.

Robustheit

Ist die Fähigkeit eines Pfadplaners auch mit ungenauen oder verrauschten Messdaten akzeptable Ergebnisse generieren zu können. Eine entsprechende Sensordatenfilterung findet bereits im Vorfeld statt und wird somit nicht Bestandteil dieser Arbeit sein.

Sicherheit

Gibt an, wie hoch die Wahrscheinlichkeit einer Kollision mit einem Hindernis ist. So kann das Ziel einer Implementierung sein, einen möglichst großen Abstand zu allen Hindernissen herzustellen. Die Optimierung dieses Faktors korreliert negativ zur Fahrzeit.

Gesamtrotation

Generell verlangsamt eine Drehung die Bewegungsgeschwindigkeit des Roboters. Dies ist zum einen darin begründet, dass die Mechanik des Nao-Roboters darauf ausgelegt ist, geradeaus am schnellsten zu laufen und zum anderen sind auf kurvenreichen Strecken oft mehr Korrektur-Manöver notwendig, was ebenfalls die Geschwindigkeit drosselt. Die Gesamtrotation ist somit ein Maß dafür, wie viel Grad der Roboter auf seinem Pfad gedreht werden muss.

Fahrzeit

Gibt die Zeit an, die ein Roboter benötigt um einen definierten Pfad abzulaufen. Dieser Wert korreliert sehr eng mit der Pfadlänge. Ist ein Pfad der schnellste Pfad, bedingt dies aber nicht, dass er auch der kürzeste ist und umgekehrt.

Echtzeitfähigkeit

Da sich die Umgebung und somit auch der Konfigurationsraum des Roboters kontinuierlich und innerhalb von Sekundenbruchteilen ändert, ist eine statische Kalkulation des Pfades im Voraus nicht möglich. Schon kurz nach dem Beginn der Bewegung auf dem Pfad, könnte ein anderer Roboter bereits den Weg versperren oder ein zuvor versperrter Weg könnte nun begehbar geworden sein. Zuverlässige Vorhersagen über die Trajektorien anderer Roboter lassen sich zudem nur für einen sehr kurzen Zeitraum treffen. Der Roboter muss also in der Lage sein, in kürzester Zeit eine Anpassung des Pfades vornehmen zu können. All diese Kalkulationen müssen auf den begrenzten Rechenkapazitäten des Roboters parallel zu den anderen Prozessen stattfinden können.

Glattheit

Der inhärente Vorteil eines glatten Pfades ist, dass dieser keine Ecken aufweist und auch sonst von einer möglichst geringen Anzahl an Krümmungen geprägt ist. Dadurch wird die Anzahl an Korrekturmanöver reduziert. Wird die Geschwindigkeit des Roboters in engen Kurven nicht gemindert, kann die Trägheit des Roboters zu einem Umsturz führen. Weiterhin kann es möglich sein, dass die Mechanik des Roboters eine abrupte Richtungsänderung gar nicht zulässt. Dem kann entgegen gewirkt werden, indem alle Kurven einen minimalen Radius aufweisen.

2.2.3 Ansätze zur Pfadplanung

In der Literatur ist eine Großzahl von Lösungsansätzen für das Problem der Pfadplanung zu verzeichnen. Diese Methoden lassen sich dabei in Wegkartenverfahren, Potentialfeldverfahren oder Zellaufteilungsverfahren klassifizieren [Lat91]. Im Folgenden sollen die Charakteristika der einzelnen Ansätze skizziert werden.

Wegkartenverfahren

Konzeptionell einigt alle Wegkartenverfahren der Kerngedanke, dass die Vielzahl an Bewegungsmöglichkeiten im freien Konfigurationsraum durch das Heraussuchen einer begrenzten Anzahl von Wegstrecken eingeschränkt wird. Es findet somit eine Komplexitätsreduktion statt. Die Sammlung an Wegstrecken kann als Graph begriffen werden. Dabei bilden Start- und Endpunkte einer Wegstrecke die Knoten und die Strecke als solche die Kanten des Graphen. Anhand ausgewählter Kriterien, wie zum Beispiel der Streckenlänge von der Start- zur Zielkonfiguration, kann mit Hilfe eines Standard-Suchalgorithmus eine entsprechende Sequenz von Kanten gefunden werden. Die Qualität des letztlich gefundenen Weges ist dabei in der Regel nicht von dem Suchalgorithmus abhängig, sondern nur von dem Verfahren, dass den Graphen generiert. Dies ist damit zu begründen, dass Suchalgorithmen, unter Verwendung gleicher Kriterien, gleiche Ergebnisse erzielen und sie sich lediglich in ihrem Effizienzverhalten unterscheiden. Wohingegen bei der Graphen-Erstellung [14] völlig differente Lösungen zustande kommen können. Es existieren eine ganze Reihe unterschiedlicher Verfahren zur Graphen-Generierung, die wichtigsten vier sollen hier erläutert werden.

[14] Die Graphen-Erstellung wird auch als Konstruktionsphase bezeichnet.

Voronoi-Diagramm Für das Entwerfen eines Voronoi-Diagramms[15] ist die Partitionierung des Raums in Regionen notwendig (siehe Abbildung 2.11).

Jeder Punkt im freien Konfigurationsraum hat zu allen Punkten des Hindernisraums einen eindeutigen euklidischen Abstand. Die Zuordnung eines Punktes aus dem Freiraum zu einer Region kann ermittelt werden, indem für jeden Punkt im freien Raum derjenige Punkt im Hindernisraum ermittelt wird, dessen Abstand minimal ist. Dieser Punkt ist auf der Oberfläche des Hindernisses zu finden. Diejenigen Punkte, die zu mehr als einem Punkt im Hindernisraum die minimale Entfernung besitzen, bilden die Grenzen der Voronoi-Regionen.

Um das Voronoi-Diagramm nun auf ein Graphensuchproblem zurückzuführen, werden Kreuzungspunkte und Pfadenden des Voronoi-Diagramms als Knoten repräsentiert, Kanten ergeben sich hingegen aus den Grenzen der Voronoi-Regionen[16]. Somit kann gewährleistet werden, dass ein Roboter, der auf den Grenzen der Voronoi-Regionen entlang fährt, automatisch den weitest möglichen Abstand zu Hindernissen einhält. Wenn eine Umgebung sehr weitläufige Freiräume bietet, kann sich dies jedoch nachteilig auswirken, da unangemessen große Umwege gefahren werden ohne einen Zugewinn an Sicherheit zu erzielen. Um dem vorzubeugen, kann ein maximaler Abstand zu Hindernissen festgelegt werden[17]. Das Verfahren kann als vollständig klassifiziert werden.

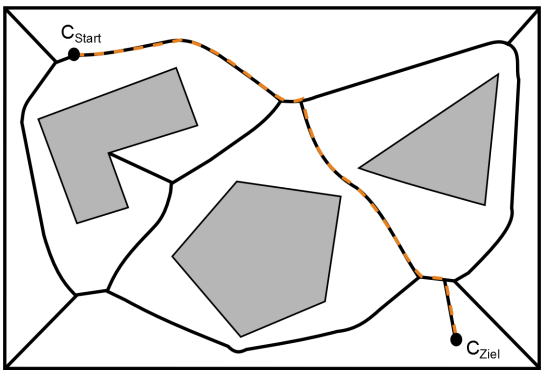


Abb. 2.11: Voronoi-Diagramm in Anlehnung an: [Die12]

Probabilistische Straßenkarte Die Idee der Probabilistischen Straßenkarte ist die Erstellung eines Graphen anhand eines samplingbasierten Zufallsprinzips (siehe Abbildung 2.12).

Zunächst werden zufällige gleichverteilte Konfigurationen in den Freiräumen der Umgebungskarte erstellt. Anschließend werden jeweils die nächsten Nachbarn miteinander ver-

[15] Voronoi-Diagramme werden in der Literatur auch als Thiessen-Polygone oder Dirichlet-Zerlegung bezeichnet.
[16] Parabolisch Grenzen des Voronoi-Diagramms können durch aneinandergereihte Knoten approximiert werden.
[17] Diese Strategie wird auch als Küstennavigation bezeichnet.

knüpft, sodass ein kollisionsfreier Graph entsteht der mittels Graphen-Suchalgorithmus nach dem kürzesten Pfad durchsucht werden kann. Die bei der Verknüpfung erstellten Kanten dürfen vorhandene Hindernisse nicht schneiden[18].

Mit steigender Anzahl an Sampling-Punkten nähert sich die Lösung asymptotisch dem optimalen Pfad an[19]. Durch die zufällige Verteilung der Punkte im Freiraum kommt es meist zu einer nicht gewünschten Bildung von „Ecken“ auf dem Pfad. Wird die Anzahl der Sampling-Punkte verringert, steigert dies die Performance des Graphen-Suchalgorithmus, verstärkt aber zugleich die unnötige Eckenbildung des Pfades. Um diesen Effekt zu entgegnen, wird im nächsten Schritt oft ein Verfahren zur Glättung des Pfades nachgeschaltet.

Die Konstruktion einer detaillierten Karte ist sehr zeitaufwändig. Daher sind Probabilistische Straßenkarten im besonderen Maße für statische Umgebungen – wie zum Beispiel Industrieszenarien – geeignet, da hier erzeugte Straßenkarten wiederverwendet werden können. Abhängig von der Anzahl der gewählten Sampling-Punkte ist das Verfahren „wahrscheinlich vollständig”.

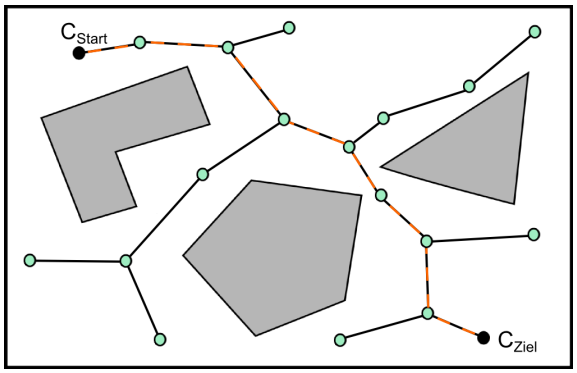


Abb. 2.12: Probabilistische Straßenkarte in Anlehnung an: [Die12]

Sichtbarkeitsgraph Das Verfahren des Sichtbarkeitsgraphen wird für zweidimensionale Konfigurationsräume mit polygonalen Hindernissen verwendet und stellt eines der ersten Verfahren zur Pfadplanung dar (siehe Abbildung 2.13) [Nil69].

Die Start-, Zielkonfiguration und die Eckepunkte der Hindernisse entsprechen den Knoten des Graphen. Kanten werden überall dort hinzugefügt, wo ein direkter Sichtkontakt zwischen den Knoten zustande kommt. Das Verfahren gilt als vollständig und garantiert zudem immer den kürzesten Weg zu finden. Ein Problem ist dabei die Nähe der erzeugten Pfade zu den Hindernissen. Da die Hindernisse der Umgebungskarte um den Roboterradius vergrößert werden, ist der effektive Abstand zwischen dem Roboter

[18] Es ist angebracht für den Abstand zwischen den Nachbarn einen obere Schranke zu definieren.
[19] Der optimale Pfad entspricht dem des Sichtbarkeitsgraphen.

und den Hindernissen gleich null. Um einen angemessenen Sicherheitsabstand zu gewährleisten, muss der Pfad somit modifiziert werden. Das Ergebnis ist nicht selten eine Trajektorie des Roboters, die zum Teil auf der Ideallinie entlang führt, zum Teil aber auch eigenartig anmutende Ausweichmanöver in der Nähe von Hindernissen bewirkt.

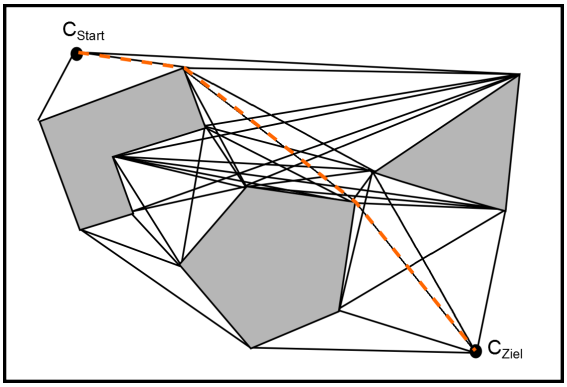


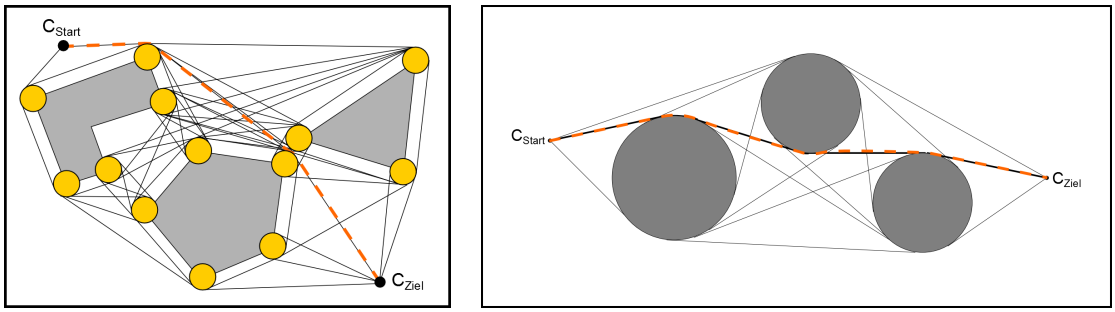
Abb. 2.13: Sichtbarkeitsgraph in Anlehnung an: [Die12]

Erweiterter Sichtbarkeitsgraph Dieses Verfahren stellt eine modifizierte Version des Sichtbarkeitsgraphen dar. Die Basis bilden auch hier geometrische Berechnungen zur Bildung eines Graphens, welcher im Anschluss mittels eines ausgewählten Suchverfahrens nach dem kürzesten Pfad untersucht werden kann. Ergänzt wird das Verfahren lediglich dadurch, dass im Falle der Verwendung einer polygonalen Hindernisrepräsentation die Polygonpunkte von Kreisen überlagert bzw. die Hindernisse ganzheitlich durch Kreise approximiert werden können (siehe Abbildung 2.14a und Abbildung 2.14b im Vergleich). Durch den Einsatz eines minimalen Kreisradius kann zudem die maximale Krümmung des Pfades bzw. die Rotationsgeschwindigkeit des Roboters beeinflusst werden. Dadurch ist das Verfahren im besonderen Maße für nicht holonome Systeme geeignet. Weiterhin garantiert die Überlagerung aller konvexen Polygonpunkte durch Kreise einen Minimalabstand zu den Hindernissen bei einem gleichzeitigen sehr harmonischen Pfadverlauf.

Die Anwendung impliziert eine Präzisions- und Laufzeitunabhängigkeit von der Spielfeldgröße[20]. Die Laufzeit ist vielmehr von der Anzahl der Hindernisse abhängig (siehe Abschnitt 3.4). Die Kehrseite des Verfahrens ist in der erschwerten Verhaltenssteuerung des Roboters durch die Gewichtung von Spielfeldregionen zu finden. Durch den vergrößerten Minimalabstand zu den vorhandenen Hindernissen kann es zudem dazu kommen, dass ein eigentlich begehbarer Weg dem System als unzugänglich erscheint. Somit ist das Verfahren als nicht vollständig zu klassifizieren. Da der Radius der Hindernisse frei

[20] Im Vergleich: Wird unter der Verwendung der Potentialfeldmethode das Spielfeld vergrößert, führt dies ohne Anpassung der Parameter zu einem Präzisionsverlust. Dieser Entwicklung kann entgegengesteuert werden, indem die Dichte des Vektorfelds erhöht wird, wodurch zusätzliche Rechenoperationen anfallen.

gewählt werden kann und der additive Abstand die Sicherheit vor Kollisionen erhöht, muss dieser vermeintliche Nachteil eher als Kompromiss zwischen Vollständigkeit und Sicherheit angesehen werden.



(a) Die Ecken der polygonalen Hindernisse werden von Kreisen überlagert, in Anlehnung an: [Die12]

(b) Die Hindernisse werden durch Kreise repräsentiert, in Anlehnung an: [Top99]

Abb. 2.14: Erweiterter Sichtbarkeitsgraph

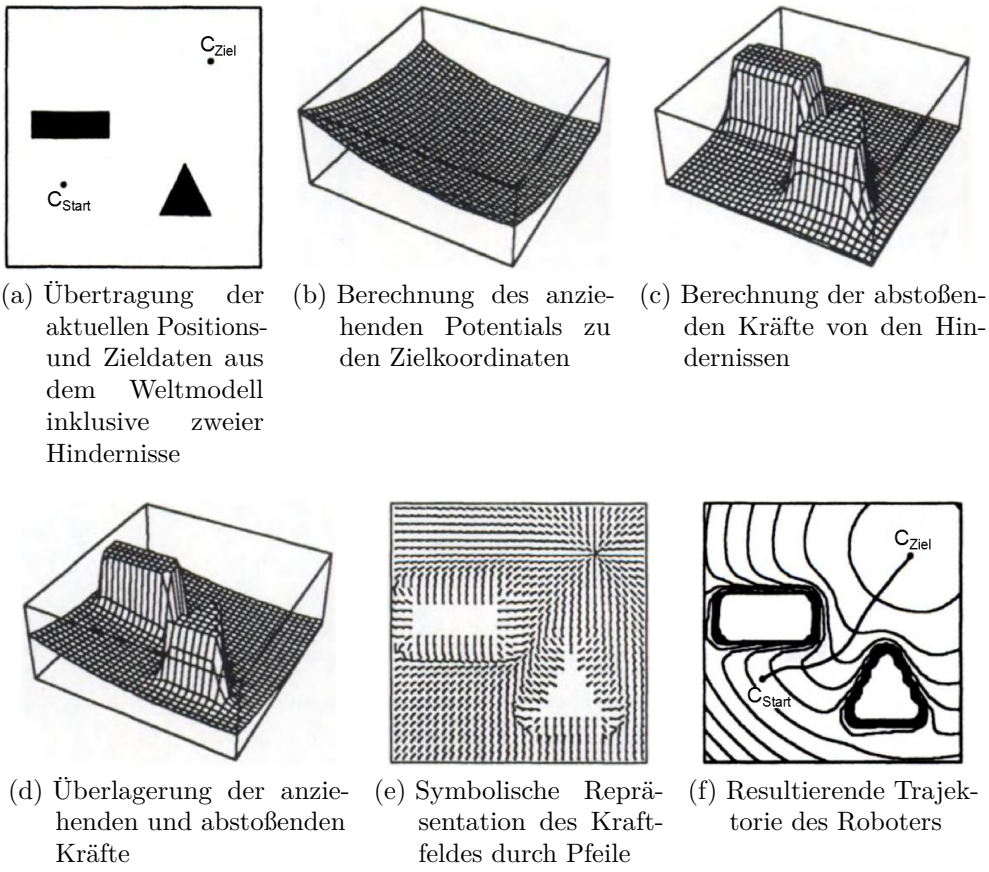
Potentialfeldverfahren

Das Potentialfeldverfahren[21] ist als Greedy-Algorithmus [22] zu klassifizieren. Bei dieser Methode wird der Roboter als Punktmasse in einem kartesischen Koordinatensystem gesehen, der unter dem Einfluss eines künstlichen Kraftfeldes steht (siehe Abbildung 2.15). Der zugrunde liegende Gedanke ist, dass die Zielkoordinate eine anziehende Kraft auf die Punktmasse des Roboters hat, während Hindernisse mit abstoßenden Kräften auf diese Punktmasse einwirken. Die Zielkonfiguration ist dabei eine Potentialsenke, die zugleich das globale Minimum darstellt. Wird die aktuelle Lage des Roboters in das kartesische Koordinatensystem des Potentialfeldes projiziert, kann die Bewegungsrichtung durch die Berechnung des negativen Gradienten bestimmt werden. Der negative Gradient gibt dabei die Bewegungsrichtung mit dem steilsten Gefälle an. Dieses Verfahren zeichnet sich durch sein einfaches und intuitives Verständnis aus. Es ist leicht zu implementieren und unter der Einschränkung, dass eine statische Umgebung vorherrscht, kann das Potentialfeld auch für andere Startkoordinaten wiederverwendet werden. Unter ungünstigen Voraussetzungen erweist sich das Verfahren bei der Erstellung des Feldes jedoch als sehr rechenintensiv. Weiterhin ist für das Erzielen guter Ergebnisse die Wahl einer geeigneten Potentialverteilung von Nöten. Schon bei wenig komplexen Potentialfeldern kann es zur Bildung lokaler Minima kommen. Dies sorgt dafür, dass der Roboter ohne Anwendung heuristischer Reparations-Algorithmen in diesen lokalen Senken stehen bleibt. Ohne die

[21] Das Potentialfeld wird auch als Gradientenfeld oder Vektorfeld bezeichnet.

[22] Greedy-Algorithmen werden auch als gierige Algorithmen bezeichnet und stellen in der Informatik eine spezielle Klasse von Algorithmen dar. Unter zur Hilfenahme einer Bewertungsfunktion wählen Greedy Algorithmen sukzessive den Folgezustand aus, der am vielversprechendsten erscheint.

Beseitigung der lokalen Minima gilt der Algorithmus als nicht vollständig. Weiterhin kann es beim Passieren enger Abschnitte zu einer zunehmend oszillierenden Bewegung des Roboters kommen [Kor91]. Das Verfahren findet oft einen kurzen Weg, aber es ist nicht immer der kürzeste.



(a) Übertragung der aktuellen Positions- und Zieldaten aus dem Weltmodell inklusive zweier Hindernisse

(b) Berechnung des anziehenden Potentials zu den Zielkoordinaten

(c) Berechnung der abstoßenden Kräfte von den Hindernissen

(d) Überlagerung der anziehenden und abstoßenden Kräfte

(e) Symbolische Repräsentation des Kraftfeldes durch Pfeile

(f) Resultierende Trajektorie des Roboters

Abb. 2.15: Potentialfeldverfahren in Anlehnung an: [Lat91]

Zellaufteilungsverfahren

Die zugrunde liegende Idee dieser Verfahren ist die Zerlegung des freien Konfigurationsraums in Teilregionen. Diese disjunkt konvexen Teilregionen werden als Zellen bezeichnet und repräsentieren in ihrer Summe einen Graphen. Jede Zelle ist dabei als Knoten des Graphen zu verstehen, wobei unmittelbar benachbarte Zellen über Kanten verbunden sind. Nach der Generierung des Graphen kann dann mit Hilfe eines geeigneten Suchalgorithmus, wie zum Beispiel A*, nach dem kürzesten Weg innerhalb des Graphen gesucht werden. Die Sequenz von miteinander verbunden Zellen, die von der Start- zur Zielzelle führt, wird als Kanal bezeichnet. Es findet eine Klassifizierung in exakte und approximierte Verfahren statt.

Exakte Zellzerlegung Charakteristisch zeichnen sich exakte Verfahren dadurch aus, dass sie eine genaue Dekomposition des freien Konfigurationsraums erstellen (siehe Abbildung 2.16). Die Menge an Zellen ist dabei endlich und sie überlappen einander nicht. Die Gesamtheit aller Zellen kommt somit exakt dem freien Konfigurationsraum gleich. Exakte Zellaufteilungsverfahren sind vollständig und finden somit immer einen Pfad, insofern dieser existiert. Ein Nachteil hingegen ist, dass die Pfade oftmals sehr nahe an Hindernissen vorbei führen. Exakte Verfahren sind den approximierten vorzuziehen, wenn im Anwendungsfall mit wenigen polygonalen Hindernissen gerechnet wird. Vorteilhaft an diesen Verfahren ist die Unabhängigkeit von der Größe des Arbeitsraums.

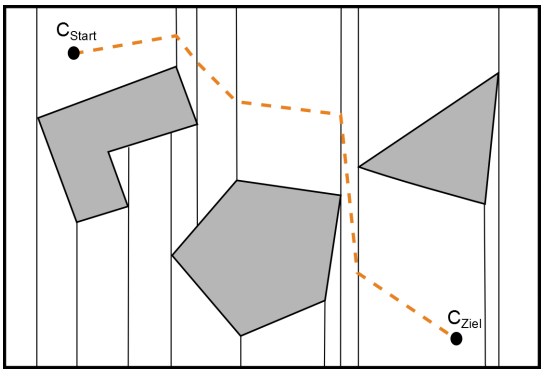


Abb. 2.16: Exakte Zellzerlegung in Anlehnung an: [Die12]

Approximierte Zellzerlegung Eine genaue Zerlegung des freien Konfigurationsraums ist bei approximierten Methoden nur selten der Fall, da sich der zugrunde liegenden Raumstruktur nur mittels einfacher Formen, wie zum Beispiel Rechtecken, angenähert wird (siehe Abbildung 2.17). Die Zellen sind auch hier so angeordnet, dass sie sich nicht überlappen. Die vereinigte Menge aller Zellen stellt lediglich eine Teilmenge des freien Konfigurationsraums dar. Approximierte Zerlegungsverfahren sind nicht vollständig. Demzufolge kann es in einem ungünstigen Fall dazu kommen, dass eine zu grobe Rasterung dazu führt, dass kein Weg gefunden werden kann. Dem kann aber durch die Verfeinerung der Auflösung des Rasters entgegengewirkt werden. Die Anzahl der Kanäle wird hierbei durch die Strukturgröße des zugrunde liegenden Musters bestimmt. Diverse Verfahren nutzen diese Eigenschaft, um mit einer groben Rasterung zu beginnen und dann, bei erfolgloser Suche, stufenweise zu verfeinern. Dies geschieht so lange, bis ein Pfad gefunden wird oder eine definierte Anzahl an Verfeinerungszyklen durchlaufen ist. Da die Anzahl der benötigten Zellen proportional abhängig ist von der Größe des Arbeitsraums, sind approximierte Verfahren generell bei kleineren Arbeitsräumen vorzuziehen.

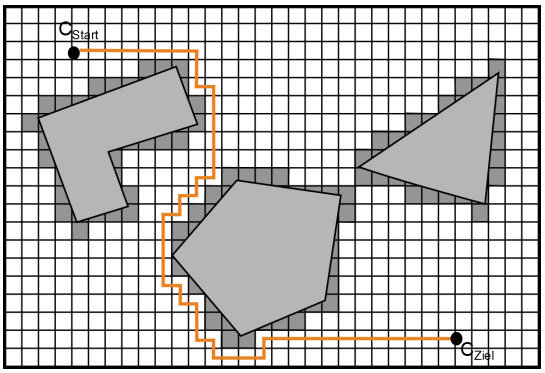


Abb. 2.17: Approximierte Zellzerlegung in Anlehnung an: [Die12]

2.2.4 Suchverfahren

Unter einem Suchverfahren wird das systematische Durchqueren eines Graphen verstanden. Ziel ist es, unter den gegebenen Bedingungen eine Route zu wählen, die von dem gegebenen Startzustand (Startknoten) des Agenten in einen gewünschten Endzustand (Zielknoten) führt[23].

Ein Graph wird durch eine Synthese von Knoten und Kanten repräsentiert und entspricht in seiner Zusammensetzung einem Zustandsraum, in dem jeder Knoten einen spezifischen Konfigurationszustand des Agenten darstellt. Der Konfigurationszustand eines Knoten kann über die Koordinaten hinaus auch Daten über die anzunehmende Ausrichtung und Geschwindigkeit des Agenten beinhalten. Das Auffinden bzw. Erreichen von Knoten ist ausschließlich über die sie verbindenden Kanten möglich. Sie stellen somit eine Überführung von einem Zustand in einen anderen dar. Es gilt dabei zu beachten, dass die Kanten durch Gewichte parametriert werden. Die Justierung der Kantengewichte erlaubt eine feinere Abstimmung des Verhaltens und ist somit für die Qualität der gefundenen Route von entscheidendem Einfluss. Die Gewichtung erfolgt im einfachsten Falle durch die Länge der zu laufenden Wege. Darüber hinaus ist es zum Beispiel auch möglich, Routen im Sichtbereich gegnerischer Spieler zu benachteiligen oder Wege zu begünstigen, die besonders weit entfernt an Hindernisobjekten vorbei führen.

Eine Übereinstimmung aller Suchverfahren lässt sich in der Expansion von Knoten finden. Unter dem Expandieren eines Knoten versteht man einen Zyklus aus der Bestimmung aller erreichbaren Nachbarknoten eines Zielknotens und der Selektion eines Nachfolgerknotens, der nach benachbarten Knoten untersucht werden soll. Um dies zu ermöglichen, müssen von einem Suchverfahren zwei Mengen verwaltet werden: die Open- und die Closed-Menge. Alle bekannten Knoten[24] werden in der Open-Menge mit einer Kostenabschätzung hinterlegt. Zu Beginn der Suche befindet sich in der Open-Menge nur

[23] Diese Beschreibung entspricht hinlänglich der Definition eines Planungsproblems.
[24] Unbekannte Knoten sind Knoten, die bis zum gegenwärtigen Zeitpunkt noch nicht untersucht wurden.

der Startknoten, alle anderen Knoten sind unbekannte Knoten. Aus der Open-Menge wird ein zu expandierender Knoten ausgewählt, untersucht, und wandert anschließend von der Open-Menge in die Closed-Menge um nicht mehrfach untersucht zu werden. Es existiert eine beachtliche Vielfalt an Suchverfahren die sich in der Verwaltung der Open- und Closed-Menge sowie der Auswahl des zu examinierenden Knotens differenzieren.

Im Folgenden sollen drei populäre Suchverfahren vorgestellt werden: Dijkstra-Algorithmus, Bestensuche und der A*-Algorithmus. Identisch ist bei allen drei Verfahren, dass sie eine Kostenfunktion $f(x)$ für die Bewertung der bekannten Knoten verwenden. Aus der Open-Menge wird dann derjenige Knoten zur Expansion gewählt, welcher die geringsten Kosten aufweist. Der Unterschied zwischen den Verfahren ist in der verschiedenen Implementierung von $f(x)$ zu finden. Um das unterschiedliche Suchverhalten besser veranschaulichen zu können, wird als Graph ein zweidimensionales Gitter angenommen. Ein Knoten wird dabei durch eine Zelle repräsentiert, die über ungerichtete Kanten zu den horizontal und vertikal benachbarten Zellen verbunden ist.

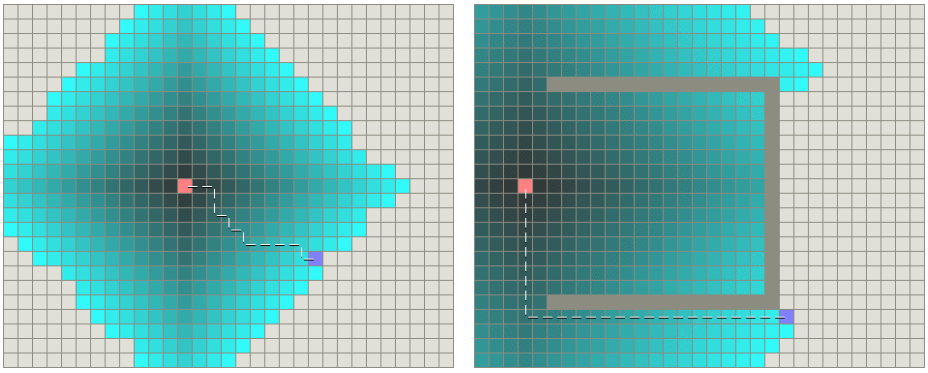
Dijkstra-Algorithmus

Ausgehend vom Startknoten examiniert der Dijkstra-Algorithmus[25] immer den am nächsten liegenden Knoten, welcher sich in der Open-Menge befindet. Bei der Untersuchung der einzelnen Knoten expandiert das Verfahren vom Startknoten scheinbar zirkulär nach außen, bis es den Zielknoten gefunden hat (siehe Abbildung 2.18a). Die Kostenfunktion $f(x)$ ist äquivalent zu den bisherigen Wegkosten $g(x)$ die aufgewendet werden müssen, um von dem Startknoten bis zum ausgewählten Knoten zu gelangen. Es gilt:

$$f(x) = g(x)$$

Dadurch bedingt, dass zuerst erzeugte Knoten auch als erstes untersucht werden, ähnelt das Vorgehen zur Selektion eines Nachfolgeknotens dem FiFo-Prinzip[26]. Die Verwendung des Algorithmus garantiert immer einen kürzesten Weg[27] zu finden, solange der Graph ausschließlich positive Kantengewichte enthält. In Abbildung 2.18b ist zu erkennen, dass der Dijkstra-Algorithmus durch sein ungerichtetes Vorgehen – denn er expandiert in alle Richtungen gleichermaßen – insgesamt eine sehr große Anzahl an Knoten untersuchen muss, bis er den Zielknoten findet. Dieses Verhalten ist darauf zurückzuführen, dass keinerlei Heuristikfunktion Bestandteil des Algorithmus ist, wodurch der Algorithmus als uninformiert zu klassifizieren ist.

[25] Benannt nach seinem Erfinder: Edsger W. Dijkstra – Erstmals veröffentlicht in [Dij59].
[26] Englisch: First In - First Out.
[27] Es können auch mehrere kürzeste Wege existieren.



(a) (b)

Abb. 2.18: Dijkstra-Algorithmus [Pathr]

Die Abbildungen zeigen auf anschauliche Weise das Suchverhalten des Diskstra Algorithmus. Dabei ist der Startknoten durch das pinkfarbenes Quadrat und der Zielknoten durch das blaufarbenes Quadrat gekennzeichnet. Farbig gekennzeichnet sind zudem alle Knoten bzw. Quadrate, die der Algorithmus bei seiner Exploration untersucht. Je heller die Färbung ist, um so weiter ist der Knoten vom Startknoten entfernt und umso größer ist der Wert von $g(x)$. Das graufarbene konkave Gebilde in Abbildung b repräsentiert ein Hindernis.

Bestensuche

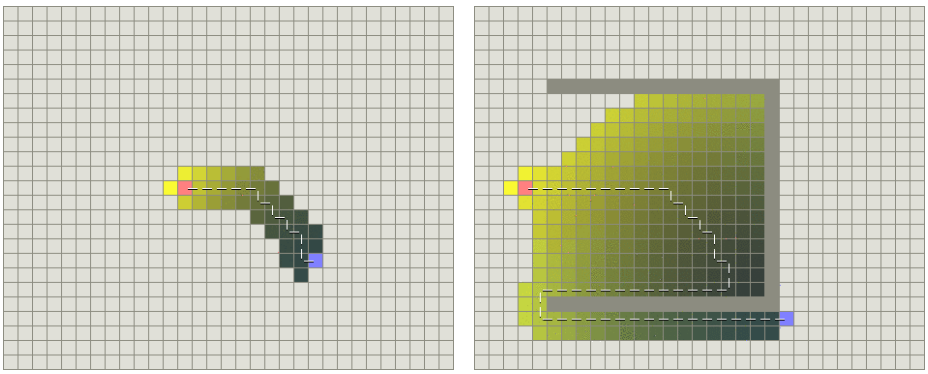
Die Bestensuche[28] arbeitet auf eine vergleichbare Weise wie der Dijkstra-Algorithmus, unterscheidet sich aber in der Bestimmung des zu examinierenden Nachfolgeknotens. Während beim Dijkstra-Algorithmus ein Knoten gewählt wird, welcher sich am nächsten zum Startknoten befindet, wird bei der Bestensuche ein Knoten ermittelt, welcher am nächsten zum Zielknoten liegt. Zum Schätzen des Abstandes vom gewählten Knoten zum Zielknoten wird eine Heuristikfunktion $h(x)$ verwendet. Die Distanz wird dabei im Regelfall über den euklidischen Abstand bestimmt. Die Kostenfunktion sieht wie folgt aus:

$$f(x) = h(x)$$

Die Verwendung der Bestensuche garantiert nicht einen kürzesten Weg zu finden und ist somit nicht optimal. Als Vorteil erweist sich dafür, dass durch die verwendete Heuristik eine zielgerichtete und somit schnellere Suche nach dem Zielknoten erfolgt. Wie in Abbildung 2.19a zu sehen ist, muss im Vergleich zum Dijkstra-Algorithmus somit nur ein geringer Bruchteil der Knoten untersucht werden. Die Bestensuche ist als Greedy-Algorithmus einzuordnen. Das gieren in Richtung Ziel erscheint als logisch und macht die Suche um einiges effektiver, stellt sich in der Verwendung jedoch immer wieder als

[28] Bestensuche (Englisch: Best First Search) wurde erstmals im Jahre 1984 von Judea Pearl in seinem Buch [Pea84] beschrieben.

problematisch heraus. Aufgrund dessen, dass der Algorithmus die Kosten des bisher zurückgelegten Weges ignoriert und nur die Kosten zum Ziel berücksichtigt werden somit auch Knoten bevorzugt, die näher am Ziel liegen, auch wenn sich dieser Weg später als sehr lang erweist (siehe Abbildung 2.19b).



(a) (b)

Abb. 2.19: Suchverhalten der Bestensuche [Pathr]

In diesen Abbildungen ist der Startknoten durch ein pinkfarbenes Quadrat und der Zielknoten durch ein blaufarbenes Quadrat gekennzeichnet. Farbig markiert sind auch hier alle Quadrate bzw. Knoten, die vom Algorithmus untersucht werden. Die graduellen Farbverlaufe sind hierbei konträr zu denen aus Abbildung 2.18 a und b. Sie werden mit geringerem Abstand zum Ziel immer dunkler. Je heller ein Feld ist, um so höher ist der Wert von h(x). Auch hier wird in Abbildung b ein Hindernis durch ein graufarbenes konkaves Gebilde abgebildet.

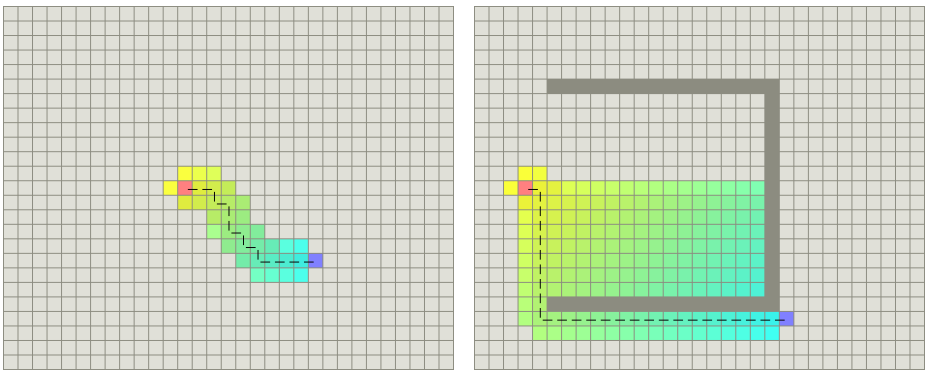
A*-Algorithmus

Der A*-Algorithmus[29] stellt zum gegenwärtigen Zeitpunkt wohl einer der beliebtesten Suchverfahren dar. Dies ist damit zu begründen, dass das Verfahren eine ideale Kombination aus dem zuvor vorgestellten Dijkstra-Algorithmus mit der Bestensuche ist und somit die Stärken beider Verfahren vereint. So ist der A*-Algorithmus in der Lage - wie der Dijkstra-Algorithmus - unter Verwendung positiver Kantengewichte für die Optimalität der Ergebnisse zu garantieren, d.h.: immer den kürzesten Weg zu finden. Wenngleich A* eine Heuristikfunktion nutzt – wie die Bestensuche – um den Abstand zum Ziel zu schätzen, woraus ein reduzierter Speicher- und Zeitaufwand resultiert (siehe Abbildung 2.20a und Abbildung 2.20b). Der Wert von $f(x)$ wird durch die Summe aus den bisherigen Wegkosten $g(x)$ und den geschätzten Kosten bis zum Zielknoten $h(x)$ repräsentiert:

$$f(x) = g(x) + h(x)$$

[29] Gesprochen: A Stern – Wurde erstmals im Jahr 1968 von Peter E. Hart, Nils J. Nilsson und Bertram Raphael in ihrem gemeinsam veröffentlichten Paper [HNR68] beschrieben.

Um die Optimalität des Verfahrens zu gewährleisten, darf h(x) die tatsächlichen Kosten nie übersteigen. Sollte $h(x)$ dennoch größer sein, führt dies dazu, dass weniger Knoten untersucht werden, wodurch das Verfahren schneller arbeitet. In Extremfällen (wenn $h(x)$ in Relation zu $g(x)$ sehr groß ist), degeneriert der A*-Algorithmus zur Bestensuche. Eine geeignete Methode um den Abstand zu bestimmen, wäre auch hier der euklidische Abstand. Im Falle einer komplett uninformierten Suche, wo $h(x) = 0$ gilt, degeneriert A* zum Dijkstra-Algorithmus.



(a) (b)

Abb. 2.20: Suchverhalten des A*-Algorithmus [Pathr]

Die Form der Darstellung in diesen beiden Abbildungen ist gleich den Illustrationen 2.18 und 2.19. Anzumerken ist lediglich, dass die Gewichtung im gelbfarbenen Bereich stärker dem Einfluss der Heuristik $h(x)$ unterliegt und im türkisfarbenen Bereiche, die bisherigen Wegkosten $g(x)$ einen größeren Einfluss haben.

3 Erweiterter Sichtbarkeitsgraph

Im vorherigen Kapitel wurden die unterschiedlichen Pfadplanungskonzepte und Suchverfahren für eine detailierte Betrachtung herangezogen und deren Charakteristika gegenübergestellt. Es ist anzumerken, dass zum gegenwärtigen Zeitpunkt fast alle angeführten Pfadplanungsverfahren von konkurierenden Teams adaptiert werden. Das Eruieren einer optimalen Lösung erscheint als sehr aufwändig, da die praktischen Unterschiede der Navigationsverfahren nach einer entspechenden Optimierungsphase wohl eher subtiler Natur sind. Die Selektion qualifizierter Technologien ist somit eher von persönlichen Präferenzen getrieben und hat folgende Designentscheidungen hervorgebracht:

- Die planbasierte Kontrolle erfolgt, wie unter Abschnitt 2.2 beschrieben, nach einem simplen Grundschema, welches die Umgebung für einen kurzen Zeitraum als statisch betrachtet und eine vollständige Neukalkulation des Pfades zu jedem zur Verfügung stehenden Zeitabschnitt vorsieht. Somit wird auf die Entwicklung einer mehrteiligen Modulkette verzichtet.
- Als Pfadplanungsverfahren wird der erweiterte Sichtbarkeitsgraph Verwendung finden, siehe Unterabschnitt 2.2.3. Dieser erhält den Vorzug, da die durch ihn generierten Graphen äußerst elegant erscheinen und die daraus ableitbaren Pfade zudem auch noch vollkommen glatt sind. Darüber hinaus ist das Verfahren laufzeitunabhängig von der Spielfeldgröße.
- Die Suche nach dem kürzesten Pfad wird mittels A*-Algortihmus durchgeführt, da diese Methode vielseitig erprobt ist und durch ein effektives Explorationsverhalten zu überzeugen vermag. Als Heuristikfunktion wird der euklidische Abstand verwendet.

Die angeführte Konstellation scheint keine überaus prekären Schwächen aufzuweisen und so sollen in diesem Kapitel die zentralen Komponenten der implementierten Software separat betrachtet werden. Die obligatorische Suche nach dem kürzesten Pfad mittels A*-Algorithmus wird dabei nicht dediziert betrachtet, da es sich hierbei um ein sehr gut dokumentiertes Standardverfahren handelt, dessen Implementierungsdetails einer

breiten Masse an Literatur entnommen werden kann[1]. Abbildung 3.1 soll eine kurzen Überblick darüber gewähren, welche Teilschritte für die Umsetzung des erweitertem Sichtbarkeitsgraphen von Nöten sind.

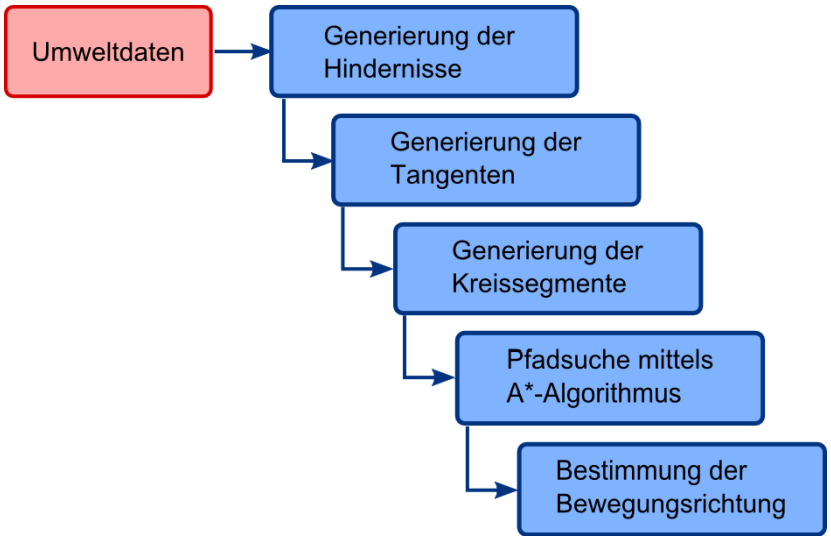


Abb. 3.1: Schematische Darstellung des Programmablaufs

Die Bestimmung der Bewegungsrichtung ist ein essentieller Bestandteil der Applikation, umfasst in seiner Gesamtheit jedoch nur eine sehr geringe Komplexität, wodurch dieses Thema in Abschnitt 3.2 mit behandelt wird.

3.1 Generierung des Graphen

Für die Formeln in diesem Kapitel sollen folgende Definitionen gelten:

Hindernis bzw. Kreis: $O = (M, r)$

Punkt: $C = (x, y)$

Mittelpunkt: $M = (x, y)$

Schnittpunkt: $S = (x, y)$

Radius: r

Tangente: $T = (\overrightarrow{C_i C_j})$

3.1.1 Hindernismodellierung

Dieser Unterabschnitt soll die Umstände beschreiben, welche zur Darstellung des repräsentativen Hindernisraums beigetragen haben. Durch die identische Bauweise der

[1] Eine nennenswerte Quelle stellt [Pathr] dar. Der Autor hat dort unter „Source code and demos" ein Sammelsurium unterschiedlichster Implementierungen zusammengetragen.

Roboter begünstigt, reicht ein einziges Modell für die Hindernisstruktur aus. Da zum aktuellen Zeitpunkt die Drehrichtungen anderer Roboter nicht bestimmt werden können, kann von einer Rotationssymmetrie bezüglich der vertikalen Ausrichtung ausgegangen werden. Dies ermöglicht eine ausreichend präzise Verkörperung der Hindernisse durch primitive geometrische Kreisfiguren[2]. Wie in Abbildung 3.2 zu erkennen ist, kann das Modell dabei aus drei Kreisen bestehen, wobei für die anstehenden geometrischen Berechnungen nur der äußere von Relevanz ist. Der innere Kreis stellt den Roboter in unmittelbar materialisierter Form dar. Die Schulterbreite eines Nao-Roboters beträgt 275 mm, wodurch sich für den inneren Kreis ein Radius von 138 mm ergibt. Würde sich ein Objekt innerhalb dieses Radiuses befinden, käme es zu einer Kollision. Der Radius des mittleren Kreises entspricht dem des inneren Kreises, multipliziert um den Faktor zwei. Bewegt sich ein Roboter auf dieser Kreisbahn, hätte er einen effektiven Abstand von null zum Hindernis, was in der Praxis generell nicht ausreichend ist. Der Radius des äußeren Kreises entspricht dem des mittleren Kreises, ergänzt um weitere 350 mm Sicherheitsabstand. Die Koordinaten der drei Kreismittelpunkte sind vollständig kongruent. Der additive Sicherheitsabstand von 350 mm resultiert aus einer vorläufigen Schätzung. Bei der Wahl dieser Konstante wurden folgende Faktoren bedacht:

- Die Genauigkeit mit welcher andere Roboter lokalisiert werden können. Dieser Wert kann vorläufig noch nicht genau bestimmt werden, da praktische Erfahrungs- und Messwerte zu diesem Zeitpunkt noch ausstehen.
- Die Genauigkeit der eigenen Lokalisierung. Diese korelliert im starken Maße mit der eigenen Spielfeldposition. Die Wahrscheinlichkeitsverteilung ist dabei sehr komplex. Nach praktischer Erfahrung haben die Koordinaten eine maximale Abweichung zur realen Spielfeldposition von 300 mm.
- Ungenauigkeiten in der Robotersteuerung.
- Die unvorhersehbare Trajektorie gegnerischer Roboter.

Sowohl Start- als auch Zielpunkt werden für die Erstellung des Graphens als Hindernisse mit einem Radius von null mm definiert. Dies ist notwendig, da nur so die Tangentenberechnung zu bzw. von den beiden Punkten erfolgen kann.

[2]Dieser Sachverhalt erscheint darüber hinaus als wenig kritisch, da die Abmaße der Fußlänge zur Schulterbreite mit einem Verhältnis von 1 zu 1,6 als durchaus homogen bezeichnet werden können.

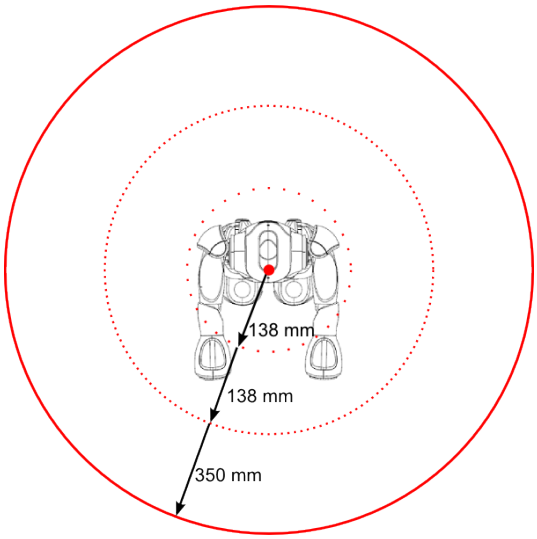


Abb. 3.2: Hindernismodellierung

Eine weitere Möglichkeit der Darstellung wäre die in [Our04] erläuterte Approximation der Kreise durch Polygone. In der genannten Arbeit wird auch ausgeführt, dass viele Berechnungen auf einem polygonbasierten Modell erheblich langsamer sind, wodurch eine weitere Betrachtung nicht Gegenstand dieser Arbeit sein wird.

3.1.2 Berechnung der Tangenten

Die Berechnung der Tangenten stellt einen essentiellen Schritt zur Bildung des Graphen dar. Eine Tangente ist ein Pfadelement, welches im Graphen durch eine Kante repräsentiert wird. Jede berechnete Tangente wird darauf geprüft, ob sie von Hindernissen überlagert wird. Nur wenn dies nicht der Fall ist, kann sie dem Graphen hinzugefügt werden. Abbildung 3.3 lässt erkennen, dass sich zwischen zwei Kreisen immer vier Tangentenberechnungen ergeben: zwei äußere und zwei innere. Für die Berechnung der inneren und äußeren Tangenten muss $O_1 \neq O_2$ und $y_{O_1} \leq y_{O_2}$ gelten. Trifft die zweite Bedingung nicht zu, wird O_1 mit O_2 getauscht. Weiterhin ist die Funktion $atan2(y,\ x)$ wie folgt definiert:

$$atan2(y,\ x) = 2 \arctan\left(\frac{y}{\sqrt{x^2+y^2}+x}\right)$$

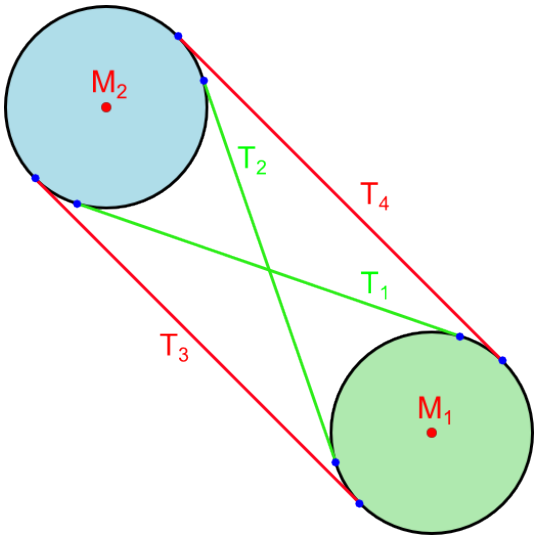


Abb. 3.3: Die vier Tangenten zweier Kreise
O_2 ist durch eine blaufarbene und O_1 ist durch eine grünfarbene Füllung gekennzeichnet. Die beiden inneren Tangenten T_1 und T_2 sind grünfarben und die beiden äußeren Tangenten T_3 und T_4 sind rotfarben dargestellt.

Aus Gründen der Einfachheit beziehen sich die folgenden Darlegungen zu den Tangentenberechnungen immer nur auf eine Tangente, die Berechnungen der zweiten Tangenten erfolgen analog. Die mathematischen Ausführungen sind dahingegen vollständig.

Berechnung der inneren Tangenten

Der zugrunde liegende Gedanke bei dieser Berechnung ist, dass die innere Tangente sich parallel zu einer Geraden befindet, welche vom Mittelpunkt des Kreises M_1 zu dem Punkt C_S liegt (siehe Abbildung 3.4a). C_S befindet sich dabei auf einem Kreisrand, welcher den Mittelpunkt in M_2 hat. Zunächst müssen die Seitenlängen des Dreiecks $\triangle C_S M_1 M_2$ ermittelt werden. Dies ist mit dem Satz des Pythagoras ohne weiteres möglich.

$$\left|\overline{C_S M_2}\right| = r_{O_1} + r_{O_2}$$

$$\left|\overline{M_1 M_2}\right| = \sqrt{(x_{M_2} - x_{M_1})^2 + (y_{M_2} - y_{M_1})^2}$$

$$\left|\overline{C_S M_1}\right| = \sqrt{\left|\overline{M_1 M_2}\right|^2 - \left|\overline{C_S M_2}\right|^2}$$

Nach der Berechnung der drei Seiten gilt es zu prüfen, ob eine innere Tangentenberechnung überhaupt zulässig ist. Dazu muss $(r_{O_1} > 0) \vee (r_{O_2} > 0)$ und $\left|\overline{M_1 M_2}\right| > \left|\overline{C_S M_2}\right|$ gelten. Die erste Regel garantiert dabei, dass zwischen Start- und Zielhindernis keine

innere Tangentenberechnung stattfindet. Die zweite Regel stellt sicher, dass sich die beiden Kreise nicht schneiden bzw. nicht ineinander liegen. Im Anschluss können die Winkel α und β mittels trigonometrischer Funktion ermittelt werden. α entspricht dabei dem Winkel zwischen der X-Achse und $\overline{M_1M_2}$ und β ist gleich $\angle C_S M_1 M_2$.

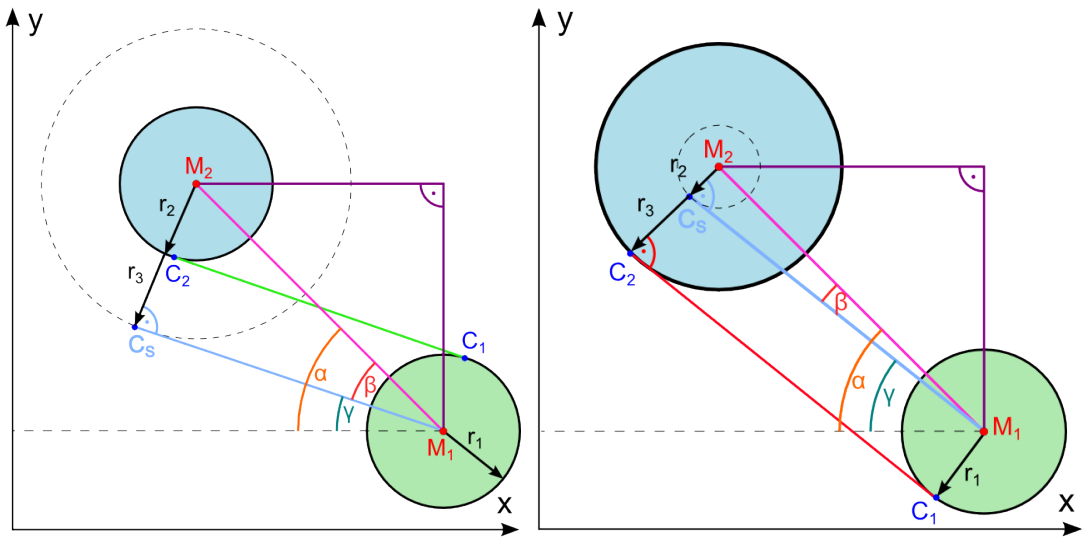
$$\beta = \left| atan2\left(\left|\overline{C_S M_2}\right|, \left|\overline{C_S M_1}\right| \right) \right|$$

$$\alpha = |atan2(y_{M_1} - y_{M_2},\, x_{M_1} - x_{M_2})|$$

Mit der Berechnung des Winkels γ liegen alle erforderlichen Werte für die finalen Berechnungen der Tangentenpunkte vor.

$$T_1 = \left(\begin{array}{l} C_1 = \left(\begin{array}{l} x_1 = x_{O_2} - r_{O_2} * \sin(\gamma) \\ y_1 = y_{O_2} - r_{O_2} * \cos(\gamma) \end{array} \right) \\ C_2 = \left(\begin{array}{l} x_2 = x_{O_1} + r_{O_1} * \sin(\gamma) \\ y_2 = y_{O_1} + r_{O_1} * \cos(\gamma) \end{array} \right) \end{array} \right) \; mit\ \gamma = \alpha - \beta$$

$$T_2 = \left(\begin{array}{l} C_1 = \left(\begin{array}{l} x_1 = x_{O_2} + r_{O_2} * \sin(\gamma) \\ y_1 = y_{O_2} + r_{O_2} * \cos(\gamma) \end{array} \right) \\ C_2 = \left(\begin{array}{l} x_2 = x_{O_1} - r_{O_1} * \sin(\gamma) \\ y_2 = y_{O_1} - r_{O_1} * \cos(\gamma) \end{array} \right) \end{array} \right) \; mit\ \gamma = \alpha + \beta$$



(a) Berechnung der inneren Tangenten (b) Berechnung der äußeren Tangenten

Abb. 3.4: Tangentenberechnungen
O_2 ist durch eine blaufarbene und O_1 ist durch eine grünfarbene Füllung gekennzeichnet. Die innere Tangente ist grünfarben, die äußere Tangente ist rotfarben, $\overline{C_S M_1}$ ist hellblaufarben und $\overline{M_1 M_2}$ ist pinkfarben dargestellt.

Berechnung der äußeren Tangenten

Das mathematische Vorgehen bei der äußeren Tangentenberechnung ist nahezu identisch mit der Kalkulation der inneren Tangenten. So sind die Berechnungen für $\left|\overline{M_1M_2}\right|$, $\left|\overline{C_SM_1}\right|$, α und β vollständig analog. Der Ablauf differenziert sich lediglich in der Kalkulation von $\left|\overline{C_SM_2}\right|$, γ, T_3 und T_4 sowie der Definition alternativer Vorraussetzungen. So gilt für die Berechnung $(r_{O_1} > 0) \wedge (r_{O_2} > 0)$, wodurch sichergestellt wird, dass zum Start- sowie zum Zielhindernis lediglich die innere Tangentenberechung angewendet wird. Weiterhin gilt, wenn $r_{O_1} = r_{O_2}$, dann muss $\left|\overline{M_1M_2}\right| > 0$ sein. Sollte dahingegen $r_{O_1} \neq r_{O_2}$ gelten, muss $\left|\overline{M_1M_2}\right| > \max(r_{O_1},\, r_{O_2}) - min\,(r_{O_1},\, r_{O_2})$ erfüllt sein. Diese Bedingungen sind erforderlich, da die beiden Kreise andernfalls ineinander oder deckungsgleich übereinander liegen können. Die abweichenden Berechnungen sehen wie folgt aus:

$$\left|\overline{C_SM_2}\right| = r_{O_1} + r_{O_2}$$

$$T_3 = \left(\begin{array}{l} C_1 = \left(\begin{array}{l} x_1 = x_{O_2} - r_{O_2} * \sin(\gamma) \\ y_1 = y_{O_2} - r_{O_2} * \cos(\gamma) \end{array} \right) \\ C_2 = \left(\begin{array}{l} x_2 = x_{O_1} - r_{O_1} * \sin(\gamma) \\ y_2 = y_{O_1} - r_{O_1} * \cos(\gamma) \end{array} \right) \end{array} \right) \; mit\ \gamma = \left\{ \begin{array}{l} \alpha - \beta,\ für\ \left|\overline{C_SM_2}\right| > 0 \\ \alpha + \beta,\ für\ \left|\overline{C_SM_2}\right| \leq 0 \end{array} \right.$$

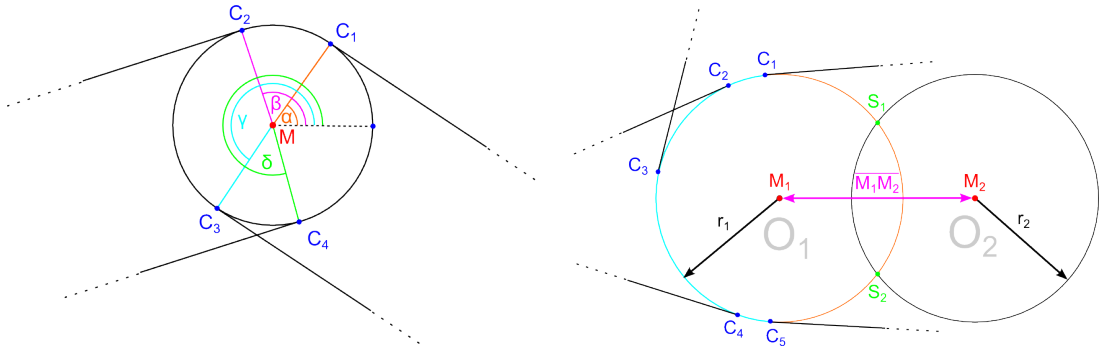
$$T_4 = \left(\begin{array}{l} C_1 = \left(\begin{array}{l} x_1 = x_{O_2} + r_{O_2} * \sin(\gamma) \\ y_1 = y_{O_2} + r_{O_2} * \cos(\gamma) \end{array} \right) \\ C_2 = \left(\begin{array}{l} x_2 = x_{O_1} + r_{O_1} * \sin(\gamma) \\ y_2 = y_{O_1} + r_{O_1} * \cos(\gamma) \end{array} \right) \end{array} \right) \; mit\ \gamma = \left\{ \begin{array}{l} \alpha + \beta,\ für\ \left|\overline{C_SM_2}\right| > 0 \\ \alpha - \beta,\ für\ \left|\overline{C_SM_2}\right| \leq 0 \end{array} \right.$$

3.1.3 Berechnung der Kreissegmente

Die Kreissegmente[3] stellen eine weitere Art von Pfadelement im erweiterten Sichtbarkeitsgraphen dar. Für ihre Berechnung muss die Bedingung $r > 0$ erfüllt sein, wodurch das Start- sowie das Zielhindernis aus den Berechnungen herausgenommen werden können. Für jedes Hindernis werden alle Berührungspunkte C_i der anliegenden Tangenten ermittelt (siehe Abbildung 3.5a). Für jeden dieser C_i-Punkte wird dann, in Abhängigkeit zum Mittelpunkt M, mittels $atan2(y_{C_i} - y_M,\, x_{C_i} - x_M)$-Funktion ein Winkel berechnet. Sollte der errechnete Winkel negativ sein (was bei $y_{C_i} < y_M$ eintritt), muss dieser mit $\pi * 2$ addiert werden. Dies ist nötig, da die Winkel im Anschluss mit

[3] Im weiteren Textverlauf auch als Kreisabschnitte bezeichnet.

Hilfe der Kleiner-Relation sortiert werden sollen, um eine geordnete Menge zu bilden. Durch den Sortiervorgang lässt sich dann darauf rückschließen, welche C_i eine direkte Nachbarschaftsbeziehung auf der Kreisbahn zueinander besitzen. Jeweils benachbarte C_i können als Kreissegment erfasst werden. In Abbildung 3.5a ist dies zum Beispiel von C_1 zu C_2 und von C_4 zu C_1 der Fall.



(a) Vier Kreissegmente mit den dazugehörigen Berührungspunkten C_i und unterschiedlich farbig gekennzeichnete Winkelmaße

(b) Ein Hindernis, welches von einem zweiten Hindernis überlagert wird. Valide Kreissegmente sind hellblaufarben und nicht valide Kreissegmente sind orangefarben abgebildet.

Abb. 3.5: Berechnung der Kreissegmente

Bewegt sich ein Roboter auf einer Kreisbahn muss ausgeschlossen werden können, dass er sich in den Hindernisraum eines zweiten Roboters hineinbegibt. Beispielhaft für dieses Verhalten wäre ein Roboter, der in Abbildung 3.5b von Position C_2 aus versucht, im Uhrzeigersinn um O_1 zu C_4 zu gelangen und dabei dem Hindernis O_2 zu nahe kommt. Um dies Verhalten zu unterbinden, muss zwangsweise jenes Segment entfernt werden, welches teilweise von O_2 überlagert wird. Identifiziert werden kann dieser Kreisabschnitt, indem ausgehend von $\overline{M_1M_2}$ linksdrehend sowie rechtsdrehend um M_1 die nächst liegenden Berührungspunkte C_i bestimmt werden. Diese bilden das zu entfernende Kreissegment.

3.2 Praktische Einschränkungen und daraus resultierende Modifikationsmöglichkeiten des Verfahrens

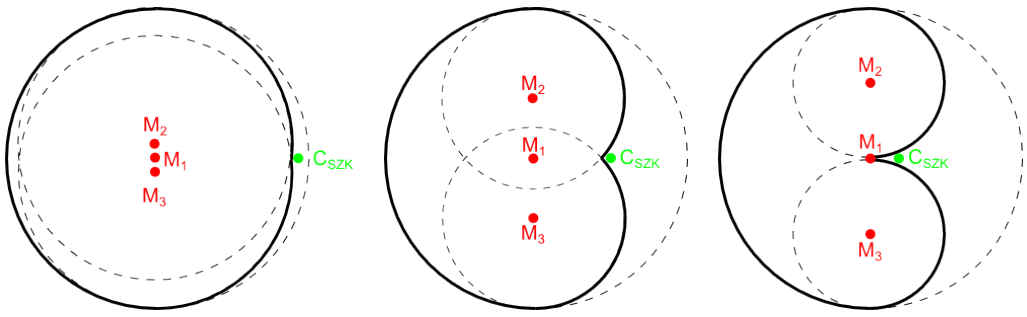
Für jede annehmbare Startkonfiguration im freien Konfigurationsraum kann ohne weiteres eine Bewegungsrichtung ermittelt werden. Dies ist dem Umstand geschuldet, dass an der Startkonfiguration anliegende Pfadstücke immer Tangenten sind, aus welcher die Bewegungsrichtung des Roboters gleich hervorgeht. Das Modell des erweiteren

Sichtbarkeitsgraphen beruht auf der Annahme, dass sich Start- und Zielkonfiguration zu jedem annehmbaren Zeitpunkt außerhalb des Hindernisraums befinden. Im regulären Spielbetrieb erscheint diese Hypothese jedoch als nicht haltbar. Aus diesem Grund muss die Frage gestellt werden, wie das Modell des erweiterten Sichtbarkeitsgraphen so modifiziert werden kann, dass auch innerhalb des Hindernisraums eine valide Pfadplanung zustande kommt. Für diese Problemstellung konnten dabei drei Lösungsmöglichkeiten erarbeitet werden, welche im Folgenden diskutiert werden sollen.

Die Modifikationen in diesem Abschnitt gelten zumeist für die Start- sowie die Zielkonfiguration. Aus diesem Grund sollen beide Punkte in verkürzter Schreibweise als *SZK*, gesprochen *Start- bzw. Zielkonfiguration*, bezeichnet und zusammengefasst werden.

3.2.1 Diversifikation des Hindernisses in drei Kreissegmente

Im Falle der Penetration eines Hindernisses durch die *SZK*, wird das Hindernis nicht mehr durch einen Kreis repräsentiert, sondern durch drei Teilkreise. Merkmale wie Position und Radius bleiben dem ursprünglichen Kreis O_a dabei erhalten, lediglich wird die *SZK* zugewandte Hälfte entfernt, wodurch ein Halbkreis entsteht. In die nun offene Seite von O_a werden die zwei kleineren, gleich großen Kreise O_{i_1} und O_{i_2} platziert. Aus O_{i_1} und O_{i_2} werden nun die Kreissegmente entfernt, welche nicht Teil der äußeren Hülle aus dem Gebilde von O_a, O_{i_1} und O_{i_2} sind. Der Abstand von O_{i_1} und O_{i_2} zu O_a, sowie die Radien von O_{i_1} und O_{i_2} korrelieren dabei mit der Penetrationstiefe der *SZK* (siehe Abbildungen 3.6 im Vergleich). Je tiefer die *SZK* eindringt, desto kleiner werden die Radien und umso weiter entfernen sich die Mittelpunkte M_2 und M_3 von M_1.



(a) Oberflächliche Penetration des Hindernisses (b) Mittlere Penetration des Hindernisses (c) Tiefe Penetration des Hindernisses

Abb. 3.6: Hindernisse, bestehend aus drei Kreis-Figuren, dargestellt mit unterschiedlichen Penetrationstiefen

Ein Vorteil dieser Modellierung besteht darin, dass die Korrektur an die gewünschte äußere Kreisbahn von O_a und der Pfadverlauf zu anliegenden Tangenten sehr harmonisch verläuft, wodurch es zu keinen abrupten Richtungsänderungen in der Bewegungsbahn

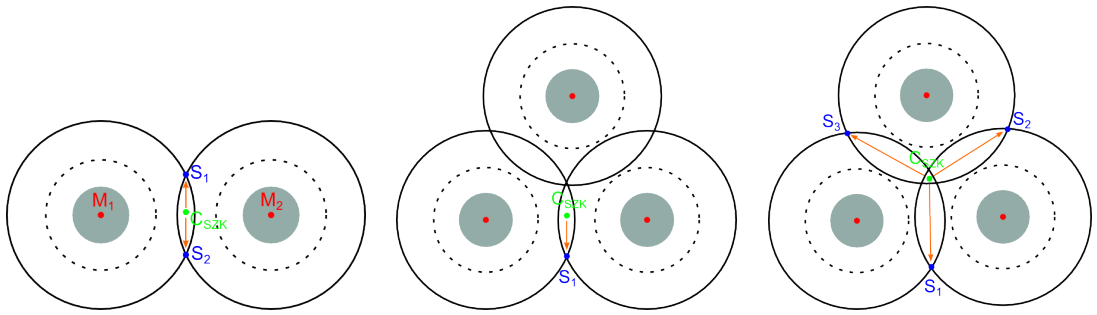
des Roboters kommt. Des weiteren verstärkt sich die Korrektur der Bewegungsrichtung, umso näher die Startkonfiguration dem eigentlichen Hindernis kommt. Da die Startkonfiguration sich zu jedem beliebigen Zeitpunkt außerhalb der Dreikreisfigur befindet, ist das an der Startkonfiguration anliegende Pfadstück immer eine Tangente, aus welcher die Bewegungsrichtung ohne weiteres abgeleitet werden kann. Nachteilig erweisen sich hingegen, dass Modifikationen im Datenmodell, bei der Berechnung der Kreissegmente und dem Verfahren zur Bestimmung der Sichtbarkeit von zwei Punkten vorgenommen werden müssen. Eine Implementierung zu Testzwecken hat aufgezeigt, dass die Änderungen am Code recht komplex sind. Durch das Hinzufügen von zwei weiteren Hindernisse und die daran gekoppelten Nachfolgeberechnungen wird zudem das Laufzeitverhalten verschlechtert. Überdies lässt sich der Spezialfall, dass sowohl Start- als auch Zielpunkt sich im Radius eines Hindernisses befinden, nur unzulänglich modellieren. Gleiches gilt für den Fall, dass die *SZK* von mehreren Hindernissen überlagert werden. Die Betrachtung der Abbildungen 3.6 legt nahe, dass eine Korrektur der Fehlstellung nicht in ausreichender Geschwindigkeit erfolgt. D.h.: Ein Roboter bewegt sich erst dann geradling von einem Hindernis weg, wenn er den Mittelpunkt fast erreicht hat. Zu diesem Zeitpunkt ist die Kollision jedoch schon erfolgt. Dies bildet das ausschlaggebende Argument, diese Methode nicht zu verwenden.

3.2.2 Geometrische Zweipunktverschiebung

Der Ansatz dieser Methode ist in zwei Schritte gegliedert. In erster Instanz muss der A*-Algorithmus dazu befähigt werden, einen Weg ermitteln zu können. Hierfür ist es notwendig, mehrere Pfadstücke nach einem festgelegten Schema einzufügen. Die manuell eingegliederten Pfadelemente weisen jedoch die Eigenschaft auf, nicht mehr vollkommen glatt in andere Pfadstücke überzugehen. Aus diesem Grund muss in einem zweiten Schritt der Bewegungsablauf durch eine Punktverschiebung geglättet werden. Im Gegensatz zu den beiden anderen Methoden wird bei der geometrischen Zweipunktverschiebung die Form und Größe der überlappenden Hindernisse nicht verändert.

Noch vor der Tangentenberechnung werden Pfadstücke eingesetzt, die von der *SZK* auf kürzestem Wege zur äußeren Kreisbahn des Hindernisses führen. Bei einer einfachen Überlagerung kann dies ohne viel Aufwand geschehen. Ein Konzept zur Lösung simultaner Überlagerung kann darin bestehen, von der *SZK* zu den Schnittpunkten S_i der sich überlagernden Kreise Pfadstücke $\overline{C_{SZK}S_i}$ einzufügen (siehe Abbildung 3.7). Die Pfadstücke sind dabei nur zulässig, wenn der jeweilige S_i-Punkt nicht von einem weiteren Hindernis überlagert wird. An dieser Stelle kann es dazu kommen, dass die Situation nur mit unverhältnismäßig hohem Aufwand analysiert werden kann, um eine elegante Lösung zu finden. Aus diesen Grund sollte aus der Menge der $\overline{C_{SZK}S_i}$ nur das

kürzeste Pfadelement im Graph vorzufinden sein.



(a) Die *SZK* wird durch zwei Hindernisse überlagert.

(b) Die *SZK* wird durch zwei Hindernisse überlagert, wobei eine Richtung durch ein weiteres Hindernis blockiert wird.

(c) Die *SZK* wird gleichzeitig von drei Hindernissen überlagert.

Abb. 3.7: Mehrfache Hindernisüberlagerung

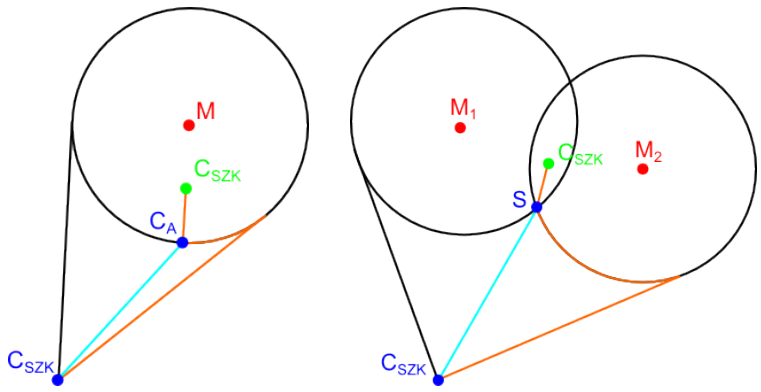
Die Schnittpunkberechnung kann dabei folgendermaßen vorgenommen werden, wobei K ein konstanter Wert ist, der für alle vier Koordinatenberechnungen benötigt wird:

$$K = \sqrt{(\left|\overline{M_1M_2}\right| + r_1 + r_2)(-\left|\overline{M_1M_2}\right| + r_1 + r_2)(\left|\overline{M_1M_2}\right| - r_1 + r_2)(\left|\overline{M_1M_2}\right| + r_1 - r_2)}$$

$$S_1 = \begin{pmatrix} x = \frac{1}{2}\left(x_1 + x_2 - \frac{(r_1^2 - r_2^2)(x_1 - x_2) + K(y_1 - y_2)}{\left|\overline{M_1M_2}\right|^2}\right) \\ y = \frac{1}{2}\left(y_1 + y_2 - \frac{(r_1^2 - r_2^2)(y_1 - y_2) - K(x_1 - x_2)}{\left|\overline{M_1M_2}\right|^2}\right) \end{pmatrix}$$

$$S_2 = \begin{pmatrix} x = \frac{1}{2}\left(x_1 + x_2 - \frac{(r_1^2 - r_2^2)(x_1 - x_2) - K(y_1 - y_2)}{\left|\overline{M_1M_2}\right|^2}\right) \\ y = \frac{1}{2}\left(y_1 + y_2 - \frac{(r_1^2 - r_2^2)(y_1 - y_2) + K(x_1 - x_2)}{\left|\overline{M_1M_2}\right|^2}\right) \end{pmatrix}$$

Eine weitere Problemstellung lässt sich anhand der Abbildungen 3.8a und 3.8b festmachen. Der geplante Weg führt in beiden Fällen unnötigerweise entlang des orangefarbenen Kreissegments. Diesem Umstand kann entgangen werden, indem von dem Punkt C_A bzw. S die Tangenten zu allen Hindernissen berechnet werden. In den angeführten Beispielen sind die gefundenen Tangenten hellblau gekennzeichnet.



(a) (b)

Abb. 3.8: Manuelle Platzierung von Tangenten

Im zweiten Schritt muss nun die Bewegungsrichtung des Roboters angepasst werden, um einen geschmeidigeren Bewegungsablauf zu ermöglichen. Die zum Tragen kommende Logik, sobald sich die Startkonfiguration im Hindernisraum befindet, soll anhand der Abbildungen 3.9a und 3.9b erläutert werden. Die Konstellation der dargestellten Punkte ist wie folgt geregelt:

- Die beiden abgebildeten Kreise sind der äußere und der mittlere Kreis, nach der Definition aus Abschnitt 3.1.1.
- Die anzunehmende Bewegungsrichtung entspricht der Strecke c.
- Es existiert eine Sakente, die durch den Mittelpunkt M führt und durch den Startpunkt C_{Start}. Auf dieser Sakente liegen ebenfalls die Punkt C_1 und C_2.
- Der Punkt C_1 liegt auf dem mittleren Kreis.
- Auf der äußeren Kreisbahn des Hindernisses sind die Punkte C_2, C_D und C_3 platziert, wobei C_D und C_3 sich auch alternativ auf einer anliegenden Tangente befinden können (siehe Abbildung 3.9b).
- Die Distanz von C_1 zu C_2 ist äquivalent zur Länge des Kreisbogens von C_2 zu C_3.
- Die Länge des Abschnitts d ist gleich der Länge des Kreisbogens d' und demzufolge ist auch die Länge b äquivalent zu der Länge von b'.
- Je weiter C_{Start} in Richtung C_1 vordringt, desto weiter wird der Punkt C_D auf der Kreisbahn Richung C_2 gezogen. Dadurch wird b und b' größer und d und d' kleiner. Die Wertänderungen verhalten sich dabei proportional zueinander.
- Passiert C_{Start} auf dem dem Weg Richtung M den Punkt C_1, so bleibt C_D auf C_2 liegen.

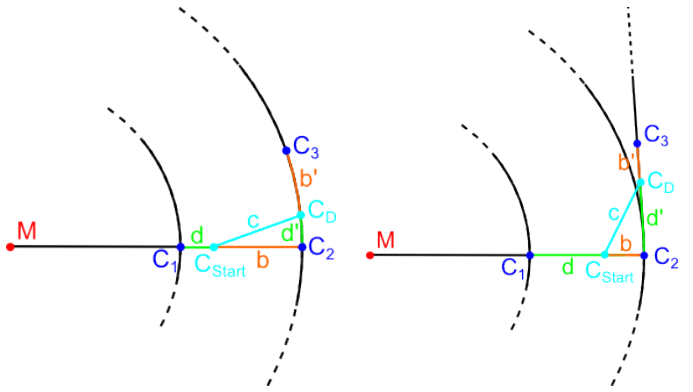


Abb. 3.9: Bestimmung der Bewegungsrichtung mittels geometrischer Zweipunktverschiebung

Wird die Bewegungsrichtung wie angeführt kalkuliert, ergibt sich ein harmonischer Kurvenverlauf, welcher in approximierter Form in Abbildung 3.10b dargestellt wird. In Abbildung 3.10a sind mehrere berechnete Bewegungsrichtungen dargestellt. Aus der Zeichnung geht hervor, dass Roboter, die einmal den Hindernisraum betreten haben, sich anfänglich dem Mittelpunkt nähern, bevor eine Korrektur erfolgt (siehe pinkfarbene Strecke). Aus diesem Grund ist es empfehlenswert, den Radius des äußeren Kreises zu vergrößern.

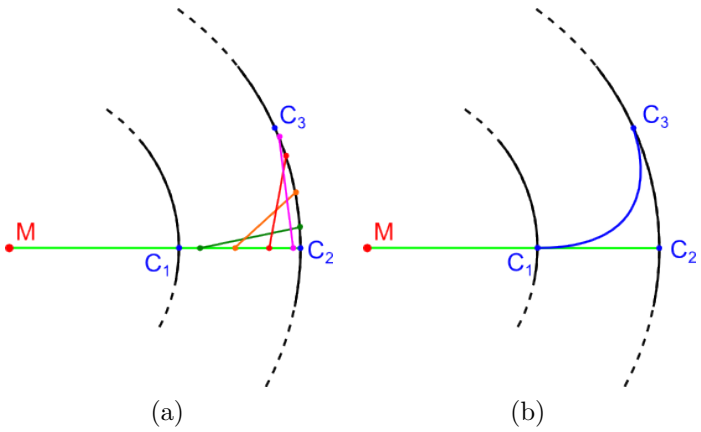


Abb. 3.10: Resultierende Trajektorie mittels geometrischer Zweipunktverschiebung

Die vorgestellte Methode zur Modifikation des erweiterten Sichtbarkeitsgraphen stellt sich als durchaus geeignetes Mittel zur Pfadgenerieung innerhalb des Hindernisraums heraus. Eine Implementierung zu Testzwecken hat zudem ergeben, dass die Ergebnisse zur Berechnung der Bewegungsrichtung plausibel und nachvollziehbar sind. Es bleibt positiv anzumerken, dass die abstoßende Kraft umso stärker wirkt, je näher der Roboter einem Hindernis kommt. Durch eine geeignete Parameterwahl kann das Verhalten zudem noch angepasst werden.

Als Resultat der durchgeführten Untersuchung scheidet diese Methode jedoch aus, da die anfallenden Modifikationen im Ganzen sehr zahlreich sind und es noch einer weiteren Verfeinerung bedarf, da nicht jede Situation optimal gelöst ist. Allem voran müssen komplexe Gebilde, bei denen zahlreiche Überlagerungen der *SZK* erfolgen, weiter untersucht werden. Innerhalb des Hindernisraums könnte das Laufverhalten auch durch die Verwendung von Bezierkurven realisiert werden. Der Einsatz parametrisch modellierter Kurven zur Konstruktion von Pfadelementen innerhalb des Hindernisraums erscheint als durchaus elegantes Mittel. Eine ganzheitliche Betrachtung der experimentellen Ergebnisse soll nicht Bestandteil dieser Arbeit sein, da auch nach umfangreichen Versuchen keine zufriedenstellende Anordnung von Kontrollpunkten ermittelt werden konnte.

3.2.3 Reduktion der Kreisradien

Das praktische Problem der Wegfindung innerhalb des Hindernisraums wird ebenfalls in [Wei98] beschrieben. Die dort vorgestellte Lösung besteht darin, den Radius des entsprechenden Hindernisses zu reduzieren, bis die *SZK* außerhalb des Hindernisraums liegt. Diese Lösung ist einfach zu implementieren und erfordert nur eine geringe Anzahl von Berechnungen, doch impliziert sie keine unmittelbare Korrektur der Fehlstellung. Der Roboter könnte einem Hindernis somit immer näher kommen, ohne dass dieser eine Revision des Kurses einleitet. Wünschenswert erscheint ein Verhalten, bei dem die Stärke der Korrekturmaßnahme abhängig von der Entfernung zum Hindernis ist.

Inspiriert durch die Potentialfeldmethode, stellt die Repräsentation der Bewegungsrichtung als Vektor den entscheidenden Schlüssel dar, um diese Unzulänglichkeit beseitigen zu können. Auf unkomplizierte Weise können Hindernisse, welche die Startkonfiguration überlagern, eine weggerichtete Kraft auf den Roboter ausüben, indem deren Wirkung ebenfalls als Vektoren dargestellt und auf den Bewegungsvektor aufaddiert wird. Die Stärke der einwirkenden Kräfte von Hindernissen skaliert dabei mit deren Entfernung zur Startkonfiguration. Im Detail wird der Bewegungsvektor wie folgt berechnet:

- $\overrightarrow{d}$ ist der gesuchte Bewegungsvektor.
- $\overrightarrow{t}$ ist der normierte ursprüngliche Bewegungsvektor. Dieser Vektor weist dabei die gleiche Ausrichtung wie die Tangente, die den Kreis an dem Punkt C_{Start} berührt, auf.
- $\overrightarrow{f_i}$ ist ein normierte Kraftvektor, der die abstoßende Kraft von überlagernden Hindernissen abbildet. Die Ausrichtung entspricht der Strecke $\overline{M_i C_{Start}}$.
- p_i ist ein positiver Skalierungsfaktor der um so größer wird, je weiter C_{Start} in den Hindernisraum eingedringt. Es gilt $p_i \in \mathbb{R}$ und $0 \leq p_i \leq 1$. Der Wert wächst

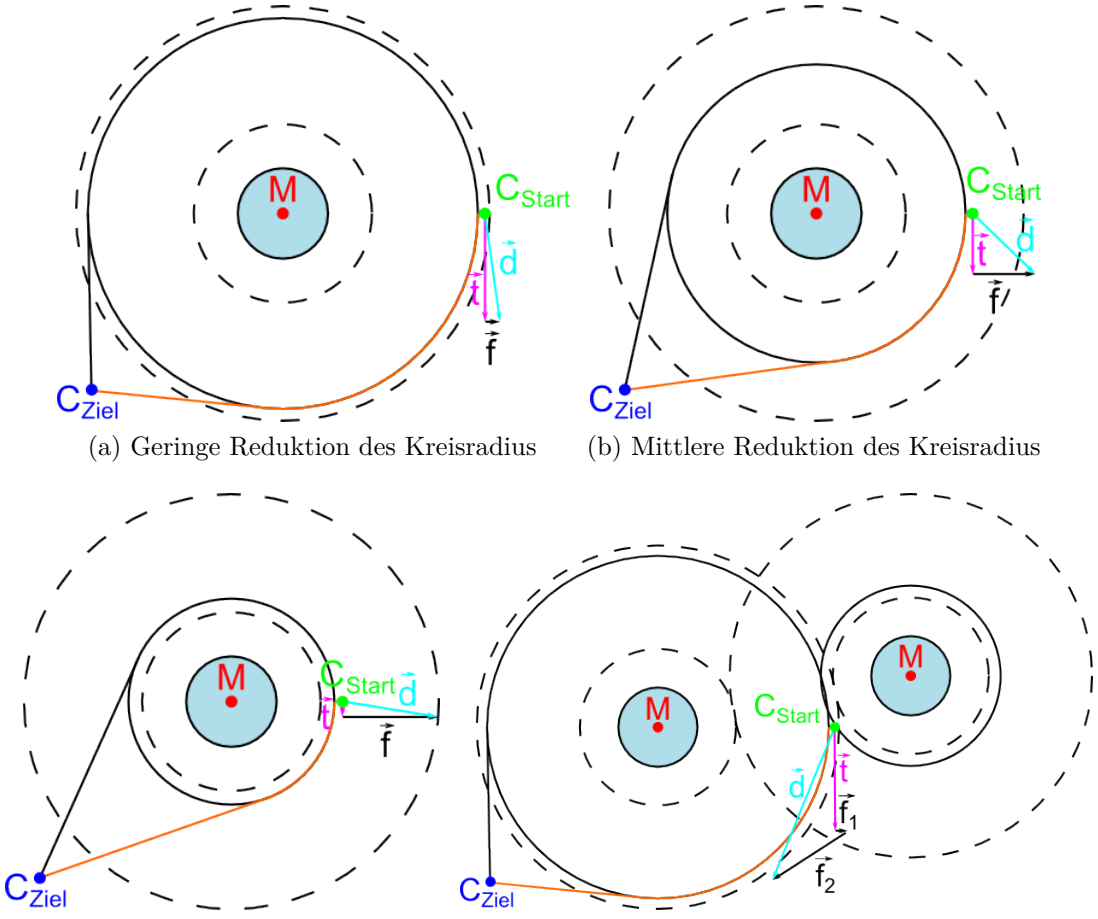
linear und ergibt 1, wenn $\left|\overline{C_{Start}M_i}\right|$ kleiner oder gleich dem Radius des mittleren Kreises ist.

- n ist die Anzahl der Hindernisse, die C_{Start} überlagern.

$$\overrightarrow{d} = \overrightarrow{t} * (1-p) + \overrightarrow{f} * p, \ für\ n = 1$$

$$\overrightarrow{d} = \overrightarrow{t} + \overrightarrow{f_1} * p_1 + \overrightarrow{f_2} * p_2 + ... + \overrightarrow{f_n} * p_n, \ für\ n > 1$$

Anhand unterschiedlicher Konstellationen soll das Verhalten in den nachfolgenden Abbildungen graphisch dargestellt werden:



(a) Geringe Reduktion des Kreisradius (b) Mittlere Reduktion des Kreisradius

(c) Große Reduktion des Kreisradius (d) Simultane Reduktion von zwei Kreisradien

Abb. 3.11: Reduktion der Kreisradien

Der äußere sowie der mittlere Kreis werden in gestrichelter Form dargestellt. Der kürzeste Pfad ist orangefarben, der resultierende Bewegungsvektor $\overrightarrow{d}$ ist hellblaufarben, der ursprüngliche Bewegungsvektor $\overrightarrow{t}$ ist pinkfarben und die Kraftvektoren $\overrightarrow{f_i}$ sind schwarz dargestellt.

Die Reduktion der Kreisradien inklusive der Modifikation zur Berechnung der Bewegungsrichtung benötigt nur eine geringe Anzahl von Rechenoperationen und liefert

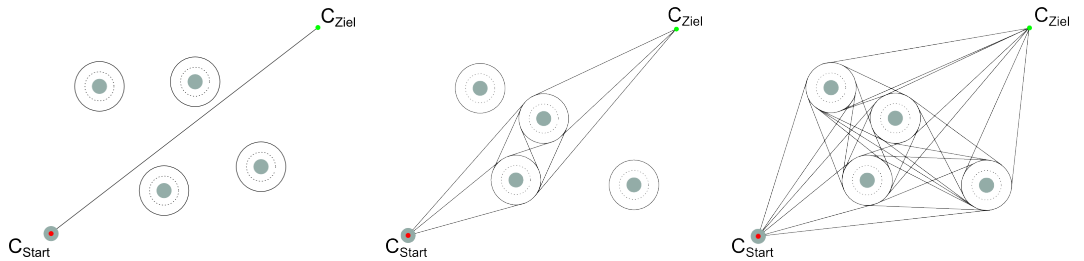
auch in komplizierten Umgebungsverhältnissen überzeugende Resultate. Darüber hinaus konnten keine markanten Schwächen für diesen Lösungsansatz festgestellt werden. Unter den drei vorgestellten Lösungen scheint diese somit die vielversprechendste zu sein und wird damit als Teil der finalen Implementierung ausgewählt.

3.3 Laufzeitoptimierung

Eines der Kernziele der Entwicklung stellt die Maximierung der Laufzeiteffizienz dar. Aufgrund dessen sollen in diesem Abschnitt drei substanzielle Optimierungen expliziert werden.

3.3.1 Selektive Entfaltung des Graphens

In Abschnitt 3.4 wird dargelegt, dass die Anzahl an Hindernissen polynomiell in die Laufzeit des Programms eingeht. Es erscheint dabei als evident, dass Hindernisse, die vollkommen abwegig von dem dirkten Weg zwischen Start- und Zielposition liegen, gar keinen Einfluss auf die praktische Wegfindung haben (zum Vergleich, siehe Abbildung 3.12b und Abbildung 3.12c). Demzufolge ist die Betrachtung dieser Hindernisse hinfällig. Um dieses Verhalten realisieren zu können, baut sich der Graph rekursiv auf. Als Ausgangspunkt ist der minimale Graph anzusehen (siehe Abbildung 3.12a). Er besteht lediglich aus einer Tangente zwischen Start- und Zielpunkt. Wird diese Tangente von einem oder mehreren Hindernissen geschnitten, werden diese Hindernisse mit in den Graphen einbezogen und die geschnittene Tangente entfernt. Im Anschluss werden für die neu hinzugefügten Hindernisse die Tangenten berechnet und geprüft, ob diese von weiteren Hindernissen überlagert werden, um sie dem Graphen hinzuzufügen. Dies geschieht so lange, bis keine Hindernisse mehr hinzugefügt werden müssen.



(a) Minimaler Graph, bestehend aus Start- und Zielpunkt sowie einer Tangente

(b) Hinreichend gespannter Teilgraph

(c) Vollständig gespannter Graph

Abb. 3.12: Selektive Graphenentfaltung

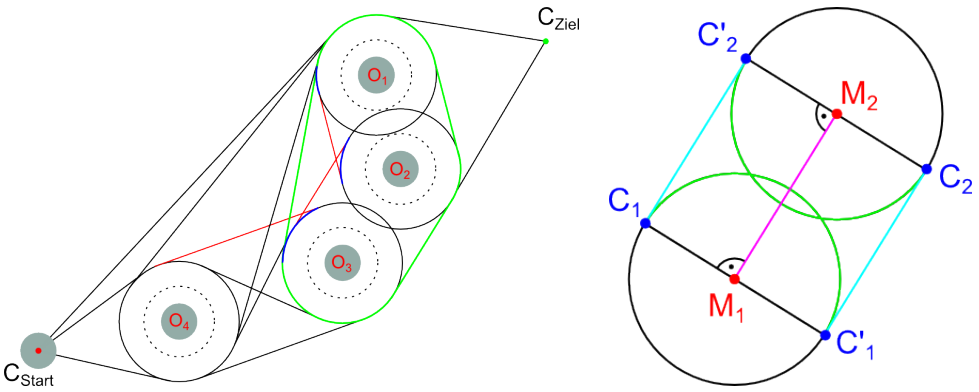
Eine weitere Methode zur Bestimmung der relevanten Hindernisse könnte realisiert werden, indem von jedem Hindernis das Lot auf die Tangente des minimalen Graphen gefällt wird[4]. Übersteigt die Länge der Lotstrecke einen bestimmten Wert, welcher auch mit der Länge der Tangente zwischen Start- und Zielpunkt skalieren kann, würde das Hindernis keine Beachtung im Graphen finden.

3.3.2 Entfernen nicht relevanter Tangenten und Kreisabschnitte

Tangenten und Kreisabschnitte die in konkave Ecken führen, wie sie in Abbildung 3.13a rot- bzw. blaufarben gekennzeichnet sind, können nie Teil eines kürzesten Pfades sein. Werden diese Kanten aus dem Graphen im Vorfeld herausgefiltert, kann sich gegebenenfalls ein signifikanter Leistungsgewinn bei der Suche nach dem kürzesten Pfad mittels A*-Algorithmus einstellen. Da alle Hindernisse eine konvexe Form aufweisen, können konkave Ecken nur vorkommen, wenn sich mehrere Hindernisse überschneiden. Diese Hindernisse werden selektiert und in separaten transitiven Mengen A_i eingeordnet. In Abbildung 3.13a sind die zusammengefassten Mengen $A_1 = \{O_1, O_2, O_3\}$ und $A_2 = \{O_4\}$. Sind zwei oder mehr Hindernisse in einer Menge enthalten, analysiert der nun initiierte Algorithmus die Hindernisse paarweise, d.h. O_1 mit O_2, O_2 mit O_3 und O_3 mit O_1. Für die Untersuchung werden die bereits im Vorfeld berechneten Berührungspunkte C_i der äußeren Tangenten herangezogen (siehe Abbildung 3.13b). Wird festgestellt, dass eine Tangente oder ein Kreisabschnitt zwischen den Tangentenendpunkten von C_1 zu C_1' bzw. von C_2 zu C_2' endet oder anfängt, kann diese Kante dem Graphen entnommen werden[5].

[4] Diese Methode wäre auch in Kombination mit dem vorgestellten Verfahren kombinierbar. Eventuell wird dies Bestandteil einer späteren Weiterentwicklung des Verfahrens darstellen.

[5] Wird die hier beschriebene Operation noch vor der Generierung der Kreisabschnitte durchgeführt, kann auf die Filterung der Kreissegmente verzichtet werden.



(a) O_1, O_2 und O_3 bilden ein konvexes Gebilde, welches durch die grünfarbenen Linien gekennzeichnet ist. Rot- und blaufarben gekennzeichnete Pfadabschnitte können entfernt werden.

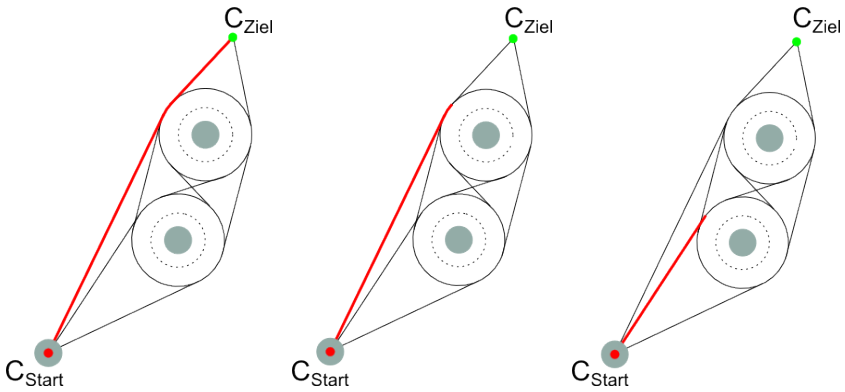
(b) Die äußeren Tangenten sind hellblaufarben gekennzeichnet. Kreissegmente und Tangenten, welche im grünfarbenen Bereich anfangen oder enden, werden entfernt.

Abb. 3.13: Entfernen irrelevanter Tangenten und Kreisabschnitte

Im Abschnitt 3.5 soll durch Laufzeitmessungen geprüft werden, ob das Entfernen der Kanten zeitintensiver ist, als die Kalkulation des kürzesten Pfades im unoptimierten Graphen.

3.3.3 Partielle Suche nach dem kürzesten Pfad

Für eine adäquate Steuerung des Roboters ist es nicht von Nöten zu wissen, wie weiter entfernte Pfadabschnitte (Kanten im Graphen) aussehen. Vielmehr ist nur die lokale Ausrichtung des Roboters, die Bewegungsrichtung und die Geschwindigkeit von elementarem Interesse. Diese drei Faktoren können schon anhand eines einzigen Pfadabschnitts ermittelt werden. Die Berechnung weiterer Pfadabschnitte erhöht lediglich die Präzision der Längenabschätzung des gesamten Pfades und verbessert somit die Qualität des gefundenen Wegs. Es ist vielmals festzustellen, dass sich die Wahl des ersten Pfadabschnitts ab einer bestimmten Suchtiefe kaum noch revidiert. Die Verwendung einer Obergrenze für die Suchtiefe könnte somit ein rentabler Trade-Off zwischen Qualitätsverlust des gefundenen Pfades und Begrenzung der maximalen Laufzeit des A*-Algorithmus darstellen.



(a) Suchtiefe größer 3 (b) Suchtiefe von 3 (c) Suchtiefe von 1

Abb. 3.14: Partielle Suche nach dem kürzestem Pfad mit unterschiedlichen Suchtiefen

Abbildung 3.14 zeigt die Ergebnisse einer Pfadsuche mittels A*-Algorithmus anhand unterschiedlicher Suchtiefen. Die Wahl einer geeigneten oberen Schranke für die Suchtiefe soll anhand empirischer Versuche in Abschnitt 3.5 ermittelt werden.

3.4 Laufzeitabschätzung

Bei der Abschätzung der Laufzeitkomplexität soll von wesentlichem Interesse die asymptotische Laufzeit der Implementierung sein. Dies bedeudet, dass das Zeitverhalten des Programms für eine potenziell unendlich große Eingabemenge untersucht wird. Es sollen dabei folgende Symboldefinitionen gelten:

- H: Anzahl an Hindernisse
- T: Anzahl an Tangenten
- K: Anzahl an Knoten im Graphen
- C: Konstante

Asymptotische obere Schranke für die Tangentenberechnung

$$4 * \sum_{n=0}^{H} n + 4H = \frac{H(H-1)}{2} 4 + 4H = 2H(H-1) + 4H = 2H^2 - 2H + 4H = 2H^2 + 2H = \mathcal{O}(H^2)$$

Asymptotische obere Schranke für die Kreisabschnittsberechnung

$$T = 2H^2 + 2H \leq C * H^2$$

$$\mathcal{O}(H * T) \leq \mathcal{O}(H * H^2) = \mathcal{O}(H^3)$$

Asymptotische obere Schranke für die Suche nach dem kürzesten Pfad mittels A*-Algorithmus

Unter der Einschränkung, dass die Heuristik die Kosten nie überschätzt, die Open-Menge als binärer Heap und die Closed-Menge als Array organisiert wird, kann nach [A22hr] die Laufzeit des A*-Algorithmus auf $\mathcal{O}\left(|K|^2\right)$ abgeschätzt werden. Die Anzahl an Knoten kann wie folgt geschätzt werden:

$$K = 2T - 2H + 2 = 4H^2 + 4H - 2H + 2 = 4H^2 + 2H + 2 \leq C * H^2$$

Wird K durch H substituiert, ergibt sich nun eine Gesamtlaufzeit des A*-Algorithmus von:

$$\mathcal{O}\left(H^4\right)$$

Kummulierte Laufzeitabschätzung

$$\mathcal{O}(H^2) + \mathcal{O}(H^3) + \mathcal{O}\left(H^4\right) = \mathcal{O}\left(H^4\right)$$

Es ist unschwer zu erkennen, dass alle analysierten Programmfragmente mit polynomiellem Aufwand lösbar sind. In polynomieller Zeit lösbare Probleme haben die Eigenschaft, dass der Aufwand zur Lösung des Problems schon bei der Eingabe relativ geringer Eingabemengen sehr groß wird. Dadurch befindet sich die Implementierung an einer Grenze zwischen praktisch lösbar und praktisch unlösbar, abhängig von der Größe der Eingabemenge. Da diese Eingabemenge jedoch relativ gering und zudem auch noch nach oben begrenzt ist, sollte dies praktisch kein Problem darstellen. Darüber hinaus sollte die durchschnittliche Programmlaufzeit weit geringer sein, da nur eine partielle Auswertung der Hindernisse stattfindet, siehe Abschnitt 3.3.1.

3.5 Laufzeitverhalten

Im Abschluss des dritten Kapitels wird das Verhalten der eigens programmierten Applikation mittels Benchmarking analysiert. Es soll dabei die Tauglichkeit der im Vorfeld erwähnten Optimierungsstrategien untersucht und ggf. validiert werden. Mit den vorliegenden Messergebnissen können dahingegen keine Rückschlüsse darauf geführt werden, ob die Implementierung performant genug für den Einsatz im Spielgeschehen ist. Eine konkrete Aussage hierfür kann erst erfolgen, wenn der Prototyp in C++ portiert, der Code optimiert und auf der Zielplattform ausgeführt wird. Dies ist jedoch nicht Ziel dieser Arbeit.

Da ausschließlich verschiedene Konstellationen der Programmkomponenten und Spielsituationen analysiert werden, stellt auch die überproportionale Leistungsfähigkeit des Testsystems gegenüber der Nao-Plattform kein Problem dar. In künstlichen CPU-Benchmarks ist das Testsystem um den Faktor 32 schneller, wobei die eigens erstellte Applikation nicht im gleichen Maße von dem Multiprozessorsystem profitieren kann. Zum direkten Vergleich der Prozessoren, siehe [CB-hrb] mit [CB-hra]. Die Systemdaten im Überblick:

Prozessor	Intel Core i7-3840QM (2,80GHz 1600MHz 8MB)
Arbeitsspeicher	24.0GB PC3-12800 DDR3 SDRAM 1600 MHz
Betriebssystem	Ubuntu 13.10 x64 Version mit allen Updates (Stand: 14.02.2014)
Java-SDK	Oracle JDK 7u51

Tab. 3.1: Konfiguration des Testsystems

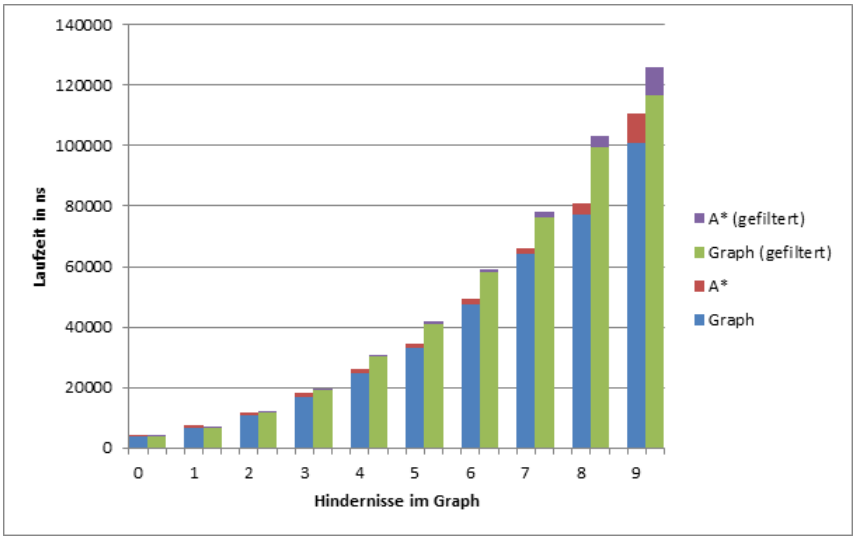
Für die Versuchsreihe wird ein Datensatz, bestehend aus 10.000 verschiedenen Spielkonstellationen, auf alle Laufzeitmessungen angewendet, um somit für die Messungen identische Voraussetzungen zu schaffen. Damit ligatypische Spielsituationen nachgestellt und die Aussagekraft der Testdaten nicht verzerrt werden, sind sowohl die Spielfeldproportionen als auch das Abmaß der Roboter intern in einem eins zu eins Verhältnis dargestellt. Ein jeder Datensatz beinhaltet dabei neun zufällig auf dem Spielfeld verteilte Hindernisse, sowie ein Start- und ein Zielpunkt, welche einen minimalen Abstand von zwei Metern zueinander aufweisen.

Mit Hilfe der ersten Versuchsreihe soll beurteilt werden, ob die Filterung nicht relevanter Pfadabschnitte einen signifikanten Leistungsgewinn verschafft (siehe Abschnitt 3.3.2). Die vorliegenden Daten werden in drei Graphiken aufgeschlüsselt, wobei diese die minimale, die durchschnittliche und die maximale Laufzeit der Gesamtapplikation abbilden. Die Säulen des Diagramms setzen sich dabei aus der benötigten Zeit für die Erstellung des Graphen und der Pfadsuche mittels A*-Algorithmus zusammen. Auf der horizontalen Achse werden die Ergebnisse in die Anzahl der für die Graphenberechnung relevanten Hindernisse aufgeschlüsselt.

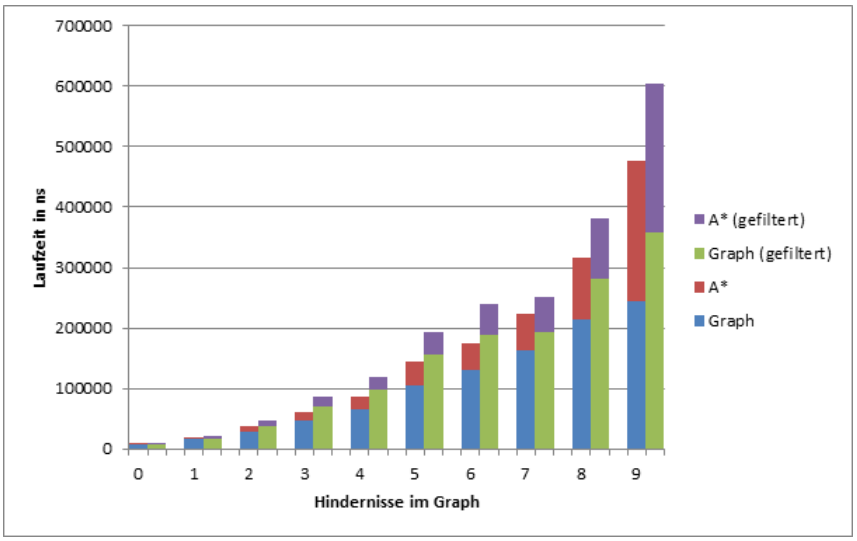
Die Betrachtung der Ergebnisse legt nahe, dass die Tangentenfilterung sich als kein förderndes Mittel zur Laufzeitoptimierung erweist. Im Mittel steigt die Programmlaufzeit um den Faktor 1,26, wobei die Laufzeit für die Suche nach dem kürzesten Pfad fast gar nicht davon profitiert. Vermeintlich ist die Ursache dafür in der nicht unerheblichen Anzahl an Mengenoperationen zu finden. Demzufolge wird das Softwaremodul zur Filterung nicht relevanter Pfadabschnitte kein Bestandteil der finalen Implementierung sein. Gut zu erkennen ist darüber hinaus die überlineare Laufzeitentwicklung anhand der Abbildungen 3.15a und 3.15b. Weniger vorhersehbar entwickelt sich dafür die in Abbildung 3.15c dargestellte maximale Programmlaufzeit. Diese kann in Extremfällen

bis zu 3.993.800 ns bzw. circa 0,004 s benötigen. Da zum gegenwärtigen Zeitpunkt jeder Spieler in einem separaten Areal agiert, ist es als äußerst unwahrscheinlich anzusehen, dass fast alle Hindernisse für die Pfadplanung beachtet werden müssen. Im Allgemeinen ist festzustellen, dass die Gesamtlaufzeit nur zu einem geringen Teil von der Laufzeit des A*-Algorithmus bestimmt wird.

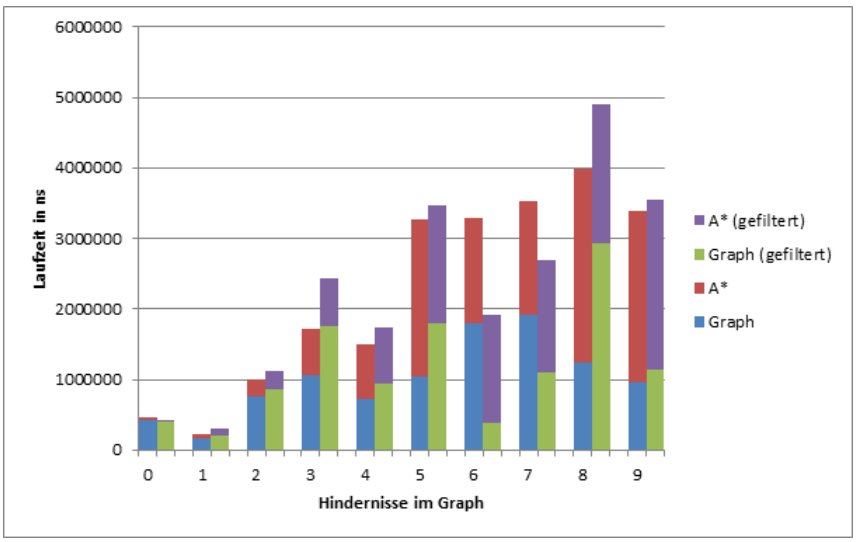
Im zweiten Versuch werden fünf Messreihen gegenübergestellt, welche unterschiedliche Obergrenzen für die Anzahl an Iterationen des A*-Algorithmus definieren. Die Daten werden auch hier in drei Graphiken abgebildet, die die minimale, die durchschnittliche und die maximale Laufzeit des A*-Algorithmus repräsentieren. Die Betrachtung schließt ebenfalls die Untergliederung in die Anzahl der für die Graphenberechnung relevanten Hindernisse auf der horizontalen Achse mit ein.



(a) Minimale Laufzeit

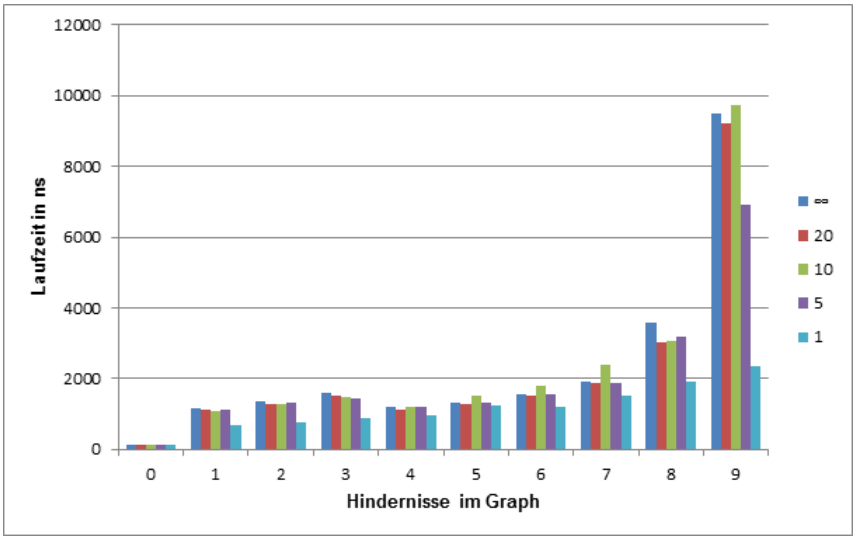


(b) Durchschnittliche Laufzeit

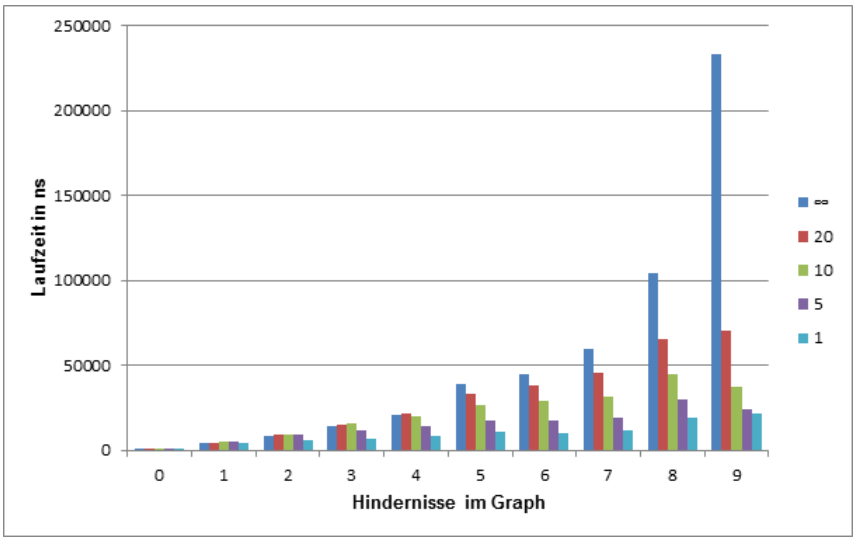


(c) Maximale Laufzeit

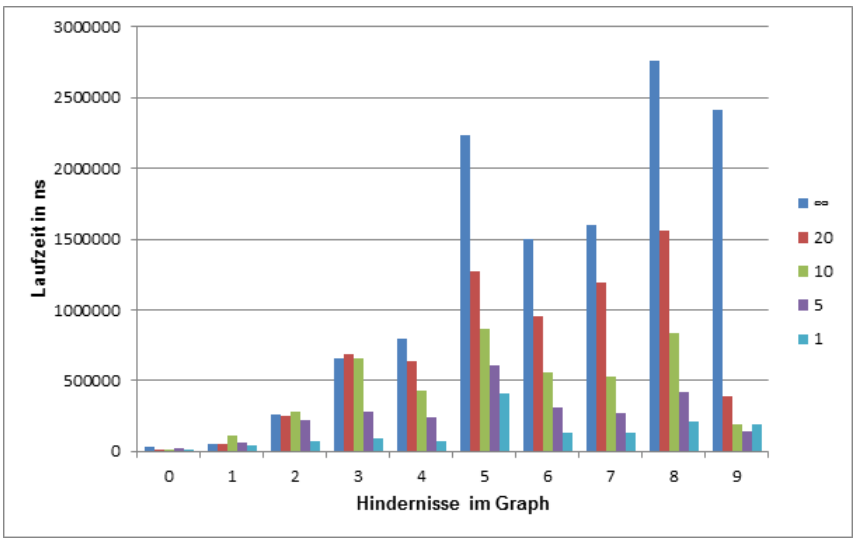
Abb. 3.15: Programmlaufzeit mit und ohne Tangentenfilterung



(a) Minimale Laufzeit



(b) Durchschnittliche Laufzeit



(c) Maximale Laufzeit

Abb. 3.16: Laufzeit des A*-Algorithmus mit unterschiedlicher Iterationsanzahl

Wird die Laufzeit zuungunsten der Qualität des gefundenen Pfades verkürzt, muss zusätzlich noch eine Betrachtung der Pfadqualität stattfinden. Dafür wird angenommen, dass die Bewegungsrichtung des Roboters optimal ist, wenn sie mit einer uneingeschränkten Iterationsanzahl des A* Algorithmus ermittelt wurde. Wird nur ein Teil des Pfades berechnet, wird die daraus berechnete Bewegungsrichtung der optimalen Bewegungsrichtung gegenübergestellt. Dies ermöglicht eine Aussage darüber zu treffen, wieviel Prozent der Fälle der nun gefundene Pfad nicht mehr dem optimalen Pfad entsprechen. Diese Werte können der folgenden Tabelle entnommen werden:

Iterationen	∞	20	10	5	1
Optimale Pfad wurde ermittelt in %	100	98,65	96,84	94,66	88,66

Tab. 3.2: Optimalität der Bewegungsrichtung in Abhängigkeit zur Anzahl der Iterationen des A*-Algorithmus

Bemerkenswert an diesen Kennzahlen ist, dass schon mit einer sehr geringen Anzahl an Iterationen überaus gute Ergebnisse erzielbar sind. Werden die Daten aus der Tabelle 3.2 in einen Kontext mit den Abbildungen 3.16 interpretiert, so ist festzustellen, dass die Restriktion der Anzahl an Iterationen sich als äußerst qualifiziertes Merkmal erweist. Vor allem die durchschnittliche und maximale Laufzeit lässt sich durch diese Maßnahme gut einschränken. Mit einer Wahrscheinlichkeit von beinah 97% die optimale Bewegungsrichtung zu ermitteln und einer damit gekoppelten drastischen Reduzierung der maximalen Laufzeit erscheint einer obere Grenze von 10 als geeignete Wahl. Wünschenswert wäre an dieser Stelle ein adaptives Verfahren, welches abhängig von der zur Verfügung stehenden Rechenzeit die Qualität der anzunehmenden Bewegungsrichtung iterativ verbessert.

4 Ergebnisse

Das Ziel der vorliegenden Arbeit war die Entwicklung eines auf Java basierten Pfadplanungs-Prototyps, welcher teilnehmende Spieler zukünftig in die Lage versetzen soll, kollisionsfreie Pfade zu planen. Hierbei stellt eine repräsentative Demo-Applikation unter Beweis, dass dieses selbst gesetzte Ziel erreicht werden konnte (siehe Abbildung 4.1).

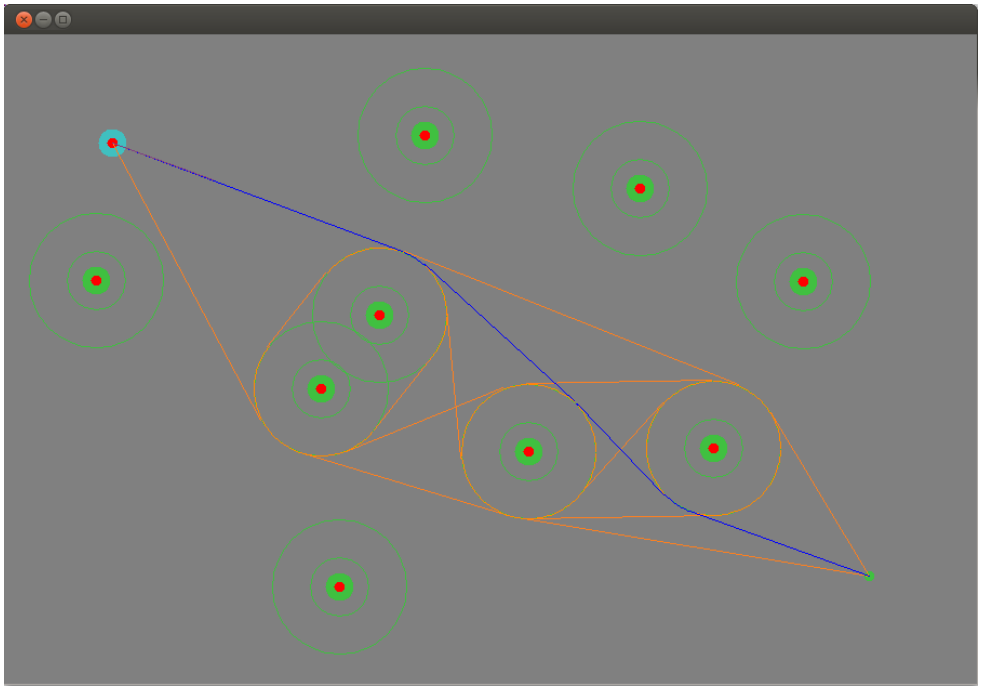


Abb. 4.1: Screenshot der 2D-Simulation. Hindernisse können dynamisch platziert, verschoben und entfernt werden.

Die Eingrenzung der Aufgabenstellung auf eine beschränkte Simulationsumgebung und der Implementierung funktionstüchtiger, aber noch nicht optimierter Softwaremodule, eignet sich im besonderen Maße um Vor- und Nachteile diverser Erweiterungen und Modifikationen mit geringem Aufwand illustrieren und evaluieren zu können.

Gewisse Charakteristika des implementierten Pfadplaners können direkt aus dem verwendeten Verfahren abgeleitet werden. So generiert die Applikation ausschließlich glatte Pfade, die einen konstanten Sicherheitsabstand zu allen Hindernissen wahren. Wegen des Sicherheitsabstands kann das Verfahren jedoch nicht als vollständig klassifiziert

werden. Somit kann es unter Umständen dazu kommen, dass eigentlich passierbare Pfade einem Spieler als unzugänglich erscheinen. In diesem Fall sieht sich der Spieler entweder dazu gezwungen einen Umweg in Kauf zu nehmen oder er ist nicht in der Lage einen validen Pfad zu planen (siehe Abbildung 4.2).

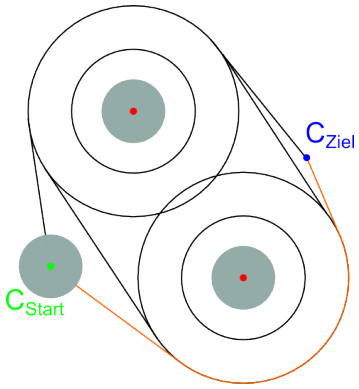


Abb. 4.2: Eine valider Pfad (orangefarben gekennzeichnet), der nicht der optimalen Route entspricht

Im Laufe der Implementierungs- und Testphase konnte die dysfunktionale Pfadplanung innerhalb des Hindernisraums als relevante Problemdomäne identifiziert werden, die einer genaueren Diskussion benötigte. Erst nach einer umfangreichen Analyse der Problemstellung konnten hierfür drei unterschiedliche Lösungsansätze entwickelt werden. Als zufriedenstellend stellte sich dabei die Reduktion des Kreisradius mit modifizierter Berechnung des Bewegungsvektors heraus. Ein Schwerpunkt der Entwicklung lag zudem auf laufzeiteffizientem Verhalten. Um hier die Qualität zu verbessern, wurden unterschiedliche Maßnahmen vorgestellt und deren Eignung in mehreren Testreihen evaluiert.

4.1 Ausblick

Die neuesten Entwicklungen im RoboCup haben deutlich gemacht, dass für das erfolgreiche Bestehen in der Standard Plattform Liga allem voran eine flexiblere Spielweise, verknüpft mit einer ausgefeilten Teamstrategie, zunehmend entscheidend ist. Die vorliegende Arbeit stellt einen konstruktiven Anteil dieser Entwicklung dar. Dessen ungeachtet stehen noch eine Vielzahl weiterer Nachforschungen in diesen Bereichen an. Im Hinblick auf die Pfadplanung bedeutet dies eine Analyse und Verbesserung des Worst Case Laufzeitverhaltens, die durchgehende Optimierung des Codes und eine Portierung der Software auf die Nao-Plattform.

Aktuell besteht noch weitrechendes Potential zur Laufzeitoptimierung. Allem voran bietet die Auswahl geeigneter Datenstrukturen aufgrund der Vielzahl an Mengenoperatio-

nen noch reichliche Möglichkeiten. Zum gegenwärtigen Zeitpunkt sind im Programmcode fast ausschließlich verkettete Listen vorzufinden. Dabei ist in solch einer Datenstruktur die Suche besonders langsam. Eine kritische Betrachtung dieser Gegebenheit steht derzeit noch aus, soll aber Teil zukünftiger Arbeiten sein. Ein Beispiel für die Möglichkeiten zur Optimierung kann am A*-Algorithmus dargelegt werden. Wird hier für die Open- und Closed-Menge ein Fibonacci-Heap verwendet, kann nach [JL] die Laufzeit von $\mathcal{O}(|K| * |K|)$ auf $\mathcal{O}(|K| * \log |K|)$ verringert werden. Darüber hinaus kann ggf. die Umstellung von doppelter Genauigkeit der geometrischen Berechnungen auf einfache Genauigkeit einen Geschwindigkeitsvorteil bieten. Dieser Faktor muss jedoch für jede Plattform gesondert evaluiert werden, da auch Hardwarekonfigurationen existieren, auf denen es faktisch keinen Unterschied macht oder sogar die Berechnung mit doppelter Genauigkeit schneller ist, als die mit einfacher Genauigkeit.

Ferner kann der Bedarf gewinnbringender Erweiterungen entstehen. Sollte diese Verbesserung jedoch nicht der Optimierung der Laufzeit dienen, kann dies eine zusätzliche Reservierung begrenzter Zeitressourcen bedeuten. Ob die Einbettung zusätzlicher Softwaremodule gerechtfertigt ist, muss dann im Einzelnen entschieden werden. Zukünftig ist es gewünscht, dass Graphen und Routen im eigens entwickelten Analyse- und Debugging-Tool „Nao-Control" angezeigt werden. Realisierbar wäre zudem ein Austausch von Informationen zwischen kooperierenden Spielern, um einer gegenseitige Behinderung zu entgehen. Eine präzise Abschätzung für die aufgewendete Zeit zum Passieren eines Pfades könnte sich ebenfalls als förderlich erweisen, um die beschränkte Aktionsauswahl besser gewichten zu können. Für diesen Zweck sind Zeitmessungen für gelaufene Wege nötig, um ein statistisches Modell zu erheben.

Bereits für die nächsten Wettkämpfe stehen für das Nao-Team HTWK Leipzig vielfältige Neuerungen an. So soll zeitnah eine Portierung der in Java vorliegenden Multi-Target Tracking Software sowie eines neuen Team Strategie Moduls erfolgen. Die Fusion der drei Softwareprototypen auf der Nao-Plattform lanciert eine neue Qualität im Spielverhalten und stellen eventuell schon die entscheidenden Verbesserungen im Wettstreit zum nächsten Titel dar.

Literaturverzeichnis

[A22hr] *A*-Algorithmus.* http://de.wikipedia.org/wiki/A*-Algorithmus. Version: 22. Februar 2014 - 16:59 Uhr

[ATFhr] *About Thunderbots FC.* http://ubcthunderbots.ca/thunderbots-ssl-team/. Version: 03. Februar 2014 - 15:04 Uhr

[CB-hra] *CPU Benchmarks - Intel Core i7-3840QM @ 2.80GHz.* http://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i7-3840QM+%40+2.80GHz&id=900. Version: 14. Februar 2014 - 18:27 Uhr

[CB-hrb] *CPU Benchmarks - Intel Atom Z530 @ 1.60GHz.* http://www.cpubenchmark.net/cpu.php?cpu=Intel+Atom+Z530+%40+1.60GHz&id=626. Version: 14. Februar 2014 - 18:33 Uhr

[Die12] Diemke, Johannes: *Pfadplanung mit harmonischen Potentialfeldern*, Diplomarbeit, 2012

[Dij59] Dijkstra, Edsger W.: *Numerische Mathematik. 1.* 1959

[Her12] Hertzberg, Joachim: *Mobile Roboter: Eine Einführung aus Sicht der Informatik.* 2012

[HNR68] Hart, Peter E. ; Nilsson, Nils J. ; Raphael, Bertram: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. (1968)

[JL] Jena, Sanjay ; Liebelt, Nils: Heuristische Algorithmen am Beispiel des A*-Algorithmus / 8-Puzzle.

[Kor91] Koren, Y.: *Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation.* 1991

[Lat91] Latombe, Jean-Claude: *Robot Motion Planning.* 1991

[Leshr] Lester, Patrick: *Using Binary Heaps in A* Pathfinding.* http://www.policyalmanac.org/games/binaryHeaps.htm. Version: 17. Februar 2014 - 00:50 Uhr

[MTUhr] *MSL-Finale: Tech United (Niederlande) vs. Water (China).* http://www.ais.uni-bonn.de/robocup.de/bilder_videos.html. Version: 03. Februar 2014 - 14:55 Uhr

[Nil69] Nilsson, Nils J.: *A Mobile Automaton: An Application of Artificial Intelligence Techniques*. 1969

[Our04] Ourima, Lina: Fast Geometric Path Planning for the Smallsize Robocup League. (2004)

[Pathr] Patel, Amit: *Pathfinding*. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. Version: 03. Februar 2014 - 13:12 Uhr

[Pea84] Pearl, Judea: *Heuristics*. 1984

[PKRhr] *Projektgruppe Kognitive Robotik BA-INF 051 (B) [B]*. http://www.ais.uni-bonn.de/SS10/PG_KR.html. Version: 03. Februar 2014 - 16:45 Uhr

[R20hr] *RoboCup 2013*. http://designyoutrust.com/2013/07/robocup-2013/. Version: 03. Februar 2014 - 15:19 Uhr

[R2Shr] *RoboCup 2013 Soccer Simulation 2D Final*. http://www.youtube.com/watch?v=BoWoIc4IrtI. Version: 03. Februar 2014 - 14:44 Uhr

[R2Thr] *RoboCup 2013 Teen Size FINAL: GERMANY / JAPAN*. http://www.youtube.com/watch?v=tdHOGCh5yYs. Version: 03. Februar 2014 - 14:56 Uhr

[RI2hr] *RoboCup IranOpen 2013 3D Simulation Final, UT Austin Villa vs Apollo3D - 1st Half*. http://www.youtube.com/watch?v=gmCLqEYmK9E. Version: 03. Februar 2014 - 14:49 Uhr

[Rob13] RoboCup Technical Committee: *RoboCup Standard Platform League (NAO) Rule Book*. 2013

[TBF05] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. 2005

[Top99] Topor, Augustinus: *Roboterfußball: Pfadplanung in dynamischer Umgebung*, Diplomarbeit, 1999

[Udu77] Udupa, Shriram M.: *Collision Detection and Avoidance in Computer Controlled Manipulators*, Diplomarbeit, 1977

[VRrhr] *VT RoMeLa robots make semifinals at RoboCup 2010*. http://www.unmanned.vt.edu/news/robocup2010.html. Version: 03. Februar 2014 - 14:58 Uhr

[Wei98] Weigel, Thilo: *Roboter-Fußball: Selbstlokalisierung, Weltmodellierung, Pfadplanung und verhaltensbasierte Kontrolle*, Diplomarbeit, 1998