



**hochschule mannheim**

# **Analyse eines Algorithmus für planendes Laufverhalten und Integration eines 3D-Sensors**

Daniel Groß

Bachelor-Thesis  
zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)  
Studiengang Mechatronik

Fakultät für Informatik  
Hochschule Mannheim

2019-04-20

Betreuer  
Prof. Dr. Thomas Ihme, Hochschule Mannheim  
Dipl.-Phys. Ute Ihme, Hochschule Mannheim

**Groß, Daniel:**

Analyse eines Algorithmus für planendes Laufverhalten und Integration eines 3D-Sensors /  
Daniel Groß. –

Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2019. ix Seiten.

**Groß, Daniel:**

Analysis of a gait- and motion-planner algorithm and integration of a 3D-Sensor / Daniel  
Groß. –

Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2019. ix pages.

## **Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 2019-04-20

Daniel Groß



# Abstract

## *Analyse eines Algorithmus für planendes Laufverhalten und Integration eines 3D-Sensors*

Das Institut für Robotik der Hochschule Mannheim besitzt diverse Laufroboter. Darunter befinden sich der Lauron III, Lauron IVa und der jüngere Akrobat. Beide besitzen sechs Beine und sind der Indischen Stabschrecke nachempfunden. Für den Akrobat wurden bereits statische Laufmuster entwickelt. Für den Lauron IVa wurde im Jahre 2003 von A. Herms ein Laufplaner entwickelt. Im Jahr 2006 modifizierte U. Ruffler den Laufplaner so, dass er inkrementell auf Basis der durch Sensoren erfassten Umgebungsinformationen arbeiten könnte. Durch sein Arbeit wurde der Algorithmus für den Echtzeit-Erkundungsbetrieb vorbereitet. Allerdings wurde der Algorithmus in einer Simulationsumgebung entwickelt, aus der er sich nicht ohne Weiteres extrahieren lässt. Ziel dieser Arbeit ist es den Quellcode beider Versionen einer vergleichenden Analyse zu unterziehen, die Unterschiede hervorzuheben und die Funktionsweise verständlich, zu erläutern. Hintergrund des Ganzen ist die Extraktion des Algorithmus, um diesen auf einem aktuellen Linux lauffähig zu machen, so dass er auch mit ROS betrieben werden kann. Der Laufplaner soll in Zukunft bei dem Roboter Akrobat implementiert werden. Ein weiteres Ziel ist die Integration eines 3D-Sensors um dem Laufplaner echte Sensordaten zur Verfügung stellen zu können.

## *Analysis of a gait- and motion-planner algorithm and integration of a 3D-Sensor*

The University of Applied Sciences Mannheim is in possession of some different six-legged walking-robots. For example the Lauron and the Akrobat are two of them. In the year 2003 A. Herms developed a gait-planner for the Lauron IVa in a simulation environment. Sixteen years elapsen since this time. Today the Akrobat is the more interesting robot. There were many possible projects to develop for the Akrobat. But the gait-planner is a very importand work and the algorithm of it should be extracted from the Lauron robot to use the algorithm for other robots. After this procedure maybe it's possible to integrate the gait-planner in the source code of the Akrobat robot. But there were still some differences between both robots. For example the Lauron uses a other framework for internal communication as the Akrobat which uses the more actually ROS framework. However the first step

---

in this direction is a clean analysis of the gait-planner source code. The first version from A. Herms is not so convenient for realtime-exploration but this is the long-time goal. Ruffler continues the development in 2006. His version is the second one and more compatible for realtime applications. The main objective in this work is to analyse both versions of the gait-planner from both developers. One more goal is the integration of a 3D-ToF-Sensor which gives Akrobat the ability to capture real informations about environment to make the planner-algorithm work in real world applications for future time.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Aufbau des Inhalts . . . . .	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 Anwendungen für Laufroboter . . . . .	6
2.2 Bionik . . . . .	6
2.3 Verwandte Arbeiten . . . . .	8
2.4 Die Stabschrecke als Inspiration . . . . .	9
2.4.1 Steuerung in der Natur . . . . .	10
2.4.2 Grundformen des Körpers . . . . .	15
2.5 Laufverfahren von Robotern . . . . .	15
2.5.1 Statische Verfahren . . . . .	15
2.5.2 Reaktive Laufverfahren . . . . .	16
2.5.3 Hybride Verfahren . . . . .	16
2.5.4 Planende Verfahren . . . . .	16
2.5.5 Neuronale Verfahren . . . . .	17
2.6 3D-Sensorik . . . . .	17
2.6.1 3D-Laser-Distanzmessgeräte . . . . .	18
2.6.2 3D-Vision-Kameras . . . . .	19
2.6.3 Time-of-Flight-Kameras . . . . .	20
<b>3 Aufbau des Laufplaners</b>	<b>23</b>
3.1 Anforderungen . . . . .	23
3.2 Übersicht der Funktionsweise . . . . .	24
3.3 Laufplanung als Optimierungsproblem . . . . .	24
3.4 Heuristiken . . . . .	26
3.5 C-Space/Konfigurationsraum . . . . .	27
3.6 Koordinatensysteme . . . . .	32
3.6.1 RKS : Roboter-Koordinaten-System (lokal) . . . . .	33
3.6.2 WKS : Welt-Koordinaten-System (global) . . . . .	34
3.6.3 Pfadfestlegung in der Version Herms . . . . .	34
3.6.4 Pfadfestlegung in der Version Ruffler . . . . .	34

<b>4 Eingabe-Informationen</b>	<b>35</b>
4.1 Start- und Zielposition . . . . .	35
4.2 Umgebungsinformationen . . . . .	36
4.2.1 Verwaltung der Daten zur Umgebung . . . . .	37
4.2.2 Darstellung des Terrains in der Simulationsumgebung . . . . .	37
4.2.3 Umgebungsinformationen aus Höhenkarte . . . . .	37
4.2.4 Umgebungsinformationen durch Stereovision-Kamera . . . . .	38
<b>5 Planung der Körperbewegung (Pfad)</b>	<b>39</b>
5.1 Festlegung des Pfades . . . . .	40
5.2 Zulässiger Bereich . . . . .	41
5.2.1 Einschränkung durch Prüfung auf Zulässigkeit . . . . .	42
5.2.2 Einschränkung durch Reichweite der Fußpunkte . . . . .	43
5.3 Einschränkung durch Pfadverlauf . . . . .	43
5.3.1 Ermittlung der Grenzen bei Eindeutigkeit . . . . .	44
5.3.2 Ermittlung der Grenzen bei Nicht-Eindeutigkeit . . . . .	45
<b>6 Planung der Schritte</b>	<b>47</b>
6.1 Bewertungsfunktion . . . . .	47
6.1.1 Kriterium 1: Bewegungsdauer . . . . .	48
6.1.2 Kriterium 2: Kippstabilität . . . . .	49
6.1.3 Kriterium 3: Untergrundstabilität . . . . .	54
6.2 Prüfung auf Zulässigkeit . . . . .	56
6.2.1 Bedingung 1 : Unzulässiger Bereich stützender Füße . . . . .	56
6.2.2 Bedingung 2: Unzulässiger Bereich für Fuß-Zielposition . . . . .	57
6.2.3 Bedingung 3: Unzulässige Bewegungsdauer . . . . .	58
6.2.4 Bedingung 4: Unzulässige Bewegung der Beine . . . . .	58
<b>7 State machine</b>	<b>61</b>
7.1 Selektion eines Fußes, der seine Position wechselt . . . . .	61
7.2 Zustands-Graph . . . . .	62
7.3 Festlegung der Folge-Zustände . . . . .	66
<b>8 Ereignisse und Listen</b>	<b>69</b>
8.1 Zusammenspiel der Ereignisse (Bewegungen) . . . . .	69
8.2 Ereignis: Fußbewegung . . . . .	70
8.2.1 Aufsetzen eines Fußes . . . . .	71
8.2.2 Anheben eines Fußes (unzulässig) . . . . .	72
8.3 Ereignis: Mittelpunktbewegung . . . . .	74
8.3.1 Mittelpunkt-Verschiebung beim Aufsetzen eines Fußes . . . . .	75
8.3.2 Mittelpunkt-Verschiebung beim Anheben eines Fußes . . . . .	75
8.4 Listen . . . . .	79
8.4.1 Die fList . . . . .	80
8.4.2 Die cList . . . . .	82

8.5 Ein Move wird zum Movement . . . . .	82
8.5.1 Movement-Erzeugung in der Version Herms-2003 . . . . .	84
8.5.2 Movement-Erzeugung in Version Ruffler-2006 . . . . .	85
8.6 Klassen . . . . .	88
8.6.1 Herms-Version . . . . .	88
8.6.2 Ruffler-Version . . . . .	89
<b>9 Verfahrensvorschlag zur Umgebungserkundung in Echtzeit</b>	<b>91</b>
9.1 Laufplaner-Vorbereitungen durch Uli Ruffler . . . . .	92
9.2 Ähnliche Verfahren . . . . .	92
9.3 Vorgeschlagenes Verfahren zur Echtzeit-Erkundung . . . . .	93
9.4 ROS-Kompatibilität . . . . .	95
<b>10 System des Akrobot-Roboters</b>	<b>97</b>
10.1 Mechanik . . . . .	97
10.2 Elektronik . . . . .	97
10.3 Betriebssysteme . . . . .	98
<b>11 Akrobot Hardware-Upgrade</b>	<b>101</b>
11.1 System zur 3D-Bild-Erfassung/-Verarbeitung . . . . .	101
11.1.1 3D-Sensor . . . . .	101
11.1.2 Recheneinheit: Intel UP-Board . . . . .	102
11.1.3 Ethernet-Anbindung . . . . .	104
11.1.4 Verwenden des 3D-Sensors . . . . .	104
<b>12 Zusammenfassung und Weiterentwicklungen</b>	<b>109</b>
12.1 Zusammenfassung . . . . .	109
12.2 Weiterentwicklungen . . . . .	110
<b>Tabellenverzeichnis</b>	<b>xi</b>
<b>Abbildungsverzeichnis</b>	<b>xiii</b>
<b>Quellcodeverzeichnis</b>	<b>xv</b>
<b>Literatur</b>	<b>xv</b>



# **Danksagung**

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Anfertigung dieser Bachelor-Arbeit unterstützt und motiviert haben.

Mein besonderer Dank gilt Herrn Prof. Dr. Thomas Ihme für die fachliche Kompetenz und die hilfreiche Betreuung während der Anfertigung dieser Arbeit.

Besonders bei Moritz Thomas möchte ich mich dafür bedanken, das er mir die erste Laufplaner-Version inklusive Virtueller-Maschine (VM) zur Verfügung gestellt hat.

Ganz besonders bedanken möchte ich mich bei meiner Familie für die Unterstützung und den Rückhalt während des gesamten Studiums, bei meiner Freundin Jennifer Klotz für die Motivation dieses Studium zu beginnen und bei meinen Freunden Dirk Schreckenberger für die hilfreiche Beratung.

x

# Kapitel 1

## Einleitung

### 1.1 Motivation

Der sechsbeinige Laufroboter Akrobat der Hochschule Mannheim wurde bereits 2014 von Wilen Askerow mit statischen Laufmustern, wie dem Dreifuß- oder Wellengang ausgestattet [3]. Dabei wurde für die interne Kommunikation zwischen Sensoren, Aktoren und den Steuereinheiten wurde über ROS<sup>1</sup> (Version indigo) realisiert.

Ein anderer sechsbeiniger Roboter, der Hochschule Mannheim ist der Laufroboter Lauron in Version III (Abkürzung für LAUF-ROboter-Neuronal gesteuert). Dieser ist bedeutend größer als der Akrobat, folglich auch bedeutend schwerer. Lauron ist schon seit längerer Zeit mit diversen Weiterentwicklungen ausgerüstet und so gesehen „gesättigt“. Der Akrobat ist noch lange nicht auf dem gleichen technischen Niveau wie der Lauron-III und -IVa. Der Akrobat ist recht schlicht konzipiert und besitzt einen kleineren Formfaktor. Versuche können an ihm mit wesentlich geringerem Aufwand durchgeführt werden. Ein weiterer Vorteil ist der Steuerungs-Betrieb über Ubuntu (Linux) und ROS, der sich im Roboter befindet. Änderungen sind direkt am Roboter möglich. Durch [14] und [23] wurde die Steuerung vollständig als Embedded-PC „on-board“ integriert. Ebenso ist der Roboter auf autarken Betrieb erweitert worden. In den beiden Arbeiten wurden weitere Verbesserungen getroffen, wie beispielsweise eine interne Hardware-Optimierung.

---

<sup>1</sup>Robot-Operating-System kurz ROS. Das System ist aufgeteilt in das eigentliche Betriebssystem ros und ros-pkg (packages), eine Auswahl an Zusatzpaketen, die das Basissystem um (meist einzelne) Fähigkeiten erweitern. Dabei wird eine Serviceorientierte Architektur nach dem „Publisher - Subscriber“-Prinzip eingesetzt, die eine Kommunikation zwischen den einzelnen Komponenten ermöglicht.

Das System verfügt mittlerweile über ein benutzerfreundliche Bedien- und Anschlussmöglichkeit für diverse Peripherie-Geräte. In dieser Arbeit wurde der Akrobat wieder ein Stück weit verbessert, indem ein weiteres System integriert wurde, dass der Steuerung (Raspberry-Pi) unterliegt. Es handelt sich um ein System zum Erfassen von 3D-Bildern, dessen einzelne Komponenten eine Gesamteinheit bilden. Es besteht im Wesentlichen aus einen ToF-Sensor (Time-Of-Flight) auch einem Embedded-PC, der allerdings eine höhere Leistung als der Raspberry-Pi besitzt. Bildverarbeitungsprozesse können sehr viel Rechenleistung beanspruchen, die somit nicht vom Raspberry-Pi aufgebracht werden muss. Auch das Spektrum möglicher Weiterentwicklungen, für Studenten, wurde dadurch breiter.

Als langfristiges Ziel wird der autonome Betrieb verfolgt. Für den LAURON-III wurde bereits ein Laufplaner auf heuristischer Basis von André Herms entwickelt [17]. Mit diesem konnten innerhalb einer Simulationsumgebung (Mit open-Inventor Bibliothek aufgebaut), bereits gute Resultate erzielt werden. Der Algorithmus des Laufplaners wurde dann 2006 nochmal von U. Ruffler weiterentwickelt und weitgehend echtzeitfähig gemacht [26]. Da an Lauron-III und -IVa keine Weiterentwicklungen mehr stattfinden, soll eine Extraktion des Algorithmus hingearbeitet werden, die diesen konserviert und auch auf weiteren sechsbeinigen Laufrobotern einsetzbar macht. Auch der Akrobat könnte davon profitieren und mit „planendem Laufverhalten“ ausgestattet werden. Für eine erfolgreiche Extraktion aus der Simulationsumgebung, muss von Grund auf die Funktionsweise verständlich gemacht werden. Mit diesem Hintergrund wurde im Rahmen dieser Arbeit eine Quellcode-Analyse des Laufplaners durchgeführt. Der Laufplaner kann somit als eine wertvolles Programm zur Steuerung von hexapod-Laufrobotern angesehen werden, dessen Funktion bereits erprobt worden ist.

### 1.2 Zielsetzung

Ziel dieser Arbeit ist es, den Roboter Akrobat für die Integration des Laufplaner-Algorithmus vorzubereiten. Der Laufplaner selbst wurde für den Roboter Lauron entwickelt. Zur Vorbereitung gehört einer vergleichende Analyse beider Versionen des Laufplaner-Algorithmus (Herms 2003 und Ruffler 2006). Die Analyse soll hierbei die vollständige Funktionsweise des Algorithmus beschreiben inklusive der Erweiterungen/Änderungen durch U. Ruffler [26]. Hierfür ist es notwendig beide

Laufplaner Versionen auf einem aktuellen Linux kompilierbar und lauffähig zu machen. Hintergrund des Ganzen ist die Konservierung des Laufplaner-Algorithmus in einer funktionsfähigen Art und Weise, um diesen für weitere Arbeiten und Versuchszwecke verfügbar zu machen. Mit diesen soll auf die Implementierung beim Roboter Akrobat hingearbeitet werden. In der aktuellsten Version verfügt der Laufplaner über einen Anytime-Algorithmus der auf Basis des Stereo-Vision Verfahrens arbeitet. Die benötigten Sensordaten wurden in der Simulationsumgebung generiert. Der Roboter Akrobat soll in Zukunft mit Laufplaner und echten Sensordaten betrieben werden. Um diesem Ziel näher zu kommen, wird bei dem Roboter ein 3D-Sensor integriert. Anders als bei Lauron, arbeitet der Akrobat mit dem Robot Operating System (ROS), darum werden alle Vorbereitungsmaßnahmen, so weit es möglich ist, auf die Verwendung mit ROS ausgerichtet. Um die Auswertung der Sensordaten zu ermöglichen, ohne dabei die Rechenressourcen des Raspberry-Pi zu belasten, wird ein weiteres Embedded-PC Board im System integriert. Dieses ermöglicht, zusammen mit dem Robot Operating System, den Datenverkehr zwischen Sensor und Gesamtsystem.

### 1.3 Aufbau des Inhalts

Der einleitende Teil behandelt Grundlegendes zur Bionik und zeigt einige Beispiele, von Robotern, aus diesem Bereich. Darauf folgend werden verwandte Arbeiten vorgestellt und das Grundwissen zum Laufverhalten, in der Natur, erläutert. Der Focus liegt dabei auf dem Laufverhalten den Stabschrecken, die als Vorbild für den Akrobat und Lauron Roboter dienen. Der abschließende Teil der Einleitung führt die unterschiedlichen Verfahren, die es im Bereich des Laufverhaltens von hexapod-Laufrobotern gibt, auf.

Der Hauptteil-A beginnt mit dem Aufbau des analysierten Laufplaners. Die Analyse dessen stellt gleichzeitig das Hauptziel dieser Arbeit dar. Der Analyseteil (Laufplaner-Teil) dieser Arbeit wurde so angeordnet, dass er möglichst dem Programmablauf, des Laufplaners selbst, gleicht. Zuvor wird jedoch der Aufbau des Laufplaners beschrieben, der mit Kapitel drei beginnt. Nach dem Aufbau wird ausführlich auf die vom Planer benötigte Eingabe eingegangen, sowie auf Bereiche in die in Verbindung damit stehen, wie zum Beispiel dem C-Space oder den Koordinatensystemen. Das darauf folgende Kapitel befasst sich mit dem Anlegen/Planen des Pfades, dem

der Körper folgen soll. An diese Stelle ist zu bemerken, dass in der Dokumentation von Herms zuerst die Schritt-Planung behandelt wird und anschließend die globale Bewegung des Körpers. In diese Arbeit wurde die Reihenfolge der beiden gedreht, da der Laufplanungs-Algorithmus auch zuerst einen Pfad festlegt, bevor die Schrittplanung erfolgen kann. Aus diesem Grund folgt nach dem Kapitel „Eingabegrößen“ zuerst das Kapitel „Pfad-Planung“ und darauf folgend das Kapitel „Schritt-Planung“. Der Inhalt der zwei Planungs-Kapitel ist notwendig, um den Inhalt des Kapitels „State Machine“, das darauf folgt, zu verstehen. Das Kapitel „State Machine“ verknüpft die Inhalte der beiden Kapitel davor. In dieser Art verläuft auch der interne Ablauf, des Algorithmus. Die anschließenden zwei Kapitel behandeln Ereignisse (Bewegungen), Movements ebenso deren Verkettung, ab der Ruffler-Version und das Zeitverhalten im Detail. Die Schleife endet mit Kapitel acht, der Movement-Erzeugung und beginnt wieder bei Kapitel vier, der Pfad-Planung.

Im Anschluss wird der aktuelle Arbeitsstand des Akrobat-Roboters im Bereich Elektronik, Mechanik und der Betriebssysteme als Tabelle aufgezeigt. Das elfte und letzte Kapitel befasst sich mit der Hardware-Erweiterung/Integration. Diese beschränkt sich auf die Integration der 3D-Bildverarbeitungs-Einheit, sowie deren ROS-Anbindung. Neben dem 3D-Sensor (PMD PicoFlex) beinhaltet diese ein weiteres Embedded-PC-Board (Intel UP-Board) und eine individuell angefertigte Frontabdeckung. Eine Zusammenfassung gefolgt von möglichen Weiterentwicklungen bildet den Schluss der Arbeit.

Als Anhang wird die, von Moritz Thomas erstellte Virtual-Machine (Debian-build-machine) und die beiden Laufplaner-Versionen, als .tar.gz Archive, mitgeliefert.

## Kapitel 2

### Grundlagen

Etwa die halbe Erdoberfläche ist mittlerweile so geformt, dass die Menschen diese mit bereiften Fahrzeugen nutzen können. Fahren auf Rollen oder Rädern ist eine kostengünstige und effiziente Art sich fortzubewegen. Es existieren dennoch viele Gebiete, die noch naturbelassen und nicht dem „Erdboden gleich“ gemacht wurden. In solchen Bereichen stößt das Rad an seine Grenzen.

Es muss eine alternative Bewegungsart verwendet werden, um auch in diesen Zonen eine maschinelle Fortbewegung anwenden zu können. Der Großteil der Menschheit hat wohl kaum Interesse daran, um sich über Felsen, Baumstämme und ähnliche Hindernisse maschinell fortzubewegen. Für solche zerklüfteten Untergründe ist die menschliche Lokomotion nicht gerade günstig. Mit einer Gangart wie sie Insekten besitzen, wären die Verhältnisse schon wesentlich besser.

Einer der wesentlichen Vorteile gegenüber einer Translation mit Rädern ist die Tatsache, dass kein Umdrehen oder Wenden nötig ist. Der Roboterkörper kann sich überall hinbewegen, ohne seine Ausrichtung ändern zu müssen. Ein Gefährt auf Rädern besitzt durchgehend Bodenkontakt und kann nicht über Hindernisse hinwegsteigen, sondern diese bestenfalls umfahren. Des Weiteren ist ein Fahrzeug mit Rädern auch immer sehr stark vom Untergrund abhängig. Hinsichtlich der Geschwindigkeit schneidet das rollende Gefährt ganz klar besser ab. Laufroboter sind langsamer und besitzen eine sehr komplexe Steuerung im Vergleich zu einem Auto, das nur fahren und in eine Richtung gelenkt werden muss.



**Abbildung 2.1:** Roboterhund Cheetah3 verfügt nur über taktile Wahrnehmung [5]

### 2.1 Anwendungen für Laufroboter

Einige Gruppierungen sind aber doch daran interessiert sich über solche riskanten Bereiche zu bewegen und dies teilweise sogar noch mit schwerer Last. Hierzu gehört bspw. das Militär oder Forstbetriebe. Mit der Zeit entwickelten sich diverse Ansätze, oft mit bionischer Vorlage, um derartiges Gelände zu beschreiten. Der Überbegriff Lauf- oder Schreitroboter wird für alle Fortbewegungsmaschinen verwendet, deren Lokomotion mit Hilfe von Beinen stattfindet. Oft sind die Vorbilder für solche Konzepte natürlichen Ursprungs und wurden von Tieren kopiert.

In den Bereichen, in denen der Mensch mit seinem bereiften Rollgefäß nicht weiter kommt, kann die Lösung ein Laufroboter sein.

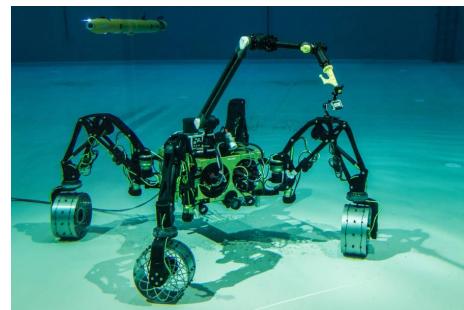
### 2.2 Bionik

In diesem Abschnitt werden einige Beispiele mit biologischem Vorbild gezeigt und auf deren Analogien in der Technik eingegangen.

Der Roboter „Cheetah 3“ ist eine Entwicklung des MIT. Sein Vorbild aus der Natur ist nicht wie der Name vermuten lässt ein Gepard, sondern vielmehr ein Hund. Technisch gesehen ist der Hunderoboter blind. Er soll sich ausschließlich über seinen Tastsinn fortbewegen und orientieren. Genau wie bei blinden Menschen wurde bei dem Roboterhund bewusst die optische Erfassung des Umfeldes weggelassen. „Blind Locomotion“ ist hier das Stichwort. Zu deutsch „blinde Fortbewegung“ wur-



**Abbildung 2.2:** Chimpansenroboter Charlie, kann auf zwei und vier Füßen laufen [19]



**Abbildung 2.3:** Sherpa-UW beim Testeinsatz unter Wasser [19]

de hier eingesetzt um auf visuelle Informationsgewinnung zu verzichten, denn diese ist oft durch Störungen wie Bildrauschen etc. entsprechend schlecht auszuwerten. Im Wesentlichen besteht die Steuersoftware aus zwei Algorithmen. Der erste ist der Kontakterkennungsalgorithmus, der die ertasteten Informationen auswertet. Bei der geringsten Berührung eines Gegenstandes wird in einer Zeitspanne von 50 ms das weitere Vorgehen bzw. die passende Körperbewegung berechnet, um die Bewegung fortzusetzen. Der zweite Teil ist für die Berechnung der benötigten Kräfte zuständig. Es handelt sich somit um einen Laufroboter, der alleine auf Grund von taktilen Informationen arbeitet.

Ein weiteres Beispiel der Bionik ist der Roboteraffe „Charlie“, der im Bild 2.2 zu sehen ist. Wie bei einem echten Schimpanse kann auch diese Roboter-Version wahlfweise eine zweifüßige oder vierfüßige Lokomotion zur Fortbewegung verwenden.

Die beiden Roboter auf den Abbildungen 2.2 und 2.3 sind Entwicklungen des DZ-KI, dem Deutschen Zentrum für künstliche Intelligenz. Abbildung 2.4 zeigt ein exotischeres Beispiel eines Roboters, den „Schlangenroboter“ KAIRO-3 des FZI-Karlsruhe. Roboterschlanze KAIRO-3 besteht aus sechs Antriebsmodulen und fünf dazwischenliegenden Gelenkmodulen. Gewicht und Länge eines einzelnen Antriebsmoduls betragen 4,5 kg und 146 mm. Bemerkenswert ist hierbei die daraus folgende große Anzahl von Freiheitsgraden (27 DOF<sup>1</sup>), daher zählt es zu den sogenannten hyperredundanten Robotersystemen.

---

<sup>1</sup>Degrees-Of-Freedom - DOF = Freiheitgrade



**Abbildung 2.4:** Schlangenroboter Kairo-3 des FZI-Karlsruhe



**Abbildung 2.5:** Laufroboter Lauron-5 des FZI-Karlsruhe

### 2.3 Verwandte Arbeiten

Auf Abbildung 2.5 ist der sechsbeinige Laufroboter LAURON-IVa (LAUFender ROboter Neuronal gesteuert) zu sehen, der bis zu einer Neigung von  $45^\circ$  und bei Bewegung bis zu  $25^\circ$ , statisch stabil ist. Aktuell konzentriert man sich bei LAURON auf die Steigerung der Autonomie und das Lösen von komplexen Navigationsaufgaben wie zum Beispiel die Kartierung der Umgebung. Weitere Forschungsinteressen liegen bei der Manipulation mittels Vorderbeinen und energieeffizientem sowie sicherem Laufen [12].

Ein weiteres und etwas größeres Exemplar eines sechsbeinigen Laufroboters, zeigt Abbildung 2.6. Hierauf sieht man Comet-IV, der vollständig autonom betrieben wird, [24]. Der Roboter wurde für die Anwendung in der Landwirtschaft entworfen, deshalb arbeiten die Aktoren auf hydraulischer<sup>2</sup> Basis.

---

<sup>2</sup>Hydraulische betriebene Aktoren arbeiten mit Öldruck und werden dort eingesetzt, wo große Kräfte benötigt werden.



Abbildung 2.6: Sechsbeiniger autonomer und hydraulisch betriebener Laufroboter Comet IV [24]

## 2.4 Die Stabschrecke als Inspiration

Die Roboter Lauron-III, -IVa und Akrobat sind nach dem biologischen Vorbild der Stabschrecke konzipiert. Die Stabschrecke gehört der Klasse der Insekten an und ist dem Stamm Gliederfüßer (Arthropoda) zugeordnet, genauer dem Unterstamm Sechsfüßer. Die Stabschrecke<sup>3</sup> (*Carausius Morosus*) zählt zu den Phasmiden bzw. den Gespenstschricken zugeordnet.

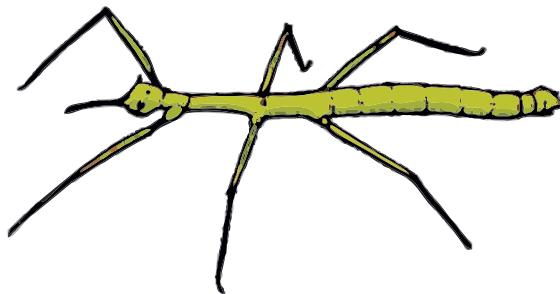


Abbildung 2.7: Stabschrecke (*Carausius Morosus*)

Jan-Ullrich Schamburek erklärt in „Bewegungssteuerung bei Insekten“ [27] wie deren Beinkoordination mit der Sinneswahrnehmung im Zusammenhang steht. Ein Kapitel ist von ihm verfasst und den Insekten gewidmet. Das Insekt mit dem meist untersuchten Verhalten ist die Stabschrecke (*Carausius Morosus*).

Abbildung 2.8 zeigt ein technisches Modell eines Insektenbeines der Stabschrecke, [27], [22]. Neue Erkenntnisse bezüglich der Lokomotion von Tieren, insbesondere mit mehreren Beinpaaren, wurde im Jahr 2018 durch eine Studie des Zoologen Dr.

<sup>3</sup>Nicht ganz korrekterweise werden die Stabschrecken oft als Stabheuschrecken bezeichnet, obwohl es keine Heuschrecken sind. Vielmehr gehören sie der Gattung der Gespenstschricken an.

Tom Weihmann vom Institut für Zoologie der Universität zu Köln belegt. Er kam zur Erkenntnis, dass die Anzahl der Beinpaare eines Lebewesens wesentlichen Einfluss auf dessen Energiehaushalt hat. Weihmann entwickelte mathematische Modelle, welche die Analyse der Bewegungsmechanik von beinantriebenen Tieren wie Insekten, Spinnen oder Tausendfüßern in Abhängigkeit der Beinpaar-Anzahl ermöglichte. Eine effiziente Fortbewegung realisieren viele Tiere über die Rückgewinnung von Bewegungsenergie. Dies ist dann möglich, wenn in elastischen Strukturen Energie zwischengespeichert und zu einem späteren Zeitpunkt wieder freigesetzt wird. Rhythmische Auf- und Ab-Bewegungen eines Körpers sind hierfür charakteristisch, wie es zum Beispiel der Fall ist, wenn ein Mensch rennt. Derartige energieeffiziente Koordinationsmuster wurden bereits bei der Entwicklung einer ganzen Reihe zweibeiniger und mehrbeiniger experimenteller Roboter genutzt. In seiner Studie zeigt Weihmann, dass mit steigender Anzahl an Beinpaaren die Energierückgewinnung zunehmend gemindert wird. Der Grund dafür ist die Notwendigkeit einer deutlich stärkeren zeitlichen Synchronisation zur Koordination der vielen Beine.

Weihmann erklärt:

„Meine Ergebnisse deuten darauf hin, dass kleine zeitliche Verschiebungen in den Beinkoordinationsmustern die Dynamik in Bewegungsapparaten von Tieren mit vielen Beinen wesentlich stärker beeinflussen als bei Lebewesen mit wenigen Beinen.“

### 2.4.1 Steuerung in der Natur

Die farbig gekennzeichneten Gelenke des Modells in Abbildung 2.8 sind auch die, die bei den Robotern Lauron und Akrobat in jedem Bein vorhanden sind. Das Insekt besitzt eigentlich sogar einen vierten Freiheitsgrad in jedem Bein. Dieser wurde bei den Robotern zur Vereinfachung weg gelassen. Es setzt diesen auch nur für spezielle Situationen ein, wie beispielsweise zum Schutz vor Angriffen. Das Nervensystem dieser Insekten ist recht einfach aufgebaut und konnte sich in der Vergangenheit gut analysieren und untersuchen lassen. Das gezeigte schematische Modell der Bewegungssteuerung in Abbildung 2.9, wurde aus beobachtetem Verhalten hergeleitet. Allgemein ist das Nervensystem in zwei Teilsysteme unterteilt, in Kopf und Ganglinienkette. Letztere verläuft auf der Unterseite entlang der Brust. Hierbei ist das Gehirn beziehungsweise der Kopfteil für die Initiierung einer Be-

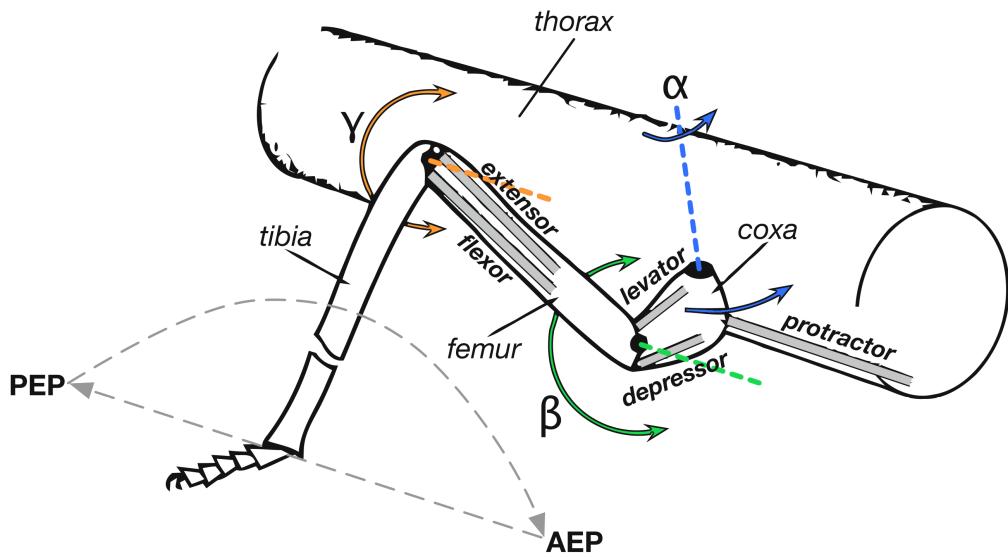


Abbildung 2.8: Abstraktion zum kinematischen Modell, [28]

Tabelle 2.1: Körperteile des Stabschreckenbeins

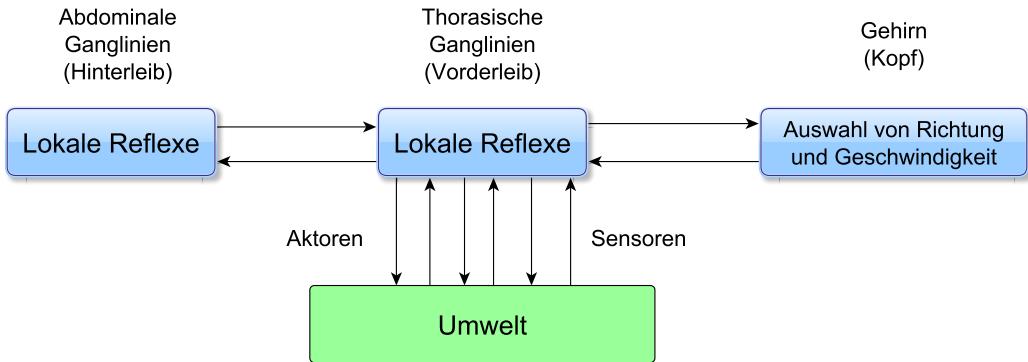
Körperteil	lateinische Bezeichnung
Unterschenkel	Tibia
Hüfte	Coxa
Oberschenkel	Femur
Fuß	-

wegung sowie deren Richtung und Geschwindigkeit zuständig. Man kann also von einem global-getaktetem Teil sprechen. Jedes Bein kann aber auch dezentral bewegt werden, ohne eine Rückmeldung an das Gehirn. Diese Tatsache gilt allerdings nicht für kopflose Insekten wie zum Beispiel Würmer, sie können nur reflexartig reagieren.

In Tabelle 2.1 sind die Körperteile, die ein Bein beinhaltet, zusammengefasst. Die Freiheitsgrade der Gelenke können aus Tabelle 2.2 entnommen werden.

Tabelle 2.2: Gelenke und Freiheitsgrade

Gelenk	Freiheitsgrad $f$
Subcoxal	2
Coxa-Trochanter	1
Femur-Tibia	1



**Abbildung 2.9:** Schema der Bewegungssteuerung

Die einzelnen Beinglieder haben jeweils eine eigene Steuerung (lokal). Die Beinpaare eines Körpersegments werden über ein sogenanntes Ganglion angesteuert. Das Ganglion steuert auch die einzelnen Gelenke der Beine. Die Beine selbst tragen auch zum zyklischen Rhythmus der gesamten Bewegungssteuerung bei.

Das zentrale Nervensystem übernimmt Bewegungsmuster wie Laufen, Kratzen, Putzen und ähnliche. Angriffs- und Verteidigungsmuster werden ebenfalls vom zentralen Nervensystem gesteuert.

Das Muster zum Ausüben des Laufens wird in den „Oszillatoren“ (siehe Abbildung 2.10) der Beine erzeugt wird. Bei langsamer bis zu normaler Laufgeschwindigkeit überwiegt der Einfluss der zurückgeführten Sensorsignale. Bei schnellem Laufen werden diese von einem anderen Rhythmus dominiert, bei dem auch eine Phasenverschiebung zwischen den Beinen auftritt.

Die lokale Einzelbeinkoordination findet über einen Schrittmustergenerator statt, der die zyklische Bewegung einzelner Beine steuert. Die Bewegungen eines einzelnen Beins werden in Schwing- und Stemmphase unterschieden. In der Schwingphase wird das Bein vom Boden abgehoben und in die anvisierte Richtung bewegt, wo es dann in die Stemmphase übergeht. In der Stemmphase stützt das Bein den Körper über dem Boden und stemmt diesen, so dass dieser in die gewählte Bewegungsrichtung geschoben wird [25].

Der Mustergenerator gibt vor, wann der Wechsel von der Stemm- zur Schwingphase erfolgt, so wie es in Abbildung 2.11 zu sehen ist. Treten Störungen innerhalb des Musters auf, so kann das Insekt seine Stabilität verlieren und umkippen. Das Umschalten wird aber auch von der Beinposition, der Beinbelastung sowie der Phase anderer Beine beeinflusst [27].

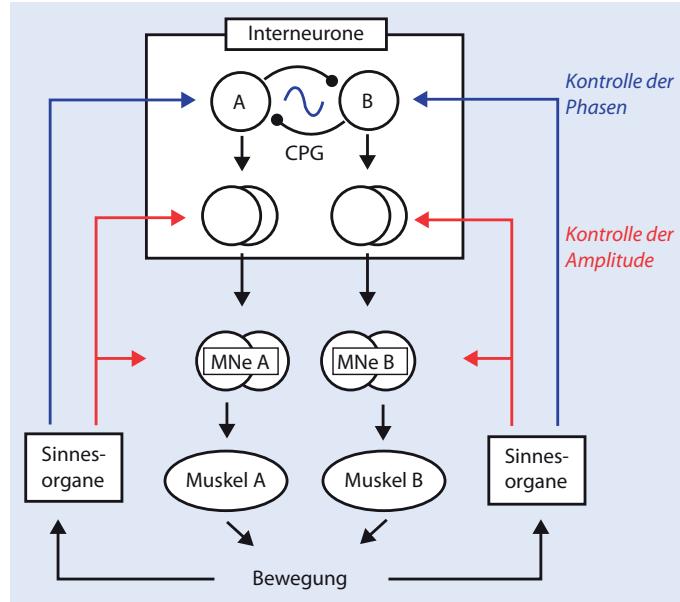


Abbildung 2.10: Schematische Darstellung der Bein-Oszillatoren, [1]

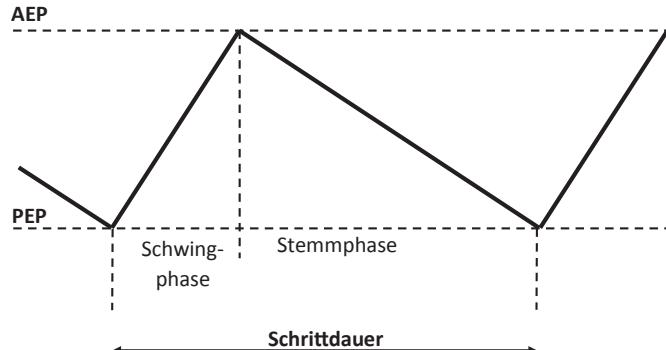


Abbildung 2.11: Merkmale des Schrittdzyklus

Das Anheben eines Beines, also der Eintritt in die Schwingphase ist von weiteren Faktoren abhängig, wie:

- Die Position des Beines muss weit genug hinten liegen (PEP<sup>4</sup>)
- Das Bein darf in diesem Moment nicht zu viel Last tragen.

Das Pendant zu PEP bezeichnet man mit „Anterior Extreme Position“ (AEP) und steht für die am weitesten vorne liegenden Position beim Schwing-/Stemm-Vorgang [22].

<sup>4</sup>Posterior Extreme Position - PEP ist die Bezeichnung für die, beim Stemm-/Schwing-Vorgang die am weitesten hinten liegende, Position.

Im Falle von Koordinationsstörungen zwischen zwei benachbarten Beinen besitzt das Insekt die folgenden Grundmechanismen (Cruse-Regeln, [9]) um diese auszugleichen:

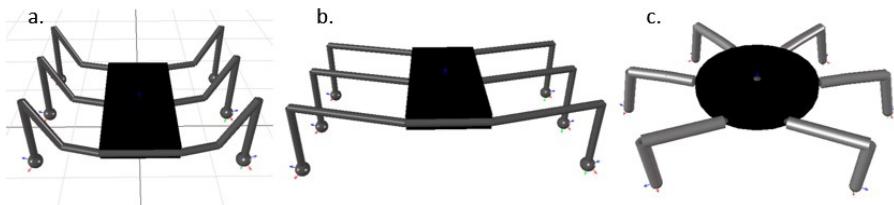
- Die Schwingphase eines Beines kann einen verzögerten Start der Schwingphase eines anderen hervorrufen.
- Der Start der Stemmphase eines Beines löst den Start der Schwingphase eines anderen aus.
- Das Erreichen der PEP eines Beines regt den Start der Stemmphase eines anderen an.

Des Weiteren besitzen die Tiere die folgenden drei reflexartigen Mechanismen:

- Die aktuelle Position kann die angezielte Position am Ende einer Schwingphase beeinflussen.
- Erhöhter Widerstand erhöht die Antriebskraft (Regelung)
- Beim einem „Auf den Fuß treten“ wird ein Reflex ausgelöst.

Das Gangmuster der Stabheuschrecke bildet eine mechatronische Welle der Schwingbeine von hinten nach vorne [6]. Bei schnellem Laufen und wenig Last wird, im ungestörten Fall, die tripode Gangart (Dreifußgang) verwendet. Hierbei wird das vorderste und hinterste Bein der einen Seite und das Mittelbein der anderen Seite synchron geschwungen. Die anderen drei Beine befinden sich derweil in der Stemmphase und stützen den Körper so, dass der Schwerpunkt etwa in der Mitte eines Stützpolygons liegt. Der Begriff Stützpolygon oder genauer noch konvexe Hülle, wird verwendet, weil je nach Gangart unterschiedliche geometrische Formen auftreten können. Eine weitere Gangart ist die tetrapode, bei welcher ständig vier Beine am Boden sind. Sie wird beim langsamen Laufen oder unter hoher Last angewendet.

Bei dieser lässt sich eine Welle von Schwingbewegungen beobachten, die von hinten nach vorne verlaufen. Aus diesem Grund wird die Gangart auch als Wellengang bezeichnet. Sobald das Insekt ein Hindernis fühlt oder visuell erkennt, stoppt es und verlagert seinen Massenschwerpunkt, indem es die Hinterbeine beugt. Anschließend positioniert es die hinteren und mittleren Beine durch kleine Schritte, sodass diese auf beiden Seiten symmetrisch zueinander liegen. Handelt es sich bei dem Hindernis zum Beispiel um einen Graben, dann werden die Vorderbeine auf



**Abbildung 2.12:** Grundformen: Arachnid(links), Reptil(mitte), Zirkular(rechts), [15]

die andere Grabenseite platziert und die beiden Mittelbeine gleichzeitig hinübergeschwungen. Die Hinterbeine folgen diesen dann in der selben Art und Weise.

#### 2.4.2 Grundformen des Körpers

Hexapode Laufroboter lassen sich grob in die drei Gruppen einteilen, siehe Abbildung 2.12. Die Unterscheidung beruht auf der Beinanordnung und der Beinhaltung.

### 2.5 Laufverfahren von Robotern

Im wesentlichen wird das Laufverhalten von Robotern in statische-, reaktive-, hybride-, planende- und neuronale-Verfahren unterschieden. Die genannten werden im Folgenden kurz erläutert.

#### 2.5.1 Statische Verfahren

Muster wiederholen sich in bestimmten Regelmäßigkeiten, dies gilt auch für Laufmuster, die zu den statischen Verfahren zählen. Gängige Laufmuster bei hexapod-Robotern sind der Dreifußgang und der Wellengang, die bereits im Abschnitt „Steuerung in der Natur“ beschrieben wurden, [25]. Ferner gibt es noch den sogenannten „Rippelgang“, bei dem immer nur ein Fuß versetzt wird. Dabei wird immer ein Fuß nach dem anderen gewählt, wobei die Reihenfolge immer die gleiche ist. Ein Nachteil dieses Musters ist, dass die resultierende Körperbewegung nur sehr langsam vonstatten geht. Der Dreifußgang wiederum ist das wohl statisch stabilste und schnellste Laufmuster das eingesetzt werden kann. Dies gilt allerdings nur für ebenes Gelände. Allgemein sind statische Laufmuster nur auf flachem Terrain anwend-

bar, da keine Rückführung von Informationen der Umgebung oder des Untergrundes erfolgt. Sie werden als eine sich wiederholende Sequenz fortlaufend abgespielt.

### 2.5.2 Reaktive Laufverfahren

Bei reaktiven Steuerungen (reaktive Laufverfahren) werden Informationen aus der Umgebung mittels Sensoren zur Steuerung zurückgeführt. Es wird eine Größe aus der Umgebung gemessen und als elektrisches Signal der Steuerung mitgeteilt, die daraufhin eine entsprechende Handlung veranlasst. Dabei gibt es keinerlei Grenzen hinsichtlich der erfassten Messgröße. Zum Beispiel kann die Neigung des zu besteigenden Terrain gemessen werden und die Beine könnten sich nach dem gemessenen Wert angewinkelt ausrichten. Ein weiteres Beispiel wären Drucksensoren an den Füßen, mit welchen der Kontakt zwischen Boden und Fuß erfasst und durch die Steuerung verarbeitet wird.

### 2.5.3 Hybride Verfahren

Es gibt aber auch hybride Steuerungen die, je nach Art des Untergrundes oder der Umgebung, zwischen verschiedenen Basisverhalten wechseln. Hybride Verfahren besitzen aber auch Nachteile, wie das Entstehen von unbedachten Verhaltensweisen, insbesondere wenn viele Sensorinformationen auf komplexe Weise beteiligt sind. Es können ungeahnte Verhaltensweisen mit überraschenden Effekten entstehen. Mischungen aus vielen Verhaltensweisen sollten eher vermieden werden. Besser ist es dann spezifische und anwendungsorientierte Basisverhaltensweisen zu integrieren, zwischen denen gewechselt werden kann.

### 2.5.4 Planende Verfahren

In erster Linie ist es Ziel der planenden Laufverfahren, eine Störung erst gar nicht auftreten zu lassen. Möglich ist ein solches Verhalten, indem die Umgebung analysiert wird und die Fußauftritte über entsprechend unkritische Bodenbereiche verlaufen. Im Idealfall verläuft die Körperbewegung dann am Stück ohne Unterbrechungen auf Grund von Berechnungen. Genau betrachtet lassen sich planende Verfahren in Vorausplanende- und in Echtzeit-Verfahren unterteilen. Der in dieser Arbeit

analysierte Laufplanungsalgorithmus, so wie ihn Herms 2003 in [17] entwickelt hat, zählt eher zu den Voraus-planenden Verfahren, denn die Planung erfolgt nicht während des Laufens, sondern bereits im Voraus. Die weiterentwickelte Ruffler-Version des Laufplanung-Algorithmus wurde schon soweit modifiziert, dass das Planen selbst während des Laufens stattfindet. Die zu durchlaufende Strecke kann dabei mit einer vorgegeben Auflösung in viele Teilstreckenstücke unterteilt werden. Bei Echtzeit-Planung muss zuerst die neu erkundete Umgebung ausgewertet werden, im Anschluss kann die Laufplanung stattfinden. Bei vorausschauender Planung muss das Gelände überwiegend bekannt sein, da sonst keine Bewegung im Voraus planbar ist. In der Herms-Version des Laufplaners wurde die Umgebung als Höhenkarte übergeben mit deren Hilfe der Bewegungsvorgang bis zum Ziel vorausgeplant wird.

### 2.5.5 Neuronale Verfahren

Verstärkt geforscht wird auch in Richtung neuronaler Laufverhalten [1]. Die Roboter-Laufsteuerung basiert dann überwiegend auf neuronalen Netzen. Diese entscheiden meistens durch Bewertung nach unterschiedlichen Kriterien (Reize). Neuronale Netze werden in der Regel auf bestimmte Verhaltensweisen trainiert. Doch gerade bei Roboterapplikation ist auch hier zur Vorsicht gebeten, da nicht mit vollständiger Sicherheit vorhersagbar ist, wie das Verhalten sein wird. Bei komplexen neuronalen Netzen ist oft nicht ermittelbar, warum diese eine bestimmte Entscheidung getroffen haben. Eine analytische Methode zur Vorhersagbarkeit sowie Nachweisbarkeit ist bislang nicht bekannt.

## 2.6 3D-Sensorik

In diesem Abschnitt werden die Grundverfahren zur 3D-Sensorik erläutert. Der Hintergrund hierfür ist nicht nur die Hardwareimplementierung der 3D-Bildeinheit beim Akrobat-Laufroboter, sondern auch das Verfahren, welches in Kapitel 9 vorgestellt wird. Unter dem Begriff 3D-Sensorik versteht man das Erfassen und Verarbeiten von Bildern, die auch Tiefen- oder Höheninformation enthalten. Diese zusätzliche Information ist für jede 2D Position hinterlegt und repräsentiert dann die "dritte Dimension".

3D-Kameras beruhen auf den Verfahren der elektrooptischen Distanzmessung. Diese werden unterschieden in:

- Impulsmessverfahren (Laufzeitmessung)
- Entfernungsmessung mittels Dopplereffekt
- Entfernungsmessung durch Interferenzen (Interferometrie)
- Phasenvergleichsverfahren
- Distanzmessung durch Lasertriangulation

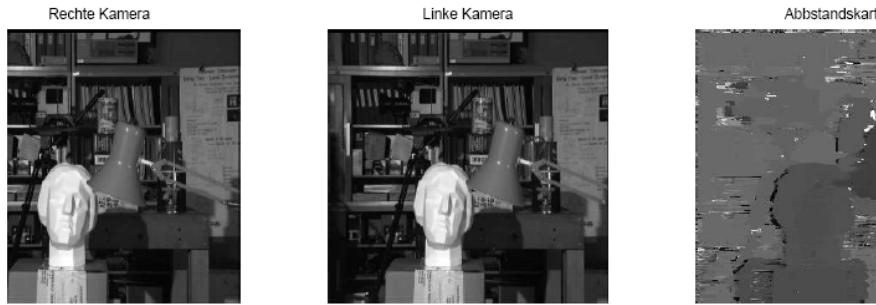
Für Sensoren, die im Bereich autonome mobile Roboter eingesetzt werden, eignen sich besonders Sensoren die mit folgenden Verfahren arbeiten:

- Laser-Distanzmessung (Triangulation und Laufzeit)
- Stereo-Vision-Verfahren (Triangulation anhand optischer Merkmale)
- Time-of-Flight-Verfahren (Laufzeit, IR-Licht Array)
- Lichtschnitt-Verfahren (Triangulation)

### 2.6.1 3D-Laser-Distanzmessgeräte

Häufig werden 3D-Laser-Entfernungsmesser im Bereich der Umgebungserkundung und der 3D-Kartierung genutzt. Auf Grund der benötigten Rechenleistung sind diese nicht immer die beste Lösung für eine Anwendung auf mobilen Robotern. Besser geeignet sind 2D-Laser-Entfernungsmesser, die meistens durch eine Rotationsbewegung um die dritte Dimension erweitert werden. Unterschieden wird dabei in punkt- oder streifen-basierende Laserscanner. Diese projizieren einen Strahl oder auch einen Streifen auf eine Oberfläche, der mithilfe von Sensoren und dem Prinzip der Triangulation der gesuchte Abstand berechnet wird. Ein Vorteil der Triangulation ist der Umstand, dass es sich um rein trigonometrische Zusammenhänge handelt. Die Messung kann darum kontinuierlich erfolgen und eignet sich damit gut zur Abstandsmessung an bewegten Objekten.

Einer weitere Methode auf der Laserabstandmesser basieren, ist die Messung der Laufzeit des Lichtes (Time-of-Flight - TOF). Hierbei wird ein Laserstrahl zum Objekt gesendet, der dort reflektiert wird. Die Dauer, die das Licht für die Strecke benötigt, wird dabei gemessen. Über die Kenntnis der Lichtgeschwindigkeit  $c_0$  und



**Abbildung 2.13:** Berechnung von Höhenkarten im Stereo-Vision System, [30]

der gemessenen Zeit  $t$ , die das Signal von A nach B braucht, lässt sich folgender Zusammenhang herstellen:

$$D = \frac{c_0}{n} \cdot t/2 \quad (2.1)$$

,wobei  $n$  die Brechzahl der Atmosphäre ist. Die Lichtgeschwindigkeit  $c_0$  beträgt  $299\,792\,458 \text{ m s}^{-1}$  (im Vakuum).

Mit der Signalgeschwindigkeit  $c = \frac{c_0}{n}$  lässt sich dann folgender Zusammenhang aufstellen:

$$D = c \cdot t/2 \quad (2.2)$$

Wobei  $D = B - A$  gilt, also die Differenz zwischen A und B die Strecke ist. Das problematische bei dem Verfahren ist die genaue Messung der sehr kleinen Zeit  $t$ . Eine sehr kurze Dauer von 6,67 ns entspricht bereits 1,0 m.

## 2.6.2 3D-Vision-Kameras

Dieses Verfahren wurde von J. Bout in der Arbeit [7] beim Roboter Lauron implementiert. Dabei wurde das Verfahren in der Open-Inventor Simulationsumgebung eingebettet und auch am Laufplanungs-Algorithmus getestet und analysiert.

Hier soll jedoch das Grundprinzip des Verfahrens erläutert werden, das in Abbildung 2.14 dargestellt ist.

Stereo-Vision-Kameras erzeugen ihre 3D-Messungen durch Berechnungen anhand zwei versetzter Bildaufnahmen, so wie in Abbildung 2.13. Zunächst nehmen die zwei Kameras, die versetzt positioniert sind, gleichzeitig Bilder von der gleichen Szene auf. Auf beiden Bildern wird der gleiche Referenzpunkt gewählt und dessen

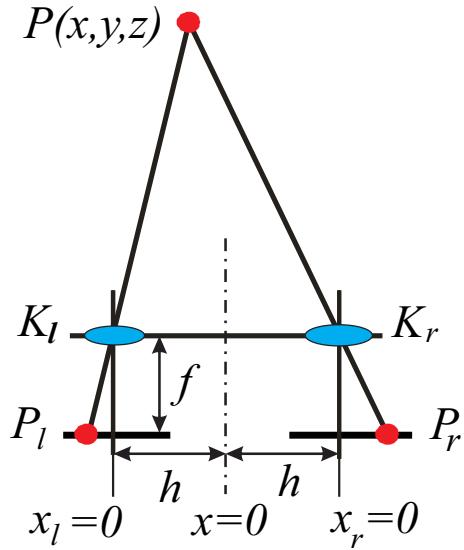


Abbildung 2.14: Prinzip der Triangulation, [4]

Entfernung gemessen. Durch die Disparität beider Aufnahmen lässt sich mit Hilfe der Linsengleichung und der Triangulation so die Tiefe bestimmen.

Für exakte Messungen ist es besonders wichtig, dass auch die Kamera exakt kalibriert ist. Meistens werden die Daten dann über eine Schnittstelle, die hohe Datenraten übertragen kann (wie IEEE 1394 (FireWire) oder USB3.0), auf einen PC übertragen, der diese dann verarbeitet.

#### *Intel Realsense R200*

Auf dem Bild 2.15 ist ein 3D-Sensor, der Baureihe Realsense vom Hersteller Intel, zu sehen. Dieser arbeitet mit dem Stereo-Vision Prinzip.

#### 2.6.3 Time-of-Flight-Kameras

Über die Phasenverschiebung von emittierten modulierten Infrarotlicht wird bei diesem Verfahren die Tiefenmessung ermöglicht. Das Prinzip ist ähnlich wie das be-



Abbildung 2.15: 3D-Kamera, basierend auf dem Stereo-Vision-Verfahren

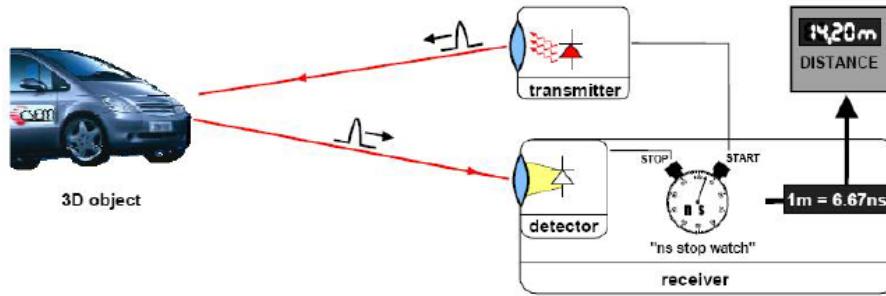


Abbildung 2.16: Funktionsprinzip ToF (Time-Of-Flight), [20]

reits beschriebene Verfahren der Laser-Abstandsmessung, das auf dem Prinzip der Lichtlaufzeitmessung basiert. Da die Geschwindigkeit des Lichtes bekannt ist, kann die Entfernung gemessen werden, indem die Laufzeit gemessen wird, die das Signal benötigt um von A nach B zu gelangen.

Beim den zugehörigen Sensoren liegen die aktive Lichtquelle und der Empfänger in der Regel sehr nah beieinander, so auch beim PMD PicoFlex. Das erleichtert die Kalibrierung und kompensiert auch Schatteneffekte. Abbildung 2.16 zeigt das ToF-Verfahrensprinzip.

Der Messung der Laufzeit muss hierbei im Nanosekunden-Bereich erfolgen. Der Lichtimpuls wandert zum Ziel und zurück, wobei die benötigte Dauer gemessen wird. Für 1,0 m wird eine Zeit von 6,67 ns benötigt, wobei genau genommen das Licht den Weg, für Hin- und Rückweg, doppelt durchlaufen muss.

Sender und Empfänger müssen aus diesem Grund bereits im Voraus genau synchronisiert werden. Zur Verfügung stellen die Sensoren in der Regel:

- Farbkodierte-Bilder (entspricht gemessener Phasenverschiebung)
- Intensitätskodierte-Bilder (entspricht gemessener Amplitude)
- Punktwolken (Darstellung der Tiefeninformation zu jedem Messpunkt)

Die Tiefe/Entfernung werden über Farben im Bereich von rot (Nahbereich) bis violett (entfernt), dargestellt, was auch als „Falschfarendarstellung“ bezeichnet wird. Bei dem Verfahren treten des öfteren Messfehler auf, die unbrauchbare Werte zur Folgen haben. Diese können beispielsweise durch eine zu niedrige Modularität des Lichtes entstehen, die jedoch mit entsprechenden Verfahren kompensiert werden können.

## 2 Grundlagen

---

Die Entscheidung, welcher 3D-Sensor im System des Akrobat verbaut wird, fiel auf einen Sensor des Herstellers PMD, Modell PicoFlex, der mit diesem Verfahren funktioniert. Dieses Modell benötigt nicht zwingend einen USB3.0 Anschluss und belastet auch weniger stark die Rechenkapazitäten des UP-Boards. Die Auswertung erfolgt nämlich schon auf der Seite des Sensors, der somit die Tiefeninformation direkt als Punktwolke liefert. Zudem besitzt er eine Leistungsaufnahme im niederen Bereich, was dem autarken Betrieb zu Gute kommt.

## **Kapitel 3**

# **Aufbau des Laufplaners**

Der Laufplaner wurde im Jahr 2003 für den Laufroboter Lauron-III entwickelt. Der Planer und auch die Simulationsumgebung, die mit Open-Inventor aufgebaut wurde, ist von André Herms in der Arbeit [17] entwickelt worden.

Die Arbeit von Herms stellt die Grundlage dar, auf der die späteren Weiterentwicklungen durch Uli Ruffler, aus dem Jahre 2006 aufbauen. U. Ruffler hat diverse Probleme erkannt, die ein Portieren auf ein reales System erschweren. Des Weiteren hat er den Laufplaner in der Art erweitert, dass ein Erfassen des Umfeldes innerhalb der Simulationsumgebung über eine Stereovision-3D-Kamera möglich ist. Der zugehörige Algorithmus stammt von Richard Bade und wurde in Form eines Anytime-Algorithmus implementiert.

In diesem und folgenden Kapiteln wird an entsprechenden Stellen explizit darauf hingewiesen, sofern etwas von U. Ruffler verändert oder erweitert wurde. Ebenso wird an selber Stelle der Hintergrund für das Einbringen der Änderung erläutert.

### **3.1 Anforderungen**

Benötigt wurde entweder ein Verfahren, das immer innerhalb der Dauer zu einer Lösung kommt oder ein Anytime-Verfahren, das unabhängig von dessen Laufzeit ist und unter Garantie mindestens eine gültige Lösung findet. Herms realisierte die Laufplanung über Heuristiken, welche von ihm genaustens untersucht wurden. Die performantesten Heuristiken, welche auch die besten Resultate zeigten wie das

„Random Sampling“, wurden dann für den Algorithmus in seiner Arbeit [17] verwendet.

## 3.2 Übersicht der Funktionsweise

Der Laufplaner als Gesamtsystem lässt sich mehr oder weniger in drei Hauptprozesse unterteilen: Inn die Planung der Schritte, die im lokalen RKS (Roboter-Koordinaten-System) stattfindet, in die Bewegungs-Planung des Roboter-Körpers (findet innerhalb des Welt-Koordinaten-Systems statt) und in die Zustands-Festlegung der Füße.

In der Regel wird die Planung der Schritte und die der Route separat behandelt. Mit anderen Worten ist keiner der Bereiche für den anderen zuständig. Bei planenden Laufverfahren ist eine derartige Trennung nicht ganz möglich, da beide Bereiche von einander abhängig sind. Es kann keine Schrittplanung erfolgen, ohne die Information über die Zielposition oder den Wegverlauf. Denn das Gelände bzw. die Bodengegebenheiten fallen auch mit in die Bewertung der Lösungen ein und beeinflussen auch die Anzahl guter gefundener Lösungen.

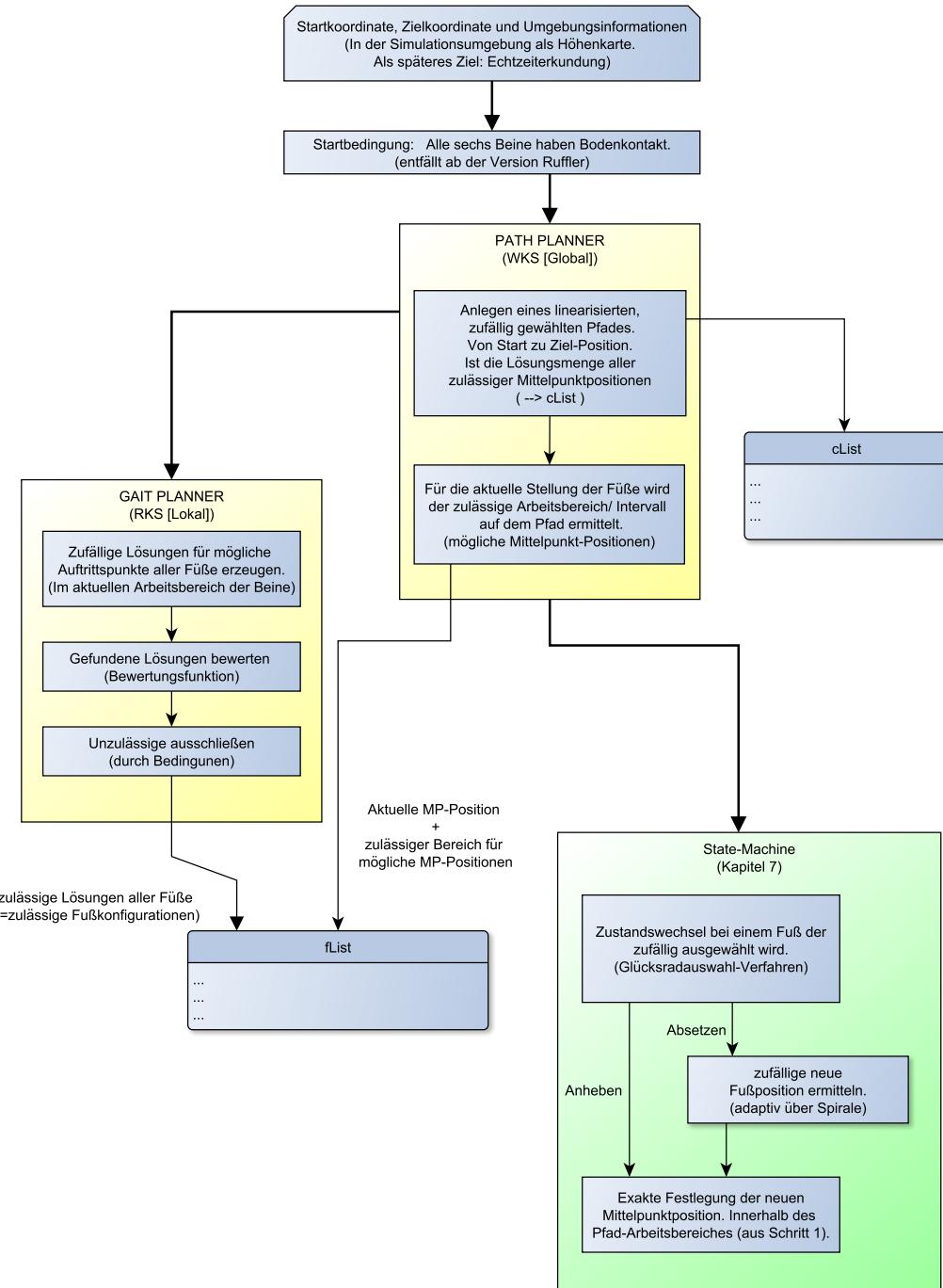
## 3.3 Laufplanung als Optimierungsproblem

André Herms betrachtete in seiner Arbeit [17] das Laufplanungsproblem als ein Optimierungsproblem. Bei dieser Art der Betrachtung gibt es im Allgemeinen einen Raum möglicher Lösungen und eine Bewertungsfunktion, die auch als Fitnessfunktion bezeichnet wird. Angenommen  $f$  sei die Bewertungsfunktion der Lösungsmenge  $\Omega$ .

$$f : \Omega \rightarrow \mathbb{R} \quad (3.1)$$

Gesucht wird eine Lösung  $x \in \Omega$  die einen größtmöglichen Funktionswert  $f_{\text{rating}}(x)$  aufweist.

Man spricht dann von einem Maximierungsproblem. Umgekehrt gibt es auch Minimierungsprobleme, bei denen die beste Lösung die mit der geringsten Bewertung ist.



**Abbildung 3.1:** Übersicht zum Programmablauf

Um eine entsprechende Aussage darüber machen zu können, inwiefern der erhaltene Wert nun gut ist, muss der Wert der „besten“ Lösung bekannt sein. Hierbei spricht man dann von „eigentlichen Optimierungsproblemen“ mit einer Formulierung wie dieser:

$$\max f(x) | x \in \Omega \quad (3.2)$$

Der beste Wert kann als Referenz angesehen werden, sofern dieser eben bekannt ist. Aufgrund der Komplexität vieler, auf dieser Weise betrachteten Probleme ist es oft gar nicht möglich einen „allerbesten Wert“ ausfindig zu machen. Wird noch zusätzlich ein Grenzwert  $g \in \mathbb{R}$  hinzugezogen, so entsteht aus dem Optimierungsproblem ein sogenanntes Entscheidungsproblem. In der Informatik wird oft genau dieses betrachtet. Durch die weniger gute Handhabung von reellen Zahlen werden solche Probleme in der Regel auch mit einer diskreten Bewertungsfunktion  $f : \Omega \rightarrow \mathbb{N}$  betrachtet.

## 3.4 Heuristiken

Unter Heuristik versteht man eine analytische Vorgehensweise, die mit begrenzten Informationen und kurzer Dauer trotzdem zu wahrscheinlichen Ergebnissen oder weiterverwendbaren Lösungen kommt. Es gibt eine Vielzahl von Heuristiken, eine der wohl bekanntesten ist das Ausschlussverfahren oder in der Statistik das zufällige Stichproben-Verfahren.

Auf Grund der Resultate die Herms in seiner Arbeit [17] verglich, wurde der Focus auf diese Heuristiken gelegt:

- Random Sampling
- Lokale Suche
- Simulated Annealing

Die besten Ergebnisse hatte Herms mit dem Random-Sampling erreicht. Uli Ruffler entwickelte den Laufplaner unter Verwendung des Random-Sampling Verfahrens weiter und beschränkte sich auf dieses, da selbiges die besten Resultate lieferte. Auf Grund der ohnehin sehr komplexen Funktionsweise, wurde dies auch im Rahmen dieser Arbeit so beibehalten. Die „Lokale Suche“ und das „Simulated Annealing“

sind weitere Heuristiken zur Lösungsfindung, auf die hier nicht weiter eingegangen wird, aber die Herms in seiner Arbeit [17] untersucht und auch angewendet hat.

### 3.5 C-Space/Konfigurationsraum

Die Anzahl unabhängiger Freiheitsgrade eines Systems entsprechen der Dimension seines Konfigurationsraumes. Eine Roboter-Konfiguration  $q$  ist die Festlegung der Positionen aller Punkte eines Roboters. Im Normalfall wird diese über einen Vektor aus Positionen und Orientierungen ausgedrückt.

Die Definition des Konfigurationsraums lautet folgendermaßen: Sei  $\mathcal{W} \subset \mathbb{R}^3$  der Raum, indem sich der Roboter mit seiner abstrakten Beschreibung  $\mathcal{R}$  bewegt. Sei  $F(q)$  der Roboter mit der Konfiguration  $q \in \mathcal{C}$ .

Hindernisse werden als „Obstacles“<sup>1</sup> bezeichnet, deren Menge  $\mathcal{O}$  eine Teilmenge des Arbeitsbereichs  $\mathcal{W}$  ist.

Demnach gilt:  $\mathcal{O} \subset \mathcal{W}$ . Der freie Bereich innerhalb des Konfigurationsraums wird mit

$$\mathcal{C}_{\text{free}} = \{q \in \mathcal{C} | F(q) \cap \mathcal{O} = 0\} \quad (3.3)$$

ausgedrückt.

Die Menge der Obstacles lässt sich dann als Restmenge von gesamtem Konfigurationsraum und freiem Konfigurationsraum, über den folgenden Zusammenhang, beschreiben.

$$\mathcal{C}_{\text{obstacles}} = \mathcal{C} / \mathcal{C}_{\text{free}} \quad (3.4)$$

Herms erwähnt die hohe Anzahl der auftretenden Freiheitsgrade, welche durch die Beine des Laufroboters entstehen. Er kommt auf eine Anzahl von insgesamt 18 Freiheitsgraden durch die Beine und 2 die durch die Translationen entlang der x-Achse und y-Achse entstehen, also insgesamt 20 Freiheitsgrade. Diese hohe Anzahl resultiert aus den sechs Beinen, wobei jedes Bein drei Freiheitsgrade besitzt.

Die Freiheitsgrade der Beine sind auf das lokale RKS zurückzuführen und für die Handlungen im WKS, wie der Routenplanung, kaum von Relevanz, da diese (lokalen) Freiheitsgrade an den Robotermittelpunkt gebunden sind. Der bestbekannte und deterministische Algorithmus zum Lösen des „Motion Planning“-Problems

---

<sup>1</sup>Englischer Begriff für Hindernisse, der im Bereich der Pfadplanung üblicherweise verwendet wird.

#### Art der Bewegung

Translation	entlang der x-Achse
Translation	entlang der y-Achse
Translation	entlang der z-Achse
Rotation	um die x-Achse
Rotation	um die y-Achse
Rotation	um die z-Achse

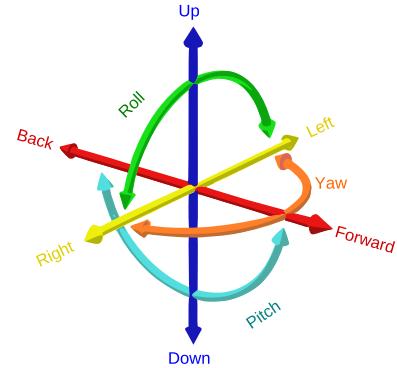


Abbildung 3.2: Die sechs Freiheitsgrade eines starren Körpers innerhalb des Raumes, [13]

[8] benötigt im Konfigurationsraum exponentielle Laufzeit. Bei solch einer hohen Anzahl der Freiheitsgrade ist die Berechnung des Konfigurationsraums äußerst aufwendig und zeitintensiv.

Aus diesem Grund kam zur Zeit, als Herms die Arbeit verfasste, nur der Lösungsweg über Heuristiken in Frage, da die Berechnungen online (im Roboter) erfolgen sollten. Die Berechnung sollte performant und in polynomialem Laufzeit geringen Faktors erfolgen. Es wurde in Kauf genommen, dass nur suboptimale Lösungen gefunden werden, dafür aber die polynomiale Laufzeit vorhanden ist.

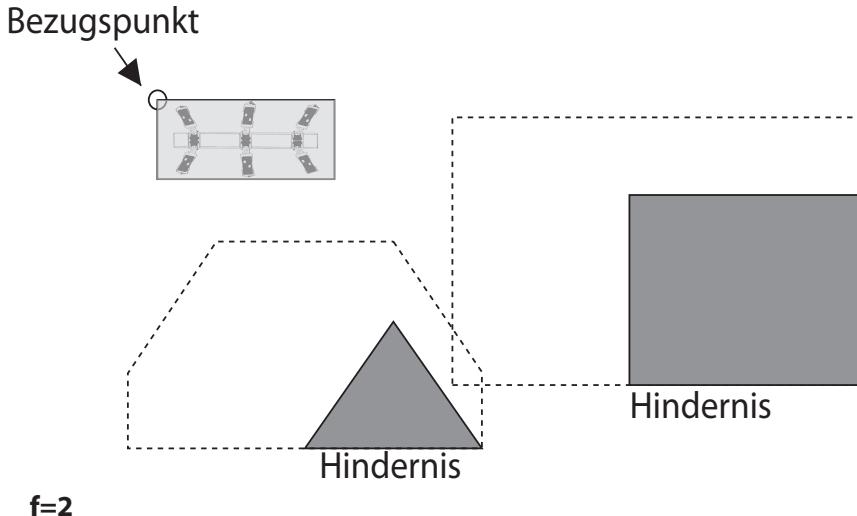
Innerhalb des dreidimensionalen Raumes wird eine Konfiguration  $q$  mit der Form  $(x, y, z, \alpha, \beta, \gamma)$  vollständig beschrieben.

Zur Veranschaulichung ist dies in Abbildung 3.2 dargestellt.

Als Beispiel wurde von Herms das Szenario von 3.3 gezeigt. Dieses beruht allerdings auf lediglich zwei Freiheitsgraden, die durch Translationen entlang der x-Achse und der y-Achse zurückzuführen sind. Bei solchen harten Einschränkungen sind die Möglichkeiten zum Finden eines Pfades natürlich entsprechend eingeschränkt.

Die gesuchte Kurve (Pfad), die zwischen den Hindernissen verläuft, ist nicht vorhanden, da sich im Konfigurationsraum die Hindernisse überschneiden. Viel sinnvoller wäre eine mehr abstrahierte Betrachtungsweise, in welcher der Roboter zum Beispiel als starrer Körper im Raum betrachtet wird und dies ausschließlich im globalen Koordinaten-System.

Vereinfacht lässt sich der Roboter als ein Kasten betrachten, der solche Maße besitzt, dass Arbeitsbereiche der Beine/Füße und die des Körpers damit abgedeckt sind. Die wichtigsten Freiheitsgrade, mit denen dieser „Kasten“ auf einer Landkar-



**Abbildung 3.3:** C-Space bei zwei Freiheitsgraden durch x- und y-Translation

te (Sicht von oben in 2,5-D) um Hindernisse herum navigiert werden kann, sind diese:

- Translation entlang x-Achse
- Translation entlang y-Achse
- Rotation um z-Achse

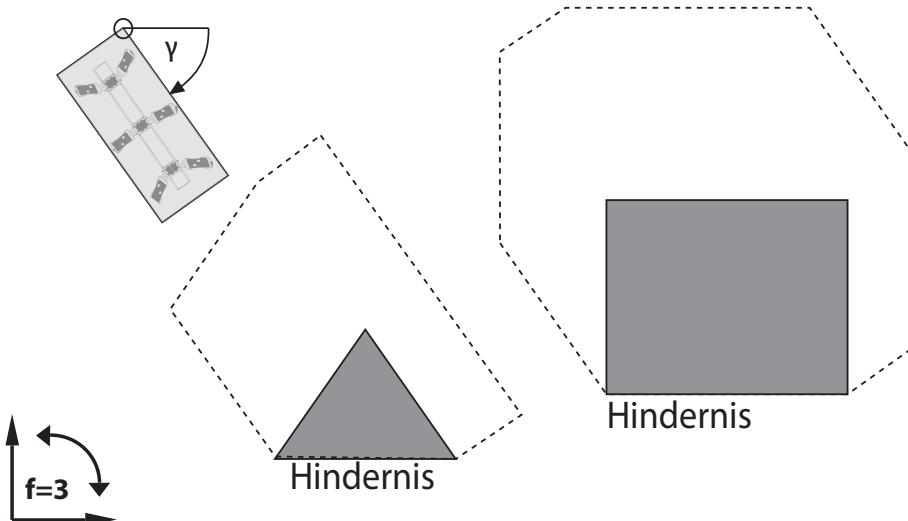
Somit lässt sich die Konfiguration  $q$  als 3-Parameter Darstellung repräsentieren.

$$q = (x, y, \gamma) \quad (3.5)$$

In 3.4 wird das gleiche Beispiel nochmal aufgegriffen, allerdings mit drei Freiheitsgraden und einer Roboter-Konfiguration, die verdreht ist. Hinzugekommen ist die Rotation um die z-Achse, die hier mit  $\gamma = 60^\circ$  im Uhrzeigersinn gedreht, vorliegt.

Ein solches Rotieren „auf der Stelle“, bei dem der Roboter-Mittelpunkt an der gleichen Stelle bleibt, ist sowohl bei beiden LAURON- als auch beim Akrobat-Laufroboter gegeben. Es ergibt sich eine ganz andere Form des C-Space, mit der es jetzt durchaus einen Weg gibt, um den Raum zwischen den Hindernissen zu durchlaufen. Mit anderen Worten muss es Kurven geben, die dazwischen verlaufen.

Durch die Problematik des komplexen C-Spaces werden in der Praxis die sogenannten „Motion Planning“-Probleme höherer Dimension mit einfacheren Verfahren ge-



**Abbildung 3.4:** C-Space bei drei Freiheitsgraden durch x-, y-Translation und z-Rotation.

löst, indem diese gemischt angewandt werden. Beispiele dieser Verfahren sind die Diskretisierung, Suche innerhalb von Graphen oder das Erzeugen von Zufallswerten (Random-Sampling) wie es auch von Herms im Laufplaner verwendet wird.

#### PRM

Ein weiteres und gerne verwendetes Verfahren sind sogenannte PRM's. PRM steht für „Probabilistic-Road-Maps“ und funktioniert, wie der Name schon vermuten lässt, auf Basis einer Straßenkarte.

Das Verfahren findet oft Lösungen bzw. Pfade, über die Hindernisse auf kurzem Wege umgangen werden. Grob kann das Verfahren in zwei Bereiche unterteilt werden:

- Kartenkonstruktion (Pre-processing)
- Plan-Generierung (Query-processing).

Allerdings hat auch dieses Verfahren Schwachstellen. Diese liegen beim Verbinden von Knotenpunkten, deren Verlauf zwischen engen Passagen aus Hindernissen hindurchführt.

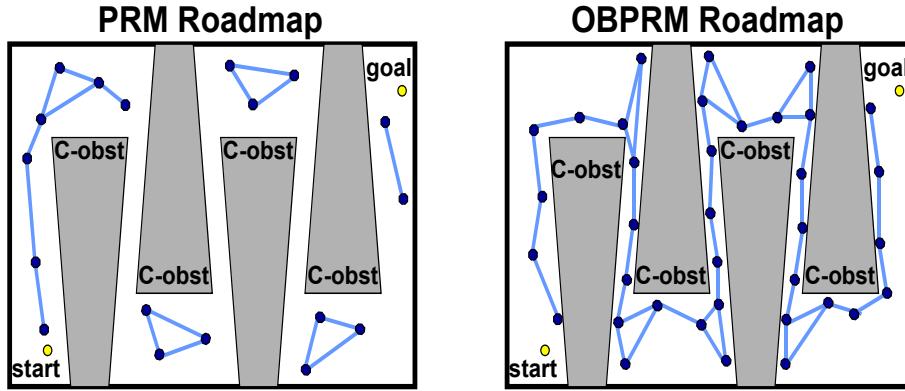


Abbildung 3.5: (Obstacle-Based)-Probabilistic-Road-Map [29]

Dieses Problem lässt sich besser mit dem weiterentwickelten OBPRM<sup>2</sup>-Verfahren [2] in den Griff bekommen. Bild 3.5 zeigt einen Vergleich der beiden Verfahren, [21].

Beide Verfahren sind jedoch nicht für Echtzeit-Erkundungen gedacht, da auch bei diesen Verfahren die Kenntnis über das gesamte Gebiet vorhanden sein muss. Im Jahr 2018 wurde im Journal „IEEE Robotics and Automation Letters“ ein Verfahren vorgestellt, das speziell für Echtzeit-Erkundungseinsätze, auf grobem Gelände, entwickelt wurde. Dieses wird später, an passender Stelle in Kapitel neun 9.3, als ein mögliches Verfahren für den Laufroboter Akrobat vorgestellt.

Bei diesem Verfahren werden auf zwei Arten Information über die Umgebung gewonnen. Dies erfolgt zum einen über Sensoren und zum anderen über die Bewegung des Roboters selbst. Die gesammelten Informationen werden zu einer gemeinsamen Fusions-Karte verarbeitet. Im Falle einer zukünftigen Implementierung dieses Verfahrens könnte der Akrobat die Informationen über die Umgebung selbst gewinnen und auswerten.

#### *Hinweis zum Source-Code*

Die Pfad-Festlegung des Körpers findet beim Laufplaner von A. Herms ohne Beachtung von Hindernissen (Obstacles) statt. Es wird zwar ein Weg per Zufall angelegt, doch die Qualität wie dieser verläuft ist ebenso zufällig. Das Ziel in der Arbeit von Herms war nur die Laufplanung selbst, doch wie bereits erwähnt ist bei planenden

<sup>2</sup>OBPRM = Obstacle-based-Probabilistic-Road-Maps. Innerhalb des Bereichs der Pfadplanung werden Hindernisse als Obstacles bezeichnet.

---

**Listing 1** In der cList angelegter Pfad für die Mittelpunkt-Verschiebung

---

```
void Movement::createRandom2( float _lengthC){
    this->initCenterPos = Position( 0., 0.);
    int lengthCenter = 0;
    float r;

    // choose a length for the center list (exponentially distrib.)
    do {
        r = (float)rand() / (float)RAND_MAX;
        lengthCenter++;
    } while (r <= _lengthC);

    this->cList.clear();

    // create random center moves
    for (int i = 0; i < lengthCenter; i++){
        float r1 = (float)rand() / (float)RAND_MAX; // random number in (0,1)
        float r2 = (float)rand() / (float)RAND_MAX; // random number in (0,1)
        float r3 = (float)rand() / (float)RAND_MAX; // random number in (0,1)
        float r4 = (float)rand() / (float)RAND_MAX; // random number in (0,1)
        // a Position is defined as:
        // typedef SbVec2f Position (2D float vector)
        Position dest(r1 * 5000. - r2 * 5000., r3 * 5000. - r4 * 5000.);

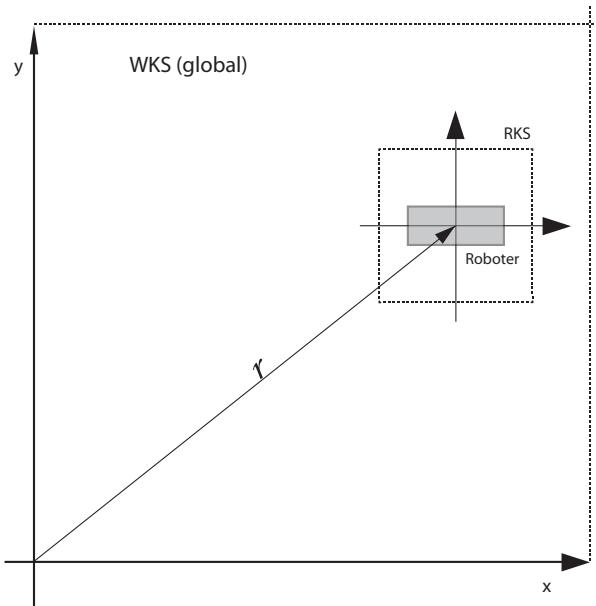
        if ( i == lengthCenter - 1 ) {
            // force endPos to be reached
            dest = this->endPos;
        }
        cList.push_back(dest);
    }
}
```

---

Verfahren auch ein Mindestmaß an Routenplanung notwendig um Schritte planen zu können.

## 3.6 Koordinatensysteme

Für den Laufplaner-Algorithmus werden zwei Koordinatensysteme wie sie in Abbildung 3.6 gezeigt sind, verwendet. Ein globales, das die gesamte Umgebung, also das Terrain umfasst und ein lokales, in dem die Schrittplanung erfolgt.



**Abbildung 3.6:** Zusammenhang zwischen WKS(global) und RKS(lokal)

#### *Hinweis zu Methode des Source-Codes*

Die Entfernung zwischen Zentrum des RKS und dem Nullpunkt des WKS, kann durch folgende Funktion ermittelt werden:

```
static inline float distanceToZero( )
```

#### **3.6.1 RKS : Roboter-Koordinaten-System (lokal)**

Die Position des RKS innerhalb des Weltkoordinaten-Systems lassen sich über die Mittelpunkt-Position des Roboters  $\vec{r}$  zuordnen. Die Roboterposition  $\vec{r}$  ist gleichzeitig auch immer der Ursprung des RKS und der Mittelpunkt des Körpers.

Ist das lokale Koordinaten-System mit einer Auflösung von 256x256 Werten definiert, so befindet sich der Roboter im Ursprung des lokalen RKS, an der Position (128, 128).

#### 3.6.2 WKS : Welt-Koordinaten-System (global)

Innerhalb des WKS<sup>3</sup> können die Eckpunkte des RKS<sup>4</sup> durch den Zusammenhang

$$pos(i,j) = (\vec{r}_x + (i - 128), \vec{r}_y + (j - 128)) \quad (3.6)$$

ermittelt werden.

#### 3.6.3 Pfadfestlegung in der Version Herms

In der Laufplanerversion Herms wird die gesamte Pfadfestlegung beim Start des Programms angelegt. Die Einteilung in Segmente bzw. Abschnitte findet ebenfalls zu Beginn statt. Das komplette „Movement“, welches die gesamte Bewegung von Startposition zur Zielposition beschreibt, hängt davon ab. Denn die linearisierte Route, nach dem Verfahren aus Listing 1 definiert die zulässigen Positionen im WKS, welche für den Mittelpunkt zulässig und somit möglich sind (cList).

#### 3.6.4 Pfadfestlegung in der Version Ruffler

Uli Ruffler änderte dies, indem er den Laufplaner so modifizierte, dass die Route nicht komplett am Stück bei Programmstart angelegt wird. Somit werden immer nur Teilstücke der Route geplant und auf diese dann der Schritt-Planer angewendet. Der Hintergrund der anderen Vorgehensweise ist es, den Roboter mehr für den realen Einsatz vorzubereiten. Da es das zukünftige Ziel ist, den Roboter autonom, mit Echtzeit-Erkundung zu betreiben, hat Ruffler hiermit gute Vorarbeit geleistet. Die Route wird zu Beginn *nicht* auf einmal, von Startposition bis zur Zielposition, angelegt, sondern sukzessive während des Schreibens. Ebenso erfolgt die Schritt-Planung nur noch für das anstehende Teilstück des Pfades (Route). Als Folge davon wird auch kein komplettes Movement mehr kreiert, welches die Bewegung von Start zum Ziel beschreibt. Movements werden ab der Ruffler-Version stückweise erzeugt und anschließend verkettet werden. Dies wird in Kapitel acht genauer beschrieben 8.

---

<sup>3</sup>Ist die Abkürzung für das Welt-Koordinaten-System

<sup>4</sup>RKS steht für Relatives-Koordinaten-System (lokal) oder auch Roboter-Koordinaten-System.

## Kapitel 4

# Eingabe-Informationen

Innerhalb der Simulationsumgebung, die mit Open-Inventor<sup>1</sup> aufgebaut ist, werden zum Aktivieren des planenden Laufens mindestens drei Informationen benötigt. Diese werden zusammenfassend als Eingabe-Informationen bezeichnet. Dabei handelt es sich um diese drei:

- Startposition (2d-float-vector)
- Zielposition (2d-float-vector)
- Informationen über Umgebung (3d-vector, da Information über Höhen zu allen Punkten enthalten ist)

### 4.1 Start- und Zielposition

Bei der Start- und Zielposition handelt es sich um Koordinaten, über welche die entsprechenden Positionen innerhalb eines zweidimensionalen Feldes definiert sind. Das Weltkoordinatensystem entspricht einem solchen 2D-Feld. Dieses ist von kartesischer Form und verläuft positiv in die x/y-Richtung. In dieser Arbeit wird dieses vereinfacht als WKS<sup>2</sup> bezeichnet. Der Ursprung liegt hier nicht in der Mitte, im Gegensatz zum lokalen RKS<sup>3</sup>.

---

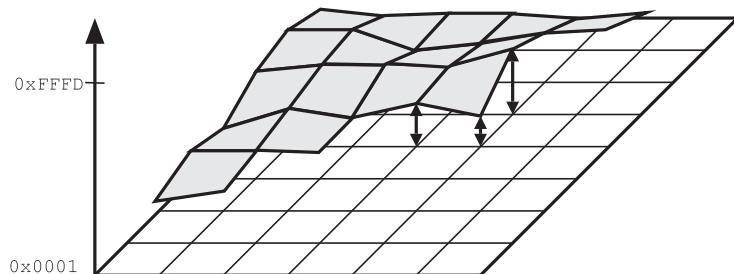
<sup>1</sup>...

<sup>2</sup>WKS ist die Abkürzung für das Welt-Koordinaten-System, welches in diesem Fall global anzusehen ist. Es ist zweidimensional und verläuft positiv mit zunehmendem x- und y-Werten

<sup>3</sup>RKS ist die Abkürzung für Roboter-Koordinaten-System. Es ist ebenfalls zweidimensional, sein Ursprung befindet sich allerdings in der Mitte. Die Position der Mitte des RKS ist identisch mit der Position des Robotermittelpunkts und somit auch mit der Position des Massenschwerpunktes.

## 4.2 Umgebungsinformationen

Umgebungsinformationen können für die Anwendung innerhalb einer Simulationsumgebung auf synthetische Weise erzeugt und übergeben werden. Hierfür eignen sich zum Beispiel Höhenkarten. Der ursprüngliche Laufplaner von André Herms [17] arbeitet nach diesem Prinzip. Damit ist dem Roboter bereits im Voraus die gesamte Karte bekannt, was beim realen Einsatz nur selten der Fall ist. Allerdings ist dies für eine Simulationsumgebung absolut ausreichend. Im realen Einsatz ist dem Roboter nur das Blickfeld bekannt und er muss die gesamte Karte erst erkunden und aus Teilstücken zusammensetzen. Wird die Umgebung durch visuelle Sensorinformationen gewonnen, so muss diese in eine Datenstruktur überführt werden. Je nach Verfahren der 3D-Bilderfassung bieten sich unterschiedliche Arten an um die Umgebung bei der Laufplanung weiterzuverarbeiten.



**Abbildung 4.1:** Terrain-map zur Eingabe der Umgebungsinformation, [17]

Abbildung 4.1 zeigt die schematische Darstellung der Umgebungsinformation bzw. der Höhenwerte des Bodens, auf dem sich der Roboter bewegt. Diese wurde synthetisch erzeugt und in Form einer Höhenkarte dem Programm übergeben. Hierfür werden Terrain-Karten im TIFF-Dateiformat verwendet. Sie repräsentieren das Gelände bzw. dessen Höhenmaße in Form von Grünwerten. Die Höhenkarte wird in ein „Mesh“<sup>4</sup> quadratischer Maschen gewandelt. Aus diesem Gitter ist das Terrain im  $\mathbb{R}^3$  modelliert und lässt sich durch eine Erweiterung von U. Ruffler auch mit Texturen versehen. Die Texturen waren auf Grund seiner Implementierung einer Stereovision-Kamera erforderlich, denn das Verfahren funktioniert wesentlich schlechter wenn die Umgebung einfarbig ist.

---

<sup>4</sup>Mesh ist der englische Begriff für Gitter. Die Darstellung geometrischer Körper als Meshes ist eine weit verbreitete Form der Darstellung, wie sie in 3D-Modellierung- oder CAD-Anwendungen verwendet wird. Eine Textur liegt üblicherweise in dieser Form der Ansicht nicht vor.

**Tabelle 4.1:** Werte der Höhenkarte [17]

Wert	Information	Beschreibung
0x0000	$-\infty$	Höhenwert kleiner als 0x0001
0xFFFFE	$+\infty$	Höhenwert größer als 0xFFFFD
0xFFFFF	undefined	Es liegt keine Information vor

### 4.2.1 Verwaltung der Daten zur Umgebung

Die Informationen über die Umgebung sollten nach damaligen Anforderungen in einer kompakten Weise realisiert werden. Zudem sollte die Verwaltung in einer abstrakten Datenstruktur stattfinden, um eine gewisse Unabhängigkeit der Sensoren zu erreichen. An dieser Stelle sei vermerkt, dass genau dies eine Verwendung auch auf anderen Systemen ermöglicht.

### 4.2.2 Darstellung des Terrains in der Simulationsumgebung

Entsprechend der damaligen Anforderung wurde versucht den Speicherbedarf gering zu halten. Somit wurde die Höhenkarte intern mit einer Genauigkeit von 256x256 Quadraten aufgelöst. Der resultierende Speicherbedarf beträgt 131072 Byte bzw. 128 kiloByte und war somit auch für den Einsatz auf Mikrocontrollern geeignet. Der eigentliche Höhenwert liegt im einem Wertebereich von 16 Bit und ist für die Ecken jeder Masche hinterlegt. Hierfür stehen 65536 mögliche Höhenwerte zur Verfügung.

Für spezielle Informationen hinterlegte Werte, sind in folgender Tabelle aufgelistet.

Ein Loch im Boden wird über den Wert  $-\infty$  charakterisiert. Bei diesem ist nur ein Überqueren zulässig, jedoch kein Auftreten. Anders ist es bei dem  $+\infty$ -Wert, denn bei Bereichen mit diesem Wert ist weder ein Überqueren noch ein Auftreten zulässig. Bildlich gesehen stellt dies eine Säule dar.

### 4.2.3 Umgebungsinformationen aus Höhenkarte

Eine Möglichkeit wäre eine Konvertierung der Sensorinformation in eine Höhenkarte (Tiff-Dateiformat), wie sie bisher für die Eingabe verwendet wurde. Diese

entspricht der ursprünglichen Art der Eingabe, wie sie für die Simulationsumgebung<sup>5</sup> von André Herms entwickelt wurde. Die Höhenwert-Information ist dann als Intensitätswert, des grünen Farbspektrums enthalten.

### 4.2.4 Umgebungsinformationen durch Stereovision-Kamera

Mit der Erweiterung von Uli Ruffler und dem Stereovision-Kamera-Algorithmus von Richard Bade wurde eine Möglichkeit implementiert, mit der Informationen über die Umgebung gewonnen werden können. Dies wurde in der Art und Weise umgesetzt, dass es dem späteren Einsatz auf einem realen Roboter möglichst gleich kommt. Die Höhenkarte ist zwar dennoch notwendig, um ein Gelände zu simulieren, aber der Roboter hat diese selbst nicht mehr zur Verfügung. Er bekommt nur noch die Umgebungsinformationen, welche durch die Stereovision-Kamera erfasst werden (Sichtfeld), [7]. Um eine Gesamtkarte zu erstellen muss der Roboter alle Teilbereiche des Szenarios gesehen haben. Diese Teilstücke müssen gespeichert werden und anhand bestimmter Merkmale zu einer Gesamtkarte verbunden werden. Dies fällt eigentlich in den Bereich der Routenplanung oder sogar in den Bereich der Kartierung. Der Algorithmus von Richard Bade ist ein Anytime-Algorithmus<sup>6</sup>, der bessere Ergebnisse liefert, desto mehr Laufzeit er zu Verfügung hat.

---

<sup>5</sup>Open-Inventor ist eine Bibliothek die es ermöglicht in objektorientierter Programmierung 3D-Modelle zu erzeugen. Die Nachfolger von Open-Inventor ist Coin-3D, wobei es sich um eine Open-Source-Implementierung der Open-Inventor-API handelt. Coin3D wurde ursprünglich als kommerzielle Software der Firma Kongsberg Oil and Gas Technologies entwickelt.

<sup>6</sup>Unter Anytime-Algorithmen versteht man Algorithmen, die beliebig oft pausiert werden können, ohne das bisherige Ergebnis zu verlieren. Wobei die Berechnung zu jeder Zeit fortgeführt werden kann. Die Qualität des Ergebnisses wird mit zunehmender Dauer der Ausführung besser. Ein Ergebnis geben diese immer zurück, wobei bei geringerer Laufzeit auch die Qualität geringer ist.

## Kapitel 5

# Planung der Körperbewegung (Pfad)

Dieses Kapitel beschäftigt sich mit der Ermittlung beziehungsweise der Festlegung eines Pfades von der Startposition zur Zielposition. Entlang dieses Pfades bewegt sich der Mittelpunkt des Roboters, der gleichzeitig auch der Ursprung des RKS ist und mit  $\vec{r}$  gekennzeichnet wird. Wie bereits erwähnt wurde, lässt sich bei den planenden Laufverfahren die Pfadplanung nicht gänzlich von der Schrittplanung trennen, da für diese entweder ein Pfad oder mindestens eine Richtung definiert sein muss, in welche die Bewegung ausgeübt werden soll. In der Simulationsumgebung war bisher durch die Höhenkarte auch gleichzeitig die gesamte Weltkarte, in der sich der Roboter bewegt, bekannt. Da dies beim späteren Betrieb auf dem echten Roboter nicht der Fall ist, muss sehr wahrscheinlich ein anderes Verfahren eingesetzt werden, um ein Ziel kenntlich zum machen.

Theoretisch wäre zwar die Festlegung einer Zielposition machbar, jedoch ist zu dieser keine Pfadplanung möglich, weil das Gelände bis zu dieser Position unbekannt ist. Bekannt ist dann lediglich die Startkoordinate und die frontale Sicht des Roboters, welche von begrenzter Weite ist. Das Gebiet muss dann zuerst erkundet werden, um eine komplette Pfadplanung zu ermöglichen.

Ein Verfahren mit einer Funktionsweise ähnlich der eines Kompasses wäre eine einfache Methode, welche keine komplette Wegbeschreibung zum Ziel erfordert, sondern lediglich eine Zielposition, die dem Roboter eine Richtung vorgibt. Bei der Beschreibung des Laufplaners wird sich auf das Verfahren beschränkt, welches die Höhenkarte verwendet, denn die Analyse des Laufplaners, von Herms entwickelt und erweitert durch Ruffler, ist das Hauptthema dieser Arbeit. Vorschläge zu anderen Verfahren, die das Problem mit der Umgebungsinformation lösen, werden in Kapitel neun nach der Laufplaner-Analyse vorgestellt.

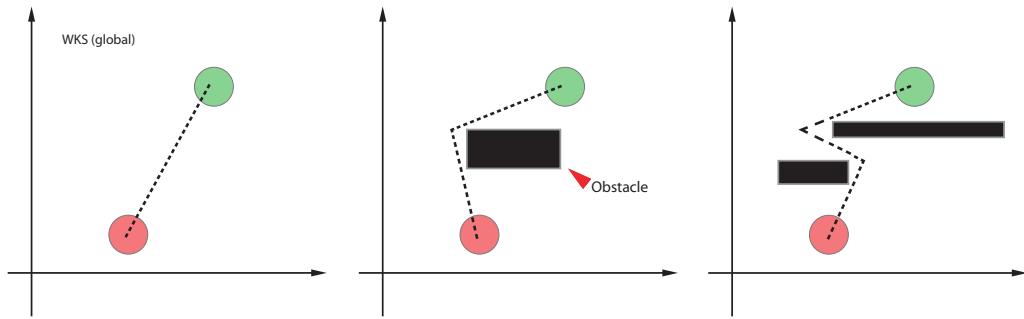


Abbildung 5.1: Umwege der Pfadführung auf Grund von Hindernissen

## 5.1 Festlegung des Pfades

Idealerweise verläuft der Pfad, dem der Roboter von A nach B folgen kann, ohne dass sich Hindernisse dazwischen befinden. In diesem Fall besteht der Pfad aus einer geradlinig verlaufenden Strecke die von Punkt A zu Punkt B führt. Dieser Fall tritt in der Realität wohl eher selten auf. Es ist also davon auszugehen, dass kein Pfad per „Luftlinie“ vorliegt.

Durch unüberwindbare Hindernisse entstehen automatisch auch Umwege. Liegt ein Hindernis zwischen A und B, so muss auch der Pfad dazwischen mindestens einen Knick, wie im mittleren Beispiel des Bildes 5.1, aufweisen.

Somit gilt: Je weniger Streckenabschnitte benötigt werden, umso direkter ist der Weg zur Zielposition. Entsprechend dieses Verhaltens wurde die Lösungsfindung, die hierbei der Anzahl der Abschnitte (Segmente) entspricht, über eine bestimmte geometrische Verteilung erstellt.

Es soll eine höhere Wahrscheinlichkeit vorliegen, um Pfade mit geringerer Anzahl von Abschnitten zu finden. Umgekehrt sollen Routen, die eine hohe Anzahl von Abschnitten benötigen, weniger wahrscheinlich sein.

Folgende Einzelwahrscheinlichkeit der Geometrischen Verteilung (diskret) wurde eingesetzt.

$$P(X = k) = \begin{cases} p(1 - p)^{k-1} & : k \geq 1 \\ 0 & : x < 1 \end{cases} \quad (5.1)$$

Der Erwartungswert

$$E(X) = \frac{1}{p} \quad (5.2)$$

steigt mit kleiner werdendem  $p$ . Das heißt es ist eher zu erwarten, ein Ergebnis zu erhalten, welches eine hohe Anzahl von Abschnitten aufweist. Hierbei darf  $p$  Werte zwischen 0 und 1 annehmen.

Die Varianz lässt sich über folgende Formel bestimmen:

$$V(X) = \frac{1-p}{p^2} \quad (5.3)$$

*Implementierter Algorithmus hierzu ist:*

*Problem der Pfadführung auf Grund von geringer Distanz zu Hindernissen*

Dabei beachtet das Verfahren, so wie es Herms verwendet, Hindernisse nur bedingt. Bei diesem Vorgang ein Pfad für die Mittepunktposition festgelegt, die aus mathematischer Sicht ein Punkt ist und somit auch eine unendlich kleine Fläche besitzt. Beim Anlegen des Pfades wird das Körpervolumen nicht berücksichtigt, genauer gesagt nicht geprüft. Daher kann es vorkommen, dass die Wegführung zu dicht an Hindernisse vorbeigeführt wird. Da in seiner Arbeit der Focus auf der Schrittplanung lag, wurde dies etwas vernachlässigt. Wie Ruffler auch schon beschrieb, kann dies unschöne Effekte innerhalb der Simulation erzeugen, wie beispielsweise ein Durchdringen von Objekten.

## 5.2 Zulässiger Bereich

Für die neue Mittelpunktposition kommt auf Grund ihrer aktuellen Position und Fußstellung nur ein ganz bestimmter Bereich in Frage. Der zulässige Bereich unterliegt also weiteren Einschränkungen, die hier genauer erläutert werden.

---

**Algorithm 1** Algorithmus 9, Zufallszahlen-Erzeugung (geometrisch verteilt)

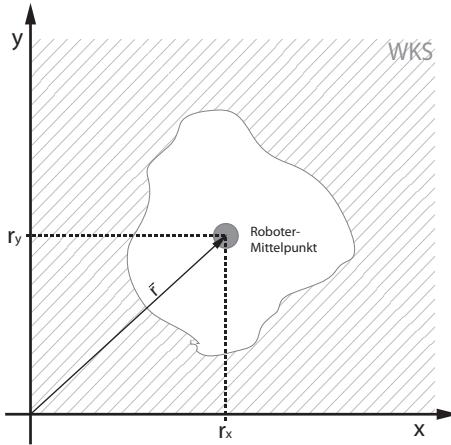
---

```

n := 0;
repeat
    n := n + 1
    choose uniformly distributed random number r ∈ [0,1]
until p < r
return n

```

---



**Abbildung 5.2:** Einschränkung durch Untergrundeigenschaften und Arbeitsbereich der Füße

### 5.2.1 Einschränkung durch Prüfung auf Zulässigkeit

Für die Mittelpunktposition sind nur Bereiche, bei denen keine dieser Eigenschaften zutreffen, zulässig.

- Der Untergrund des Bereichs ist instabil.
- Der Roboter könnte kippen.
- Die Position eines Fußes liegt außerhalb dessen physikalischer Reichweite.
- Es tritt eine Kollision zwischen Beinen auf.
- Es werden zu kurze Zeiten gefordert, denen der Beinregler nicht nachkommen kann.

Wie in Abbildung 5.2 zu sehen, wird der mögliche Mittelpunktpfad bereits durch die Untergrundeigenschaften, sowie dem Arbeitsbereich der aktuellen Fußpositionen, eingeschränkt. Zum besseren Verständnis wurde der Mittelpunkt und der Pfad im übertriebenem Maße dargestellt. In Wirklichkeit ist sowohl der Punkt, als auch die Pfadlinie unendlich schmal. Die Koordinatensysteme der folgenden drei Fälle sind ohne Einheit beschriftet, da die Simulationsumgebung ein einheitenloses Maß verwendet.

### 5.2.2 Einschränkung durch Reichweite der Fußpunkte

Der Pfadbereich wird auch durch die physikalische Reichweite der Fußpunkte eingeschränkt.

Hierfür kommen insgesamt zwei Einschränkungen zum Einsatz:

- Reichweite der Fußpunkte zum Mittelpunkt
- Einhaltung des „minimalen Stability Margin“

Der Mittelpunkt muss demnach innerhalb der „reduzierten“ konvexen Hülle (konvexe Hülle abzüglich minimaler Stability-Margin) liegen und gleichzeitig auch im erreichbaren Bereich der Füße. Dieser Fall ist vereinfacht in Abbildung 5.3 dargestellt.

#### *Änderung in der Ruffler-Version*

An dieser Stelle wurde bei der Version von U. Ruffler eine Änderung durchgeführt, die in Kapitel acht unter Abschnitt „Ereignis: Mittelpunktbewegung“ 8.3.2 erläutert wird.

### 5.3 Einschränkung durch Pfadverlauf

Zu Beginn wurde bereits ein Pfad festgelegt, der auf Grund einer bestimmten Wahrscheinlichkeit in mehrere gerade Stücke unterteilt ist. Nur auf diesem soll sich der

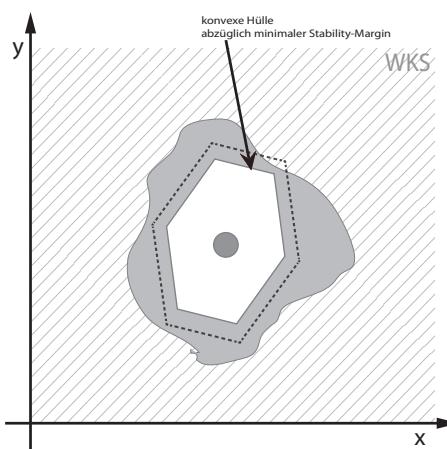
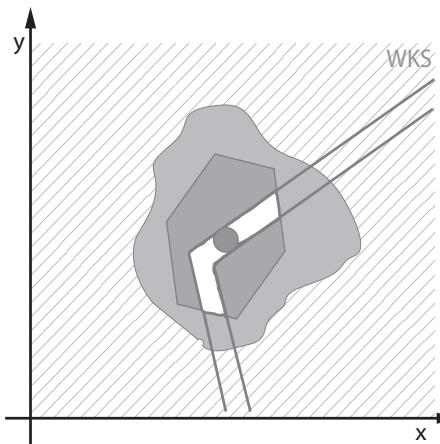
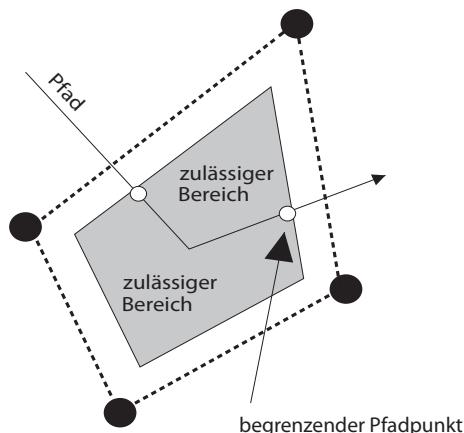


Abbildung 5.3: Einschränkung auf stabilitätserhaltenden Bereich



**Abbildung 5.4:** Einschränkung durch Pfadverlauf



**Abbildung 5.5:** Grenzen des Pfadbereichs

Mittelpunkt bewegen können. Durch die bekannte aktuelle Position und den genannten Kriterien, stellt dies die letzte Einschränkung dar. Der Pfad ist zwar von Start bis Ziel bekannt und auch festgelegt, aber durch die Einschränkungen und dem Pfadverlauf selbst, kommt nur ein ganz bestimmtes Pfad-Intervall auf der Weltkarte in Betracht. Dieser zulässige Bereich ist in Bild 5.4 dargestellt.

### 5.3.1 Ermittlung der Grenzen bei Eindeutigkeit

Das begrenzte Intervall wird über die begrenzenden Pfadpunkte festgelegt. Im Idealfall durchläuft der Pfad wie in 5.5 über eine Eintritts- und eine Austrittsstelle den zulässigen Bereich. Mathematisch sind diese als Schnittpunkte zwischen Pfad und Rand des zulässigen Bereiches definiert.

Ein analytisches Errechnen des zulässigen Bereichs sowie der begrenzenden Punkte ist nicht ohne Weiteres möglich. Herms verwendet aus diesem Grund ein numerisches Suchverfahren, dass durch „Probieren“ überprüft ob ein angegebener Punkt innerhalb des zulässigen Bereiches liegt. Das verwendete Suchverfahren wird als „binäre Suche“ bezeichnet.

Die binäre Suche setzt jedoch zwei bekannte Punkte voraus, welche sich auf dem selben Pfadabschnitt (linear verlaufende Teilstrecke) befinden. Der Pfad wird in der Regel selten auf geradlinigem Wege den zulässigen Bereich durchlaufen, so wie es in Abbildung 5.5 der Fall ist. Der Verlauf ist zwar eindeutig, jedoch sind zwei Pfadabschnitte beteiligt, was durch den Knick bedingt ist.

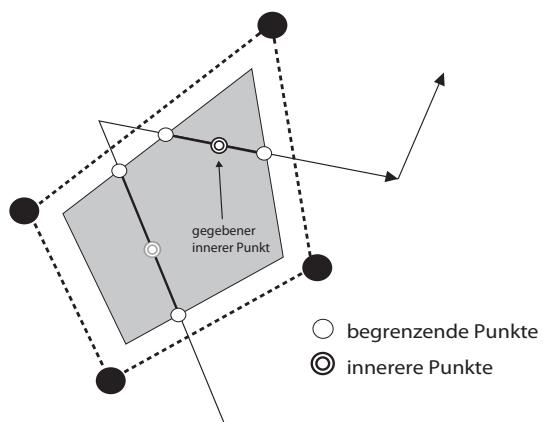
Ohne Weiteres kann die binäre Suche so nicht angewandt werden. Um diese einsetzen zu können, wird die Suche ausgehend von einem vorgegebenen Punkt gestartet, der innerhalb des zulässigen Bereiches liegt. Hierfür bietet sich am einfachsten der Punkt an, welcher sich an der Stelle des Knicks befindet. Ausgehend von diesem Punkt wird in beide Richtungen der Endpunkt beider Stecken gesucht. Die Voraussetzungen für die binäre Suche sind mit dem Finden der Endpunkte erfüllt. Die binäre Suche auf die gleichen Strecken angewendet, liefert dann die exakten Schnittpunkte der Strecken und dem Rand des zulässigen Bereiches.

### 5.3.2 Ermittlung der Grenzen bei Nicht-Eindeutigkeit

Bei einem Pfadverlauf wie er in 5.6 zu sehen ist, kann nicht eindeutig bestimmt werden welches das gesuchte Paar begrenzender Pfadpunkte ist. Durch eine weitere Information eines im Inneren liegenden Punktes, der auch auf dem gesuchten Abschnitt liegt, kann die Zuordnung dann doch erfolgen.

Die gesuchte Richtung des Verlaufs kann über einen Punkt, der im Außenbereich liegt, gewonnen werden. Hierfür kommt dann der Schnittpunkt beider Teilstrecken in Betracht. Dieser befindet sich auch auf der Strecke des bereits durchlaufenen und bekannten Pfadstückes. Indem dieser nun als Startpunkt des gesuchten Streckenabschnitts bekannt ist, kann somit auf dessen weiteren Verlauf geschlossen werden. Der innere Punkt, des bereits durchlaufenen Streckenteils, ist verblassst dargestellt.

Da somit die Grenzen für zulässige Mittelpunkt-Positionen bekannt sind, kann die exakte Festlegung des Mittelpunktes erfolgen. Die Auswahl hierfür geschieht auch wieder zufällig.



**Abbildung 5.6:** Mehrfaches Auftreten von Grenzen des Pfadbereichs

# Kapitel 6

## Planung der Schritte

Es sei an dieser Stelle darauf hingewiesen, dass beim Programmablauf zuerst ein Pfad festgelegt wird. Das in diesem Kapitel beschriebene Optimierungsproblem bezieht sich auf die Schrittplanung, die innerhalb des lokalen Roboter-Koordinaten-Systems, dem RKS stattfindet.

Nachdem die gefundenen Lösungen die Bewertungsfunktion durchlaufen haben, werden diese nochmal auf allgemeine Zulässigkeit geprüft.

### 6.1 Bewertungsfunktion

Bei der Betrachtungsweise als Optimierungsproblems, gibt es immer eine Bewertungsfunktion die manchmal auch als Fitnessfunktion bezeichnet wird. Die Bewertungsfunktion selbst ist aus einzelnen Bewertungsgliedern zusammengesetzt, die nach unterschiedlichen Kriterien  $f$ , mit entsprechender Gewichtung  $\lambda$  bewerten und somit Einfluss auf das Gesamtergebnis nehmen.

$$f_{\text{rating}}(\omega) = \lambda_1 f_1(\omega) + \lambda_2 f_2(\omega) + \dots + \lambda_n f_n(\omega) \quad (6.1)$$

Die Faktoren  $\lambda_i$  ermöglichen zusätzlich eine Gewichtung des bestimmten Kriteriums. Anhand einer solchen Bewertungsfunktion kann jede gefundene Lösung mit einem Zahlenwert beurteilt werden. Selbstverständlich geschieht dies nur anhand der eingebrachten Kriterien.

Herms verwendete in seiner Arbeit [17] die folgenden:

1. Bewegungsdauer
2. Kippstabilität
3. Untergrundstabilität

### *Erweiterung ab der Version Ruffler*

Ruffler erweiterte die Bewertungsfunktion um ein weiteres Glied, dass allerdings negativ in die Gewichtung eingeht, also von der Gesamtbewertung subtrahiert wird. Dieses bewertet das „zielgerichtete Laufen“, dass die ursprüngliche Anforderung „immer das Ziel zu erreichen“ ersetzt.

$$\lambda_{\text{dest}} \cdot \frac{\text{verbleibende Wegstrecke}}{\text{Planungslänge}} \quad (6.2)$$

Die gesamte Bewertungsfunktion sieht ab der Version Ruffler folgendermaßen aus:

$$f_{\text{rating}}(\omega) = \lambda_1 f_1(\omega) + \lambda_2 f_2(\omega) + \dots + \lambda_n f_n(\omega) - \lambda_{\text{dest}} \cdot \frac{\text{verbleibende Wegstrecke}}{\text{Planungslänge}} \quad (6.3)$$

### 6.1.1 Kriterium 1: Bewegungsdauer

Obwohl nicht immer bei allen Bewegungen die Dauer eine Rolle spielt, sollte diese in irgendeiner Weise bewertet werden. Andernfalls würde das Ergebnis einer Bewegung identisch mit dem von „keiner Bewegung“ (Bewegung mit der Dauer 0 s) sein können. Diese Tatsache würde das gesamte Bewertungsergebnis verfälschen. Daher wurde die Dauer  $T_{\text{dest}}$  einer Bewegung als eines der wichtigsten Kriterien eingestuft.

Dies ist auf die gesamte Bewegungsdauer des Mittelpunktes von Startposition bis zur Zielposition bezogen und kann daher durch simple Addition der Dauer einzelner Ereignisse erreicht werden. Doch die gesamte Dauer der summierten Pfadteilstücke hat nur eine bedingte Aussagekraft, da hier die zurückgelegte Wegstrecke ebenso bedeutend ist. Somit wird die Gesamtdauer durch die gesamte zurückgelegte Strecke dividiert. Dieser Zusammenhang wird als Geschwindigkeit bezeichnet und in

der Physik formal mit  $v = \frac{s}{t}$  beschrieben. In dieser Form wäre allerdings die Gesamtdauer proportional zur Durchschnittsgeschwindigkeit und weniger als Kriterium geeignet. Beispielsweise könnte sich der Roboter, ohne Notwendigkeit, hundert Mal um ein Hindernis herumbewegen und die errechnete mittlere Geschwindigkeit ist die Gleiche, wie bei einer einzelnen Bewegung um das Hindernis. Aus diesem Grunde wurde neben der zurückgelegten Strecke auch die direkte Entfernung, von Startposition zur Zielposition, in der Rechnung berücksichtigt.

Der endgültige Term lautet:

$$f_{\text{rating}} = \frac{\overline{P_{\text{Start}} P_{\text{Ziel}}}}{t_{\text{dur}}} = \frac{|\vec{P}_{\text{Ziel}} - \vec{P}_{\text{Start}}|}{t_{\text{dur}}} \quad (6.4)$$

#### *Hinweise zur Methode im Source-Code*

```
MyTime Movement::getFullDuration()
```

#### *Implementierter Algorithmus zum Kriterium 1*

---

##### **Algorithm 2** Berechnung der Gesamtdauer $t_{\text{dur}}$ , [17]

---

```

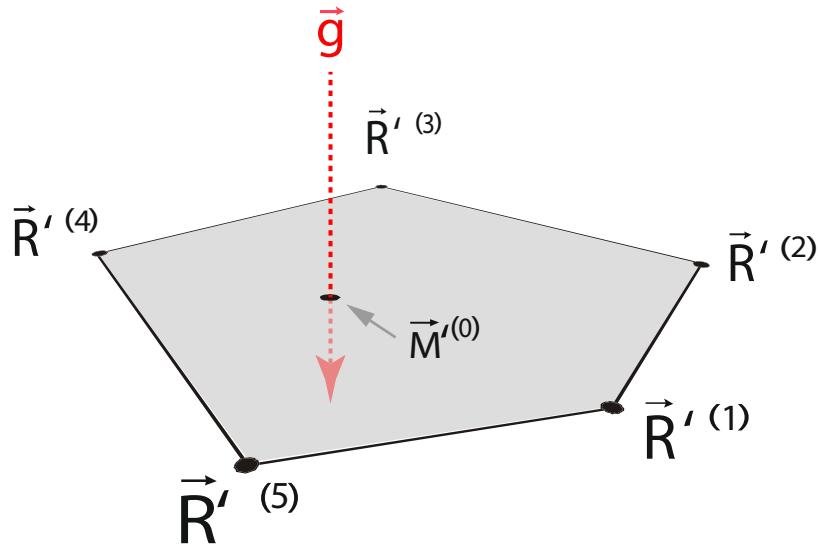
 $t_{\text{dur}} := 0$ 
for all  $e \in \text{CenterEvents}$  do
     $t_{\text{dur}} := t_{\text{dur}} + \text{startTime}(e) + \text{duration}(e)$ 
end for

```

---

### 6.1.2 Kriterium 2: Kippstabilität

Hier wurde die Kippstabilität des gesamten Roboters und auch die Stabilität des Untergrundes, auf dem er sich bewegt, bewertet. Die Beine des Roboters dienen in erster Linie den Funktionen Stützen und Fortbewegen. Die grundlegende Voraussetzung für einen stabilen Stand ist, dass immer drei Füße auf dem Boden stehen und stützen. Stehen allerdings nur die drei Füße einer Seite auf dem Boden, so kippt der Roboter mit Sicherheit. Daher muss einer von drei Füßen auf der anderen Seite als Stützfuß fungieren. Im Idealfall wird der Roboter durch den vorderen und hinteren Fuß einer Seite und dem mittleren auf der gegenüberliegenden Seite gestützt. Auf



**Abbildung 6.1:** Stützpolygon mit Projektion des Roboter-Massenschwerpunktes

diese Weise würde der Roboter die stabilste Konfiguration einnehmen, die mit drei stützenden Füßen möglich ist.

#### Konvexe Hülle

Das größtmögliche Polygon, dass durch Verbinden der äußersten, abstützenden Füße entsteht, nennt man konvexe Hülle. Bildlich kann man sich ein Gummiband vorstellen, welches um die abstützenden Füße gespannt wird. Der Zusammenhang zwischen Roboter-Massenschwerpunkt und konvexer Hülle, ist der in Richtung Gravitationskraft auf den Untergrund projizierte Massenschwerpunkt (in Abbildung 6.2 durch den Schwarz-Weißen Kreis dargestellt). Liegt dieser in dem von stützenden Füßen aufgespannten Bereich, so ist die grundsätzliche Bedingung für stabilen Stand gegeben.

Berechnung der Massenschwerpunkt-Projektion: Sei  $\tilde{M}^0$  die orthogonale Parallelprojektion des Roboter-Massenschwerpunktes  $M^0$ , so muss sich die Projektion  $\tilde{M}^0$ , die entlang des Gravitationsvektors verläuft, innerhalb der konvexen Hülle  $P$  (Stützpolygon) befinden [18]. Abbildung 6.1 zeigt das Stützpolygon mit der Projektion des Massenschwerpunktes.

Um die Projektion  $\vec{M}'^{(0)}$  wie in Abbildung 6.1 zu ermöglichen wird die Projektionsebene  $E_{Proj}$  eingeführt. Der Normalenvektor  $\vec{n}$  der Ebene ist dabei parallel, zum Gravitationsvektor  $\vec{g}$ , ausgerichtet. Die Ebene lässt sich somit beschreiben durch:

$$E_{Proj} : \vec{n} \cdot \vec{R} = d \text{ mit } \vec{n} \parallel \vec{g}; |\vec{g}| = 1; |\vec{n}| = 1; d \geq 0, \quad (6.5)$$

Das Stützpolygon ergibt sich durch die parallele Projektion der Fußpunkte mit Bodenkontakt. Die Projektion des Stützpolygons ergibt sich durch die Projektion der Fußpunkte, über den Ausdruck:

$$\vec{R}'^{(i)} = \vec{R}^{(i)} + (d - \vec{n} \cdot \vec{R}^{(i)}) \cdot \vec{n} \quad (6.6)$$

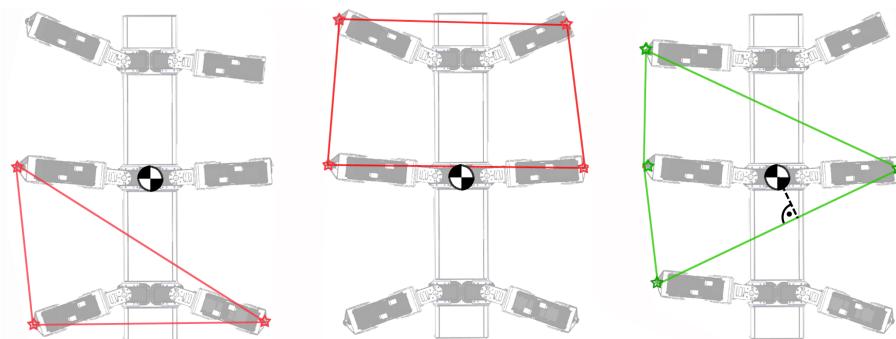
wobei  $\vec{R}'$  die Projektion der Fußpunkte  $\vec{R}$  ist.

Die gesuchte Projektion  $\vec{M}'^{(0)}$  des Roboter-Massenschwerpunktes  $M^{(0)}$  resultiert aus den Zusammenhang:

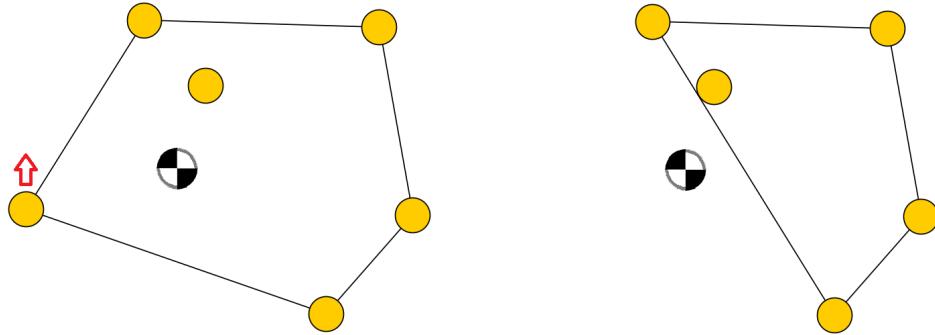
$$\vec{M}'^{(0)} = M^{(0)} + (d - \vec{n} \cdot M^{(0)}) \cdot \vec{n} \quad (6.7)$$

Durch die Anzahl der Beine bedingt gibt es diverse Konstellationen von aufgespannten Dreiecken, darunter auch welche mit grenzwertiger bis zu unzureichender Stabilität. Dies wäre der Fall, wenn die Projektion des Massenschwerpunktes auf der Kante der konvexen Hülle oder außerhalb dieser liegt.

Bei Laufrobotern wird dieser Sachverhalt als Maß für die Stabilität genutzt und als sogenannter „Stability-Margin“ bezeichnet. Genauer handelt es sich um den kleinsten rechtwinkligen Abstand zwischen Projektion des Massenschwerpunktes und dem Rand der konvexen Hülle. Liegt der Massenschwerpunkt schon bei drei



**Abbildung 6.2:** Links: statisch instabil, Mitte: gerade noch instabil, Rechts: statisch stabil



**Abbildung 6.3:** Änderung der konvexen Hülle durch Anheben eines Fußes.  
Folge: Die Projektion des Massenschwerpunktes befindet sich danach im unzulässigen Bereich

stützenden Beinen innerhalb der konvexen Hülle, so kann diese nur durch alleiniges Erhöhen der Anzahl von Stützbeinen nicht kleiner werden. Die konvexe Hülle kann in diesem Fall allenfalls größer werden, dies gilt dann auch für den Stability-Margin.

Anders ist es beim Verringern der Anzahl von stützenden Beinen. Sobald nämlich die Anzahl von vier Stützpunkten auf drei gesenkt wird, kann der Fall eintreten, dass durch das neu entstandene Stützpolygon der Massenschwerpunkt plötzlich außerhalb des stabilen Bereiches liegt. Dies muss in der Steuerung abgefragt und überprüft werden, sodass ein Kippen rechtzeitig verhindert wird. In Abbildung 6.3 wird dieser Sachverhalt nochmals deutlich gemacht. Alle gelben Punkte stellen stützende Füße dar, die den Boden berühren.

Wenn der Roboter so konstruiert wurde, dass die Gewichtsverteilung der inneren und äußeren Komponenten gleichermaßen um den Mittelpunkt verteilt wurde, dann kann in vereinfachter Weise der Mittelpunkt sogar als Massenschwerpunkt angenommen werden. Beim Roboter Akrobat der Hochschule Mannheim wurde eine derartige Konstruktionsweise weitestgehend eingehalten. Genau genommen verschiebt sich der Massenschwerpunkt bereits durch das Bewegen der Beine etwas aus der Mitte. Dieser Effekt wurde in der Arbeit [18] von Thomas Ihme bereits untersucht, jedoch konnte dabei keine signifikante Verschiebung des Massenschwerpunktes beobachtet werden.

Unter normalen Umständen kann es nur zwei Möglichkeiten geben, damit sich der Wert des Stability-Margin verändert:

- Die konvexe Hülle ändert sich.

- Der Massenschwerpunkt bewegt sich.

Statische Stabilität ist theoretisch auch dann gegeben, wenn sich der projizierte Massenschwerpunkt dicht an der inneren Grenze des Stützpolygons befindet. Jedoch ist dies nicht empfehlenswert, denn es würde immer wieder zu Störungen kommen, da sich die Grenzen des Stützpolygons, wie auch der Massenschwerpunkt durch das Laufen selbst, wenn auch nur geringfügig, bewegen. Es muss darum ein Sicherheitsabstand definiert werden, den es für den projizierten Massenschwerpunkt einzuhalten gilt.

Nur so kann sichergestellt werden, dass der Massenschwerpunkt die Grenze der konvexen Hülle nicht überschreitet. Beim Laufplaner wurde ein Mindestabstand zwischen Schwerpunkt-Projektion und dem Rand der konvexen Hülle eingeführt, der einzuhalten ist. Dieser wird als „minimaler Stability-Margin“ bezeichnet. Um für diesen einen guten Wert festlegen zu können, musste zuerst klar sein wann dieser Fall überhaupt eintritt. Der minimale Stability-Margin wird zwischen den Ereignissen angepasst und kann darum nicht direkt bewertet werden.

Herms ist der Auffassung, dass es insgesamt drei Fälle gibt, in denen der „minimale Stability-Margin“ auftreten kann.

- Zu Beginn eines Mittelpunkt-Ereignisses  
(CenterMoveEvent)
- Zum Ende eines Mittelpunkt-Ereignisses  
(CenterMoveEvent)
- Bei Beginn eines Fußpunkt-Ereignisses  
(FootEvent)

*Hinweise zur Methode im Source-Code*

```
float Movement::getStabilityMargin(MyTime t)
```

```
float Movement::getMinStabilityMargin()
```

*Implementierter Algorithmus zum Kriterium 2*

---

**Algorithm 3** Berechnung des minimalen Stability Margin  $\phi_{\text{stab}}$ , [17]

---

```

smargin := stabilityMargin(0)
for all e ∈ CenterEvents do
    t := finishTime(e)
    smargin := min{ smargin, stabilityMargin(t) }
end for
for i from 1 to 6 do
    for all e ∈ FootEventsi do
        t := startTime(e)
        smargin := min{ smargin, stabilityMargin(t) }
    end for
end for

```

---

### 6.1.3 Kriterium 3: Untergrundstabilität

Die Beschaffenheit des Untergrundes, sowie dessen Anstieg oder Gefälle, trägt gleichermaßen zur Stabilität des Systems bei. Prinzipiell ist es sinnvoll auch die Festigkeit des Bodens zu prüfen, um eine Aussage treffen zu können ob sich der geplante Auftrittspunkt überhaupt eignet. Doch das Bewerten und Erfassen dieser Eigenschaften ist äußerst aufwendig und wurde aus diesem Grunde nicht berücksichtigt. Als Vereinfachung wird daher nur die Höhendifferenz (mathematische Steigung) zum geplanten Auftrittspunkt berechnet. Im Grunde entspricht dies der Flächensteigung in einem Punkt  $p$ , welcher sich auf einer der gerasterten Flächen des Terrains befindet. Durch die quadratische Einteilung in die besagten Rasterfelder, ist jeder dieser Punkte von vier Höhenwerten umgeben.

$$U(p) = \left\{ \begin{pmatrix} p_x \\ p_y \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \end{pmatrix} \right\} \quad (6.8)$$

wobei  $u \in U(p)$  erfüllt sein muss.

Die Funktion  $s(p)$  wurde erweitert um diese so in die Bewertung einfließen zu lassen, dass gute Lösungen hoch bewertet werden und schlechte niedrig. Tritt der Fall ein, dass die Steigung so steil ist, dass ein Abrutschen nicht verhindert werden kann, so wird diese Lösung durch  $-M$  vollkommen ausgeschlossen. Die Konstante  $-M$

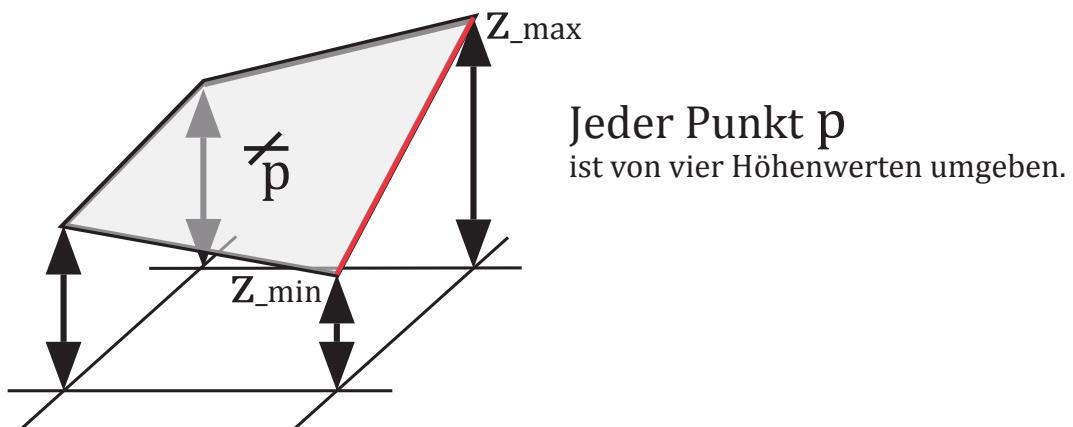


Abbildung 6.4: Steigungen eines Rasterfeldes

ist eine sehr kleine negative Zahl, die dafür sorgt dass die Bewertung niedrig und negativ ausfällt.

$$y = \begin{cases} 0 & s(p) < s_{\min} \\ -s(p) & s_{\min} < s(p) < s_{\max} \\ -M & s_{\max} < s(p) \end{cases} \quad (6.9)$$

Da die Projektion einer Rasterfläche auf den Boden immer quadratisch ist, kann für alle Kanten ein identischer Wert angenommen werden. Alle vier umgebenden Höhenwerte werden verglichen. Der höchste und niedrigste Höhenwert wird zum Ermitteln der größten Steigung des Rasterfeldes verwendet. Wie in Abbildung 6.4 leicht zu erkennen ist, kann es nur drei mögliche Richtungen geben, in denen eine höchste positive oder negative Steigung vorliegt. Diese kann entlang der y-Achse, der x-Achse oder diagonal verlaufen.

#### *Hinweise zur Methode im Source-Code*

Zur Ermittlung der Steigung bestimmter Position:

```
float Movement::getDelta(const Movement::Position& p)
```

Zur Ermittlung der Untergrund-Stabilität:

```
float Movement::getMinGroundStability()
```

*Implementierter Algorithmus zum Kriterium 3*

---

**Algorithm 4** Berechnung der minimalen Untergrundstabilität

---

```
 $\phi_{\text{ground}} := 0$ 
for all e ∈ FootEvents do
    d := destination(e)
     $\phi_{\text{ground}} := \min \{\phi_{\text{ground}}, \sigma(d)\}$ 
end for
```

---

## 6.2 Prüfung auf Zulässigkeit

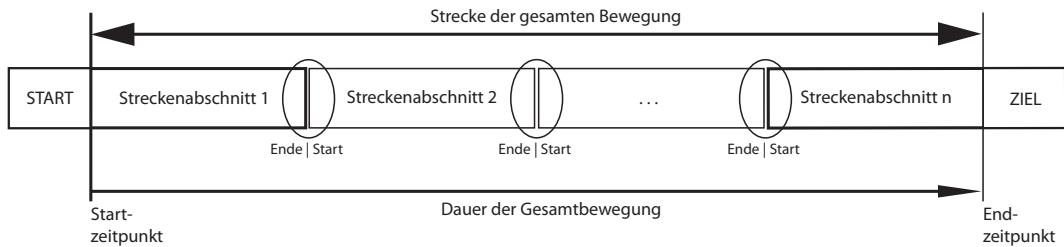
Die Überprüfung auf Zulässigkeit sortiert auch Lösungen aus, welche prinzipiell darstellbar sind, aber nicht die folgenden Bedingungen einhalten.

1. Auch während einer Körperbewegung müssen abstützende Füße im physikalisch möglichen Bereich bleiben.
2. Die angestrebte Zielposition eines Fußes darf nicht außerhalb der physikalischen Reichweite sein.
3. Die geforderte minimale Bewegungsdauer kann vom Bein-Controller nicht erbracht werden.
4. Es darf keine Kollision zwischen Beinen erzeugt werden.

### 6.2.1 Bedingung 1 : Unzulässiger Bereich stützender Füße

Die Position eines abstützenden Fußes wird von der Bewegung des Mittelpunktes abgeleitet. Ab dem Moment, in dem der Fuß aufgesetzt und zum „abstützenden Fuß“ wird, werden dessen Position innerhalb des Weltkoordinaten-Systems an der gleichen Stelle gehalten.

Die zurückgelegte Strecke einer Mittelpunktbewegung entspricht genau der zurückgelegten Strecke, welche vom abstützenden Fuß, nur in die entgegengesetzte Richtung, zurückgelegt wurde. Der Roboter und sein Mittelpunkt werden sozusagen vom Stützfuß in die entgegengesetzte Richtung geschoben, wie in 6.6 (Bedingung 4) durch die Richtungsangabe der Füße und des Mittelpunktes ersichtlich ist. Ein Weg wird natürlich nur zurückgelegt, während der Fuß Bodenkontakt hat, also darf sich dieser nicht in der Schwingphase befinden.



**Abbildung 6.5:** Zeitlicher Verlauf der Ereignis-Kette

Der Startzeitpunkt eines Mittelpunktereignisses ist gleich dem Endzeitpunkt des vorherigen Mittelpunktereignisses, wie in 6.5 gezeigt wird. Die Zeit dazwischen entspricht exakt der Zeit, die zwischen Fuß-Aufsetzen<sup>1</sup> und Fuß-Anheben vergeht. Es ist daher ausreichend, die Zulässigkeit immer zum Startzeitpunkt eines Mittelpunkt- oder Fußpunkt-Ereignisses zu prüfen.

*Hinweise zur Methode im Source-Code*

Reichweite eines Fußes prüfen:

```
bool Movement::isFootInRange(int foot, const MyTime t)
```

### 6.2.2 Bedingung 2: Unzulässiger Bereich für Fuß-Zielposition

Die angestrebte Zielposition des Aufsetzens muss immer auf die physikalisch mögliche Reichweite des jeweiligen Beins geprüft werden. Dies erfolgt durch das Prüfen auf Einhaltung der zulässigen Arbeitsbereiche.

*Hinweise zur Methode im Source-Code*

Relative Fuß-Position ermitteln:

```
Movement::Position Movement::getRelFootPos(int foot, const MyTime t)
```

Zulässiger Bereich aller Füße über die Zeit prüfen:

```
bool Movement::areFeetInRange(MyTime t)
```

---

<sup>1</sup>Herms bezeichnet in seiner Arbeit den Zeitpunkt des Fuß-Aufsetzens als „Fußpunkt-Ereignis“ und der Beginn einer Mittelpunktbewegung entsprechend als „Mittelpunkt-Ereignis“.

### 6.2.3 Bedingung 3: Unzulässige Bewegungsdauer

Die zulässige Dauer einer Körperbewegung oder eines Fußpunkt-Ereignisses ist durch die Datenstruktur nicht eingeschränkt. Das heißt es können auch sehr kleine oder große Zeiten auftreten. Am realen Roboter muss es eine minimale Dauer für Bewegungen geben, weil der dafür verantwortliche Beinregler eine gewisse Zeit benötigt, um die Zielposition zu erreichen. Dies kann beispielsweise beim Lauron eine andere Dauer sein als beim Akrobaten. Beim Akrobaten lässt sich die Geschwindigkeit über den internen Controller der Servomotoren einstellen. Ein Beinregler existiert bei allen drei Robotern in Form von Software. Beim Lauron III wurde ein C167-Regler und bei Lauron IV ein DSP-Regler verwendet. Beim Akrobaten befindet sich der Regler im Controller der Servomotoren. In beiden Fällen gibt es eine minimale Dauer der Bewegungen die zuerst ermittelt werden muss. Bei der Geschwindigkeit der Motoren ist es nicht unbedingt ratsam diese auf höchstmöglicher Geschwindigkeit zu betreiben, da diese auch die höchste Belastung der Motoren verursacht.

*Hinweise zur Methode im Source-Code*

### 6.2.4 Bedingung 4: Unzulässige Bewegung der Beine

Sowohl beim Lauron als auch beim Akrobaten kann die Kinematik der Beine eine Kollision hervorrufen. Solche Kollisionen müssen bereits bei der Planung der Schritte unterbunden werden. Aufgrund der komplexen Kinematik und Trajektorien der Beine ist eine exakte Erkennung von Kollisionen kaum möglich. Darum wird ein vereinfachtes Verfahren eingesetzt, das nach dem Prinzip wie in 6.6 gezeigt funktioniert. In diesem wird eine „unsichtbare“ Begrenzung eingesetzt. Diese verläuft jeweils vom Rand eines Fußes bis zum Thorax (Körper), wobei zwischen Begren-

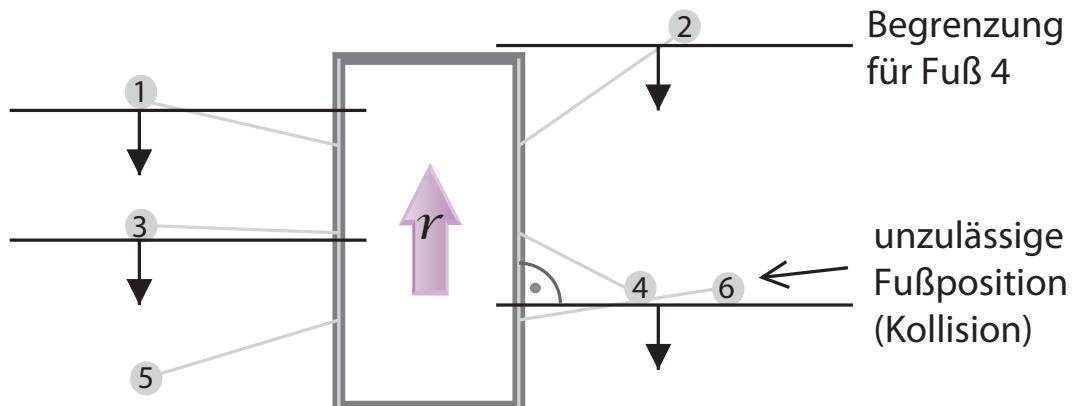
---

#### **Listing 2** Methode zur Prüfung auf die Minimaldauer einer Fußbewegung

---

```
static MyTime minTimeFootMove(  
    bool ground,  
    const SbVec3f& startPos,  
    const SbVec3f& destPos );
```

---



**Abbildung 6.6:** Kollisionsverhinderung durch (imaginäre) Begrenzung

zungslinie und Thorax immer ein  $90^\circ$ -Winkel eingehalten wird. Die Füße haben hierbei einen Radius  $r_{\text{foot}}$  von 2,5 cm zugeteilt bekommen.



# Kapitel 7

## State machine

Unter der Annahme, dass für alle Füße ausreichend viele und zulässige Lösungen gefunden wurden und auch der Verlauf des Mittelpunkt-Pfades feststeht, kann nun mit der Ausführung der Bewegung begonnen werden. Die benötigten Daten sind in der fList und der cList hinterlegt. In diesem Kapitel wird auf die Fußwahl durch die State-Machine (Zustands-Graph) eingegangen. Obwohl das Absetzen und Anheben unmittelbare Folgen davon sind, werden diese „ausführenden Handlungen“ erst im folgenden Kapitel, an Stelle 8 behandelt.

Mit der Aufteilung wie sie in dieser Arbeit angeordnet ist, lassen sich die Kern-elemente des Laufplaners je einem Kapitel (5, 6, 7) zuordnen, wobei Kapitel vier „Eingabeinformationen“ nur einmal zu Beginn durchlaufen wird. Die Anordnung wurde so gewählt, weil der Algorithmus ebenfalls in dieser Reihenfolge arbeitet.

### 7.1 Selektion eines Fußes, der seine Position wechselt

Im vorherigen Kapitel wurde die Ermittlung des zulässigen Bereiches für den Mittelpunkt beschrieben. Um den Mittelpunkt in die nun festgelegte Position zu verschieben, muss ein Fuß angehoben und versetzt werden. Die Auswahl des Fußes, der an der Verschiebung beteiligt ist, wird erst durch drei Bedingungen eingeschränkt und dann zufällig getroffen.

- Kollisionen der Füße müssen vermieden werden.
- Die Füße dürfen nur in deren zulässigen Bereich abgesetzt werden.
- Die Stabilität muss auch beim Positionswechsel der Füße erhalten bleiben.

Um die Stabilität während eines Konfigurationswechsels zu sichern, dürfen nur ganz bestimmte Kombinationen von stützenden Füßen bzw. Beinen auftreten. So muss zum Beispiel immer die Regel eingehalten werden, dass sich mindestens drei Füße im stützenden Zustand befinden. Von diesen dürfen sich nie alle auf der gleichen Seite befinden, da dann lediglich eine Linie zwischen den Stützfüßen aufgespannt wird. Diese kann den Stand des Roboters aber nicht aufrecht erhalten. Es soll immer eine Fläche (konvexe Hülle) vorhanden sein, die durch die stützenden Füße gebildet wird.

Dies wurde über einen gerichteten Graphen realisiert, der in Form einer Transitions-Tabelle nur bestimmte Kombinationen aus angehobenen und abgesetzten Füßen zulässt. Die gesamte Transitionstabelle ist in Abbildung ?? gezeigt.

## 7.2 Zustands-Graph

Der Zustands-Graph ist der essentielle Bestandteil der „State-Machine“. Sei  $G$  ein gerichteter Graph mit der Kantenmenge  $K$  und Knotenmenge  $C$ .

$$G = (C, K) \quad (7.1)$$

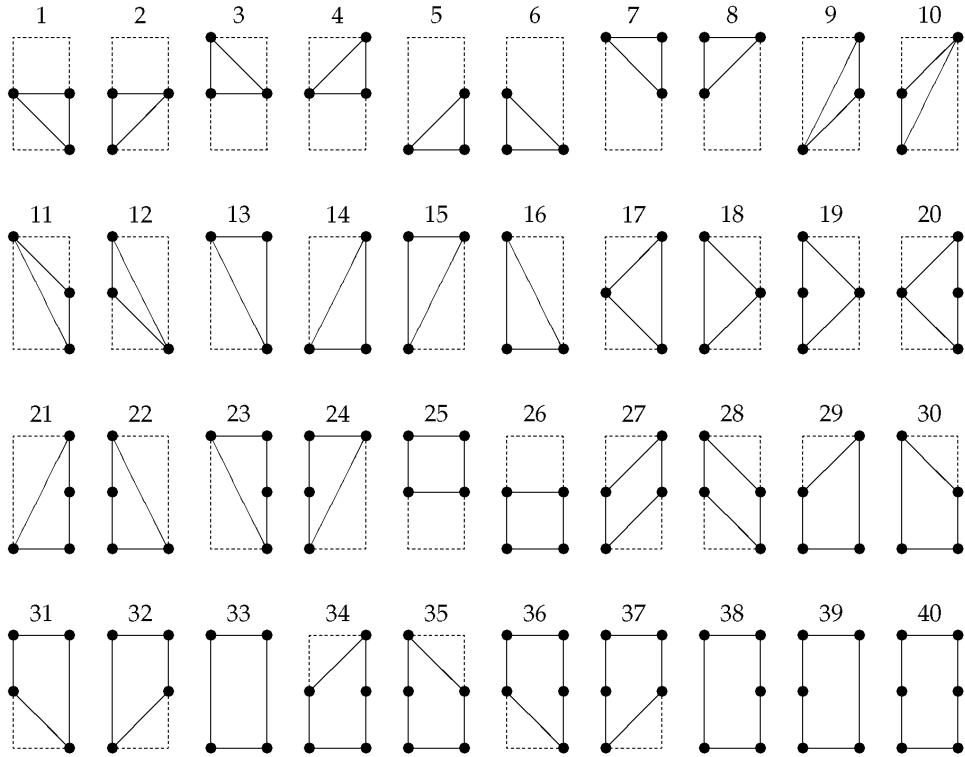
Die Kombinationen aus möglichen Stützzuständen ist durch die Knoten hinterlegt. Die zulässigen Zustände sind also in der Knotenmenge  $C$  enthalten. Eine Zustands-Kombination  $F$  (wie foot) wird aus der Menge der Fuß-Nummern dargestellt, die wiederum eine Teilmenge von  $C$  ist. Somit gilt:

$$F \subseteq \mathbb{N} \cap [1,6] = \{1,2,3,4,5,6\} \quad (7.2)$$

Die Menge aller Knoten ist definiert als:

$$C = c \in 2^{\mathbb{N} \cap [1,6]} \quad (7.3)$$

$$K = \{(c_A, c_B) \in C \times C \mid |(c_A, c_B) \cup (c_B, c_A)| = 1\} \subseteq \mathbb{N} \cup [1,6] = \{1,2,3,4,5,6\} \quad (7.4)$$



**Abbildung 7.1:** Stützzustände mit mindestens drei stützenden Füßen (schwarze Kreise), [17]

In Abbildung 7.1 sind alle Kombinationen hinterlegt, die mit sechs Füßen möglich sind und welche die genannten Bedingungen einhalten, um die Stabilität zu bewahren.

Herms weißt explizit darauf hin, dass beim Lauf des Roboters auch das Heben von zwei Füßen zur gleichen Zeit erforderlich sein kann. Dies sei anscheinend nicht über den Graphen abgedeckt, sei aber indirekt dennoch möglich, indem dem Zwischenschritt (Zwischen-Zustand) eine Dauer von  $t = 0\text{ s}$  zugesprochen wird. Das resultierende Verhalten ist ein Anheben beider Füße zur selben Zeit.

Vermutlich liegt die Ursache dafür, dass solche Wechsel nicht über den Graphen abgedeckt sind, daran, dass für direkte (zweiwertige) Zustandswechsel auch die entsprechenden Einträge in der Transitionstabelle vorhanden sein müssen. Dies ist aber nicht der Fall.

In der Arbeit von A. Herms [17] wurde folgendes Beispiel erläutert:

$$\{1,2,3,4,5,6\} \rightarrow \{1,2,4,5,6\} \rightarrow \{1,2,5,6\} \quad (7.5)$$

$i$	Stützzustand	mögliche Nachfolger	$i$	Stützzustand	mögliche Nachfolger
3	1	2, 20, 26	4	21	9, 14, 5, 38, 34
	2	19, 27, 26		22	12, 16, 6, 39, 35
	3	25, 19, 28		23	7, 13, 11, 36, 38
	4	25, 27, 20		24	8, 15, 10, 37, 39
	5	30, 21, 26		25	8, 7, 3, 4, 37, 36
	6	22, 29, 26		26	2, 1, 6, 5, 35, 34
	7	25, 32, 23		27	4, 10, 9, 2, 37, 34
	8	25, 24, 31		28	3, 12, 11, 1, 36, 35
	9	32, 27, 21		29	10, 17, 14, 6, 39, 34
	10	24, 27, 29		30	18, 11, 16, 5, 38, 35
	11	23, 28, 30		31	8, 13, 12, 17, 36, 39
	12	31, 28, 22		32	7, 15, 18, 9, 37, 38
	13	31, 23, 33		33	15, 13, 16, 14, 39, 38
	14	33, 29, 21		34	27, 20, 29, 21, 26, 40
	15	24, 32, 33		35	19, 28, 22, 30, 26, 40
	16	33, 22, 30		36	25, 31, 23, 28, 20, 40
	17	31, 20, 29		37	25, 24, 32, 19, 27, 40
	18	32, 19, 30		38	32, 23, 33, 30, 21, 40
	19	3, 18, 2, 37, 35		39	24, 31, 33, 22, 29, 40
4	20	4, 17, 1, 36, 34	6	40	37, 36, 39, 38, 35, 34

$$i = \text{Anzahl stützender Füße}$$

**Abbildung 7.2:** Zustands-Kombinationen stützender Füße, [17]

Hierbei wird zuerst Fuß 3 angehoben und anschließend Fuß 4. Der Zustand in dem Fuß 4 angehoben ist, bekommt jedoch die Dauer von 0s zugesprochen. Dadurch ist es möglich Fuß 3 und auch Fuß 4 anzuheben.

In Knoten ausgedrückt entspricht dies  $40 \rightarrow 38 \rightarrow 33$ , wobei Zustand 38 mit der Dauer 0s belegt wurde. In der Transitionstabelle aus Abbildung 7.2 ist beim Zustand 40 allerdings nur der Folgezustand 38 eingetragen. Umgekehrt ist für Zustand 33 auch nicht als Folge der Zustand 40 hinterlegt.

Damit kann ein „zweiwertiger“ Wechsel durch den Graphen gar nicht abgedeckt sein. Dies würde sich auch noch im Nachhinein beheben lassen, indem alle zulässigen Zwei-Fuß Wechsel nachgetragen werden. Prinzipiell wäre sogar ein Drei-Fuß Wechsel denkbar, wobei hier zwingend und immer von Zustand 40 begonnen werden müsste. Aber ein Wechsel von fünf stützenden Füßen auf zwei wäre nicht zulässig, da drei Füße immer in der stützenden Phase sein müssen.

---

**Listing 3** Verschiebung des Lowest-Significant-Bit  
Position der Wertigkeit des Bits entspricht der Fuß-Nummer

```
enum FootBit {  
    FOOT_EL_1 = (1 << 0),           // 00000001 = 1  
    FOOT_EL_2 = (1 << 1),           // 00000010 = 2  
    FOOT_EL_3 = (1 << 2),           // 00000100 = 4  
    FOOT_EL_4 = (1 << 3),           // 00001000 = 8  
    FOOT_EL_5 = (1 << 4),           // 00010000 = 16  
    FOOT_EL_6 = (1 << 5),           // 00100000 = 32  
    FOOT_EL_NEVER = (1 << 6) // 01000000 = 64  
};
```

---

*Hinweis zum Source-Code*

Alle Füße werden durch einzelne Bit's repräsentiert, die innerhalb des enum *FootBit*, wie in Listing 3 zu sehen, erzeugt und abgelegt werden. Vermutlich ist der Hintergrund dafür, dass später am realen Roboter über Taster eine einfache Abfrage der Füße mit Bodenkontakt erfolgen kann. Ein gedrückter Taster entspricht dann einer logischen „0“ und repräsentiert einen stützenden Fuß. Dann wäre es nämlich möglich mit nur einem Port alle Füße auf einmal abzufragen.

Um jeden Fuß anhand eines Bits darzustellen, wird das LSB<sup>1</sup> mit der Wertigkeit „Eins“ auf die entsprechende Position mittels Shift-Operator verschoben.

Wie in Listing 3 ersichtlich ist, muss für den Fuß mit Nummerierung 1 das Bit gar nicht verschoben werden.

Die Bits aller sechs Füße werden in einem char-Array, das als *FOOT\_BIT* bezeichnet ist, abgelegt (siehe Listing 4).

Zusammensetzung einer kompletten Fußkonfiguration:

---

**Listing 4** Zustandsrepräsentation aller sechs Füße durch das FOOT\_BIT

```
static const signed char FOOT_BIT[6] = {  
    FOOT_EL_1,  
    FOOT_EL_2,  
    FOOT_EL_3,  
    FOOT_EL_4,  
    FOOT_EL_5,  
    FOOT_EL_6  
};
```

---

<sup>1</sup>Ist die abgekürzte Bezeichnung für lowest significant bit, welches der niedrigsten Wertigkeit entspricht.

In Listing 5 ist erkennbar, dass für die spätere Weiterverwendung das FOOT\_BIT mittels XOR-Operator von der Bit-Darstellung zu einem Integer-Wert konvertiert wird, wobei dann nur Änderungen ermittelt werden. Damit werden nur die Füße, deren Zustand sich gewechselt hat, erfasst. Der wechselnde Fuße wird in der Variablen „whichFoot“ gespeichert.

---

**Listing 5** XOR-Verknüpfung mit einzelnen Bit-Darstellungen der Füße

---

```
int whichFoot = -1;

for (int foot = 0; foot<6; foot++)
    if ( ( lastFootConf ^ i->footConf ) == FOOT_BIT[foot] ){
        assert(whichFoot == -1);
        whichFoot = foot;
    }
```

---

Die Validierung der Folgezustände wird über folgende Funktion aufgerufen.

```
validFootConf [Stützzustand -1]
```

Da die Stützzustände in einem Array gespeichert sind, wird in der Elementwahl nochmal „1“ abgezogen.

### 7.3 Festlegung der Folge-Zustände

Welcher Fuß mit welcher Aktion als Nachfolger in Betracht kommt, wird alleinig über den Zustandsgraphen ermittelt. Dieser lässt nur Möglichkeiten zu, welche die Kriterien (in 7.2 beschrieben) einhalten. Der Graph ist die *State-Machine*, so lautet die übliche Bezeichnung des Systemteils, der für die Einhaltung bestimmter Zustände (states) sorgt.

In Betracht kommen nur Zustände, welche die Kriterien einhalten, die zu Beginn des Kapitels aufgelistet sind. Die Wahl wird wieder zufällig aus der Menge dieser Zustände getroffen. Herms erwähnt dass genau diese Auswahl für die Qualität der Bewegung verantwortlich ist. Eine gleich verteilte Wahl hätte sehr schlechte Ergebnisse zur Folge.

Die Ursache dafür liegt in der State-Machine. Diese stellt in ihrer Struktur einen Graphen aus Knoten und Kanten dar. Wenn es einen Weg oder genauer eine Kante von Knoten X zu Knoten Y gibt, dann existiert auch umgekehrt der Weg von Knoten Y nach Knoten X. Bei einer gleich verteilten Zufallswahl wird dies mit der

Wahrscheinlichkeit von 1/6 auftreten. Die Bewegung würde sozusagen rückgängig gemacht werden.

#### *Problem der rückläufigen Bewegung*

Statistisch betrachtet sollen alle Füße gleichmäßig oft für stützendes Verhalten eingesetzt werden. Erreicht wurde dies durch eine Verteilung, welche weniger wahrscheinlich die Füße trifft, die vor kurzer Zeit erst ausgewählt wurden. Daher kam der Ansatz zusätzlich zur Zufallsauswahl eine Bonusvergabe durchzuführen. Jeder Fuß besitzt ein „Konto“ für die Bonuspunkte. Bei jedem Durchlauf bekommen genau die Füße einen Bonus vom Wert „1“, die nicht bewegt wurden. Die Füße, die gerade einen Wechsel zugesprochen bekommen haben, bekommen ihr „Bonus-Konto“ auf den Wert „0“ zurückgesetzt.

Die Bonusvergabe ist wie folgt definiert:

$$P(s) = \frac{f_{\text{Bonus}}(b_s)}{\sum f(b_i)} \quad (7.6)$$

wobei sich die Wahrscheinlichkeit eines Fußes, um gewählt zu werden, aus der Einzelbewertung im Verhältnis zur Summe aller Bewertungen ergibt. Ein Auszug des Quellcodes ist weiter unten zu finden unter Listing 6.

Als Bewertungsfunktion wurde die folgende gewählt:

$$f_{\text{Bonus}} = b_i^2 \quad (7.7)$$

Ausgewählt wurde diese, weil sie im erläuterten Kontext gute Werte erbrachte.

An dieser Stelle steht fest welcher Fuß verwendet werden soll. Ob dieser angehoben oder abgesetzt wird, richtet sich nach dem Zustand, den er bislang hatte. Wird der gewählte Fuß abgesetzt, dann erfolgt die Bestimmung seiner neuen Zielposition, wie sie im Abschnitt „Absetzen eines Fußes“ 8.2.1 in Kapitel acht beschrieben wird.

#### *Implementierter Algorithmus zur Fußauswahl:*

**Listing 6** Der zufällig gewählte Fuß wird in der Variablen winner gespeichert.

File: Movement\_Random.cpp

---

```

for (int i = 0; i < 6; i++){
    bonus[i]++;
}

bonus[whichFoot(curFootConf, bestConfSucc)] = 0;
signed char winner = -1;
int r_wheel= (int)( (double)sumBonus + 1.0)* rand() / (RAND_MAX + 1.0));
int cur_wheel = 0;

for (int i = 0; i < numValidSucc; i++){
    if (cur_wheel + fbonus(bonusSucc[i]) >= r_wheel)
    {
        // And the winner is...
        winner = validSucc[i];
        break;
    }
    cur_wheel += fbonus(bonusSucc[i]);
}

```

---

**Algorithm 5** Glücksradauswahl

---

```

sum := 0
for all s ∈ successors do
    sum := sum + f( $b_s$ )
end for
for all s ∈ successors do
     $p_s := f(b_s) / \text{sum}$  { get relative value }
end for
choose uniformly random number  $r \in [0,1]$  { start the wheel}
n := 0
for all s ∈ successors do
    n := n +  $p_s$ 
    if  $n > r$  then
        stop and return s
    end if
end for

```

---

# Kapitel 8

## Ereignisse und Listen

Entsprechend des zufällig gewählten Fußes und dessen vorherigen Zustandes ist nun eine Bewegung von diesem möglich. Hierbei wird zwischen einem Absetzen und einem Anheben unterschieden. In jedem Fall kann sich der zulässige Bereich (reduzierte konvexe Hülle) und somit auch das zulässige Pfad-Intervall verändern.

Es wird in beiden Fällen eine neue Berechnung des zulässigen Intervalls noch vor der Mittelpunktbewegung durchgeführt.

### 8.1 Zusammenspiel der Ereignisse (Bewegungen)

Beispielhaft soll durch Abbildung 8.1 deutlich gemacht werden, wie die Verschiebung des Mittelpunktes und die der Stützbeine zusammenhängen. Als Beispiel wurde ein simples Schrittmuster herangezogen, der „Dreifußgang“. Bei diesem Schrittmuster wechseln immer zwei Gruppen, aus je drei Beinen, abwechselnd zwischen Stütz- und Schwingphase. Dies ist an dem Zyklus  $n$  und dem darauf folgenden  $n + 1$  ersichtlich. Dazwischen wechseln die Fußgruppen ihre Funktion.

Ebenso ersichtlich ist die Gleichheit der Streckendifferenz von stützenden Füßen und der des Mittelpunktes. Die Richtung dieser Bewegungen ist jedoch entgegengesetzt, was anhand der roten Pfeile kenntlich gemacht ist. Die Streckendifferenz von Mittelpunkt zur Zielkoordinate entspricht also der selben Streckendifferenz wie die der Füße, welche gerade für die aktuelle Mittelpunktbewegung verantwortlich ist.

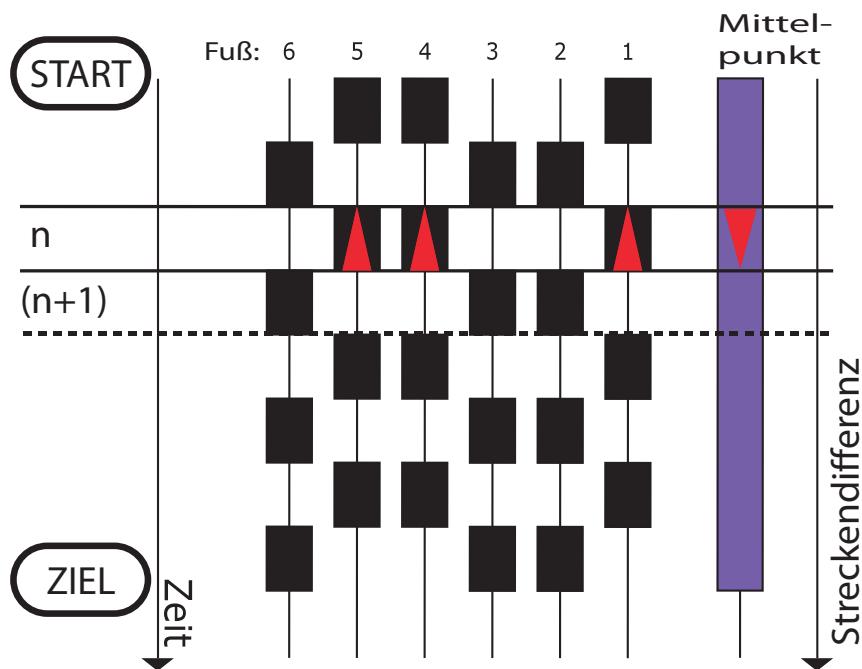


Abbildung 8.1: Zusammenhang zwischen zurückgelegten Strecken

Formal lässt sich dies wie folgt ausdrücken:

$$s_{\text{Stütz}} = -s_{\text{Mittelpunkt}} \quad (8.1)$$

In diesem Fall ist die zurückgelegte Strecke immer auf drei Füße gleichmäßig verteilt, was bei planenden Verfahren sogar eher selten auftritt.

Herms bezeichnet das Umsetzen eines Fußes auf eine neue Position, als ein Fußpunkt-Ereignis. Während diesem befindet sich der Fuß in der Schwingphase. Anschließend gelangt dieser wieder in die Stemmpphase, in der er eine Stützfunktion übernimmt.

## 8.2 Ereignis: Fußbewegung

In diesem Kapitel werden die resultierenden Bewegungen der Füße in Abhängigkeit der aktuellen Mittelpunktposition beschrieben. Die hierbei einzuhaltenden Bedingungen werden ebenso erläutert. Das Absetzen sowie das Anheben stehen in direkt im Zusammenhang zur State-Machine aus dem vorhergehenden Kapitel 7.

### 8.2.1 Aufsetzen eines Fußes

Der Absenkvgang eines Fußes/Beines kann nahezu als unkritisch betrachtet werden, da sich die konvexe Hülle hierdurch vergrößert. Darum muss der zulässige Bereich vor jedem Aufsetzen erneut geprüft werden.

Die exakte Fußposition, auf der abgesetzt werden soll, wird zufällig gewählt. Sie muss auf jeden Fall innerhalb der Reichweite liegen und soll mit größerer Wahrscheinlichkeit auch in Richtung des Ziels liegen. Aus dem Grund wurde hier eine Dreiecksverteilung genutzt, durch welche die Position zwar zufällig bestimmt wird, aber statistisch mehr in Richtung der Zielkoordinaten. Ein solches Verhalten wurde erreicht, indem als Erwartungswert eine Pseudo-Mittelpunktposition eingegeben wurde, die ein kleines Stück mehr in Richtung Ziel versetzt liegt.

Die allgemeine Formulierung einer Dreiecksverteilung lautet:

$$\begin{cases} \frac{(x-a)^2}{(b-a)(c-a)}, & \text{wenn } a \leq x < c \\ \frac{c-a}{b-a}, & \text{wenn } x = c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)}, & \text{wenn } c < x \leq b. \end{cases} \quad (8.2)$$

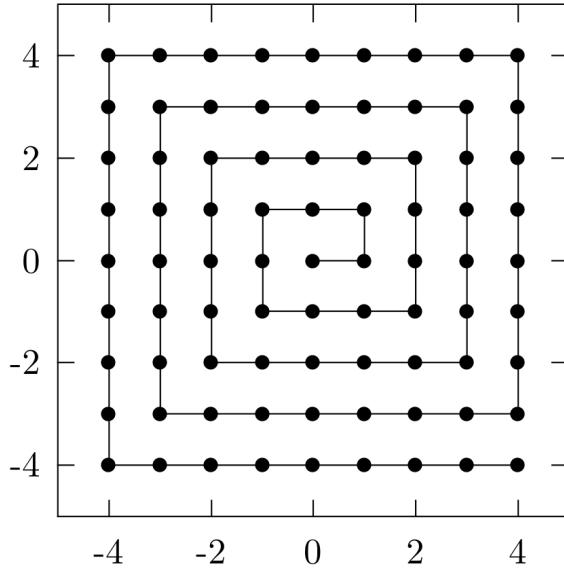
Nachdem eine zufällige Position festgelegt ist, wird diese nochmals auf Zulässigkeit geprüft. Im Erfolgsfall wird diese Position dann als gültige akzeptiert. Lautet das Ergebnis „nicht zulässig“, dann wird über eine Suchspirale<sup>1</sup> der Versuch unternommen doch noch eine naheliegende zulässige Alternativ-Stelle zu finden.

Das Bild 8.2 zeigt die „Suchspirale“, die Herms im Laufplaner verwendet. Ausgehend von der Mitte wird die geprüfte Fläche immer weiter vergrößert. Die erreichbare Endgröße deckt gerade den Arbeitsbereich des Fußes ab. Damit alle Raster-Felder (der Terrain-Map) von der Spirale erfasst werden, wurde der Abstand zwischen den Prüfpunkten kleiner gewählt, als die Auflösung der gerasterten Karte. Nur so ist sichergestellt, dass die Spirale jede Position geprüft hat.

Führt auch die Abtastung mittels Spirale nicht zum Erfolg, so muss die ursprüngliche Position inklusive ihrer nahen Umgebung als Lösung verworfen und eine von Grund auf neue generiert werden.

---

<sup>1</sup>Es gibt verschiedene Muster deren Ausbreitung durch eine mathematische Funktion beschrieben ist. Diese Muster werden oft für Algorithmen genutzt, welche Bereiche abtasten. Deren Radius bzw. Umfang kann durch Variation einer Variablen verändert werden, ebenso wie die Genauigkeit mit der abgetastet werden soll.



**Abbildung 8.2:** Suchspirale zum Finden nahegelegener Alternativ-Positionen, [17]

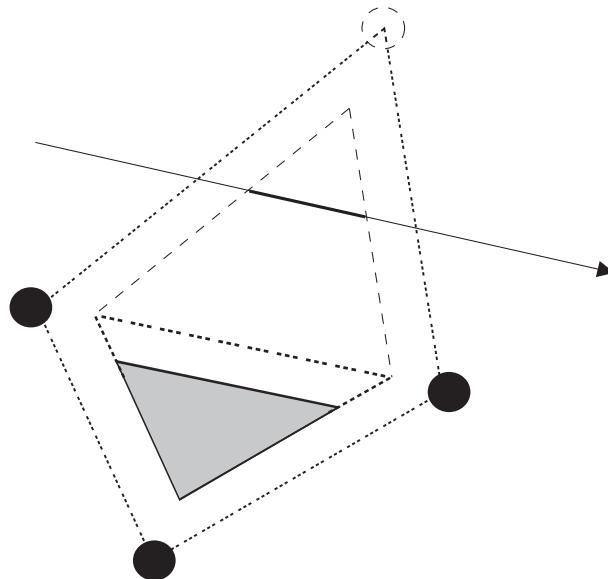
### 8.2.2 Anheben eines Fußes (unzulässig)

Wesentlich kritischer als das Absetzen, ist das Anheben eines Fußes. Hier muss genau geprüft werden, ob sich der Massenschwerpunkt nach dem Anheben immer noch innerhalb des zulässigen Bereichs befindet. Im Bild 8.3 wird ein Fall gezeigt, in dem die Wahl auf einen unzulässigen Fuß fällt. Solche Fälle müssen natürlich sofort untersagt werden. Wäre die Auswahl auf den untersten Stützfuß statt auf den obersten gefallen, so wäre dies wiederum zulässig gewesen. Es dürfen also keine Füße für ein Anheben gewählt werden, bei denen hierdurch der projizierte Massenschwerpunkt nicht mehr im zulässigen Bereich liegt.

In Abbildung 8.3 wird dem Massenschwerpunkt sozusagen der zulässige Bereich genommen, weil dieser schlagartig auf die halbe Fläche reduziert wird.

Um auf diesen Verhalt zu prüfen, werden nur die Endpunkte eines Streckenabschnittes geprüft. Herms begründet die vereinfachte Vorgehensweise mit der erhalten gebliebenen Konvexität des zulässigen Bereichs. So wird nach der Wahl des anzuhebenden Fußes zuerst geprüft ob zwischen Anfangs- und Endpunkt eines Streckenabschnittes überhaupt ein Punkt im entstehenden zulässigen Bereich liegt.

An dieser Stelle sei nochmal darauf hingewiesen, dass es sich hierbei um eine „vor-ausschauende Situation“ handelt, die sich mit einer Hypothese vergleichen lässt, denn zu dieser Zeit wird der Fuß in Realität noch nicht angehoben. Vielmehr wird dies simuliert um die Folgen dann auswerten und gegebenfalls unterbinden zu kön-



**Abbildung 8.3:** Unzulässiges Anheben eines Fußes

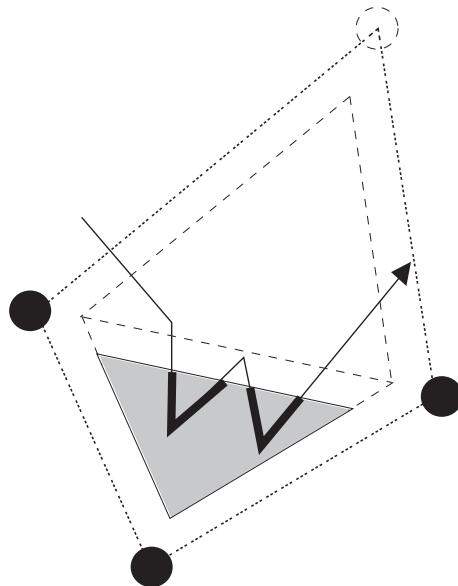
nen. Genauer betrachtet muss die ganze Prozedur abgeschlossen sein, bevor der Programmablauf das reale Anheben veranlasst. Zu bedenken ist hierbei auch, dass im Falle der Unzulässigkeit eine neue Fußwahl als Alternative getroffen wird, welche ebenso überprüft werden muss.

#### *Anheben eines Fußes (Sonderfall)*

Ein Sonderfall wie er im Bild 8.4 dargestellt wird, soll laut Herms nur die absolute Seltenheit sein. Dennoch wird er auch hier kurz erläutert, weil er wenn auch unwahrscheinlich, dennoch auftreten könnte.

Hierbei entsteht ein zulässiger Bereich, dessen Ränder so ungeschickt fallen, dass es Pfadstücke gibt, welche mehrfach zwischen zulässigem und unzulässigem Bereich verlaufen. Prinzipiell gibt es hier auch zulässige Anteile, sonst wäre das Anheben erst gar nicht zulässig gewesen.

Laut Herms werden alle ermittelten Pfadteile, die im zulässigen Bereich liegen, auch vom Algorithmus zurückgegeben. Wie im normalen Fall wird auch hier aus dem ermittelten zulässigen Bereich wieder per Zufallsverteilung bestimmt. Auf Grund des seltenen Auftretens ist der Einfluss auf die Qualität des Endergebnisses vernachlässigbar gering und die Zufallsverteilung wurde nicht extra angepasst.



**Abbildung 8.4:** Sonderfall beim Anheben eines Fußes

Sofern hierbei keine Lösung gefunden wird, so ist das Anheben ausgeschlossen und es erfolgt eine neue zufällige Auswahl.

### 8.3 Ereignis: Mittelpunktbewegung

Die Einnahme der neuen Mittelpunkt-Position ist erst nach Absetzen oder Anheben eines Fußes möglich. Noch vor Verschieben des Mittelpunktes, wird immer nochmal der neue zulässige Pfadbereich ermittelt. Durch diese Bewegung schließt sich der Zyklus und der Ablauf startet erneut.

Für alle Füße sind Fußpositionen vorbereitet, ebenso die folgenden Stützzustände und die begrenzenden Pfadpunkte. Innerhalb des aktuellen Zyklus liegen bereits die Pfad-Intervalle vor. Da auch diese in einer Liste abgelegt werden, sind natürlich auch die zulässigen Intervalle des vorherigen Zyklus in der Liste enthalten. Diese werden für die „genaue“ Festlegung des neuen Mittelpunktes benötigt. Eine stabile Versetzung des Mittelpunktes auf dem Pfad ist nur möglich, weil sich die zulässigen Pfad-Intervalle aus altem und neuem Pfadbereich ein Stück weit überschneiden.

Vor dem vollständigen Erreichen des „neuen“ Stützzustandes ist noch das alte Pfad-Intervall des vorherigen Zyklus gültig. Diese muss auch noch eingehalten werden, wenn der Fuß Kontakt mit dem Untergrund bekommt. In diesem Moment muss er

sich aber auch innerhalb des neuen zulässigen Pfadbereiches befinden. Aus diesem Grund kann nur der gemeinsame Bereich in Frage kommen, auf dem sich das alte zulässige und das neue zulässige Intervall überschneiden.

### 8.3.1 Mittelpunkt-Verschiebung beim Aufsetzen eines Fußes

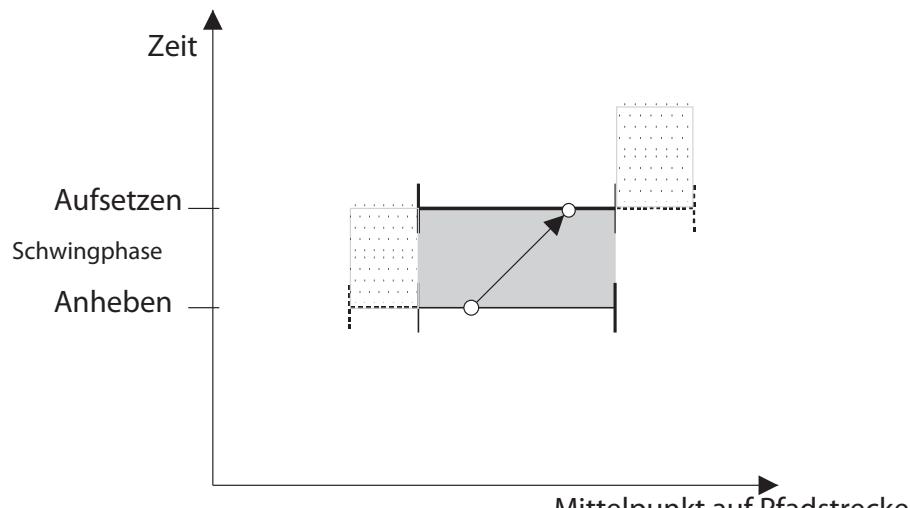
Wird ein Fuß aufgesetzt, dann vergrößert sich das Intervall, das für Mittelpunktpositionen zulässig ist. Verantwortlich dafür ist die konvexe Hülle der stützenden Füße. Die Hülle vergrößert sich in den meisten Fällen wenn ein weiterer Fuß den Stützzustand einnimmt. In welchem Maße die Vergrößerung ausfällt, kann nicht pauschal verallgemeinert werden. Es kommt dabei auf die Gegebenheiten des Untergrundes an.

In Abbildung 8.5, welches aus [17] stammt und ergänzt wurde, ist die Vergrößerung durch die gepunktete Fläche gekennzeichnet. Da in diesem Bild die interessante Stelle der Aufsetz-Zeitpunkt (im Bild als Moment bezeichnet) ist, bezieht sich dies auf die angedeutete Fläche oberhalb des Zeitpunktes des Aufsetzens. Die gepunktete Fläche links-unten soll die Verringerung des Bereichs im Anhebemoment darstellen. Für den hier beschriebenen Fall muss sie nicht unbedingt beachtet werden. Deren Zusammenhang bezieht sich auf den Anheben-Fall und wird im folgenden Unterkapitel erläutert.

Die Darstellung von Abbildung 8.5 und den weiteren Fallbeispielen ist gewöhnungsbedürftig. Der Mittelpunkt verändert beim Wechsel in den neuen Zyklus nicht zwingend seine Bewegungsrichtung, so wie es in Abbildung 8.7 gezeigt ist. Man kann sich die Bereiche überlagert vorstellen als Projektion auf die x-Achse. So ergibt sich aus dem treppenförmigen Gebilde wieder eine Strecke. Auf diese Weise ist zu erkennen, dass es die überschneidenden zulässigen Bereiche sind, die eine stabile Bewegung des Mittelpunktes über mehrere Zyklen hinweg ermöglichen.

### 8.3.2 Mittelpunkt-Verschiebung beim Anheben eines Fußes

Beim Anheben eines Fußes passiert prinzipiell Ähnliches, jedoch wird der zulässige Bereich hierbei kleiner. Dies ist logisch, wenn man bedenkt, dass beim Anheben die konvexe Hülle in der Regel kleiner wird, denn ab diesem Moment ist ein Fuß



Überlagerte Darstellung  
beider Zustände.



Nur im gemeinsamen Intervall kann,  
auch über die Schwingphase hinweg die Stabilität erhalten bleiben.

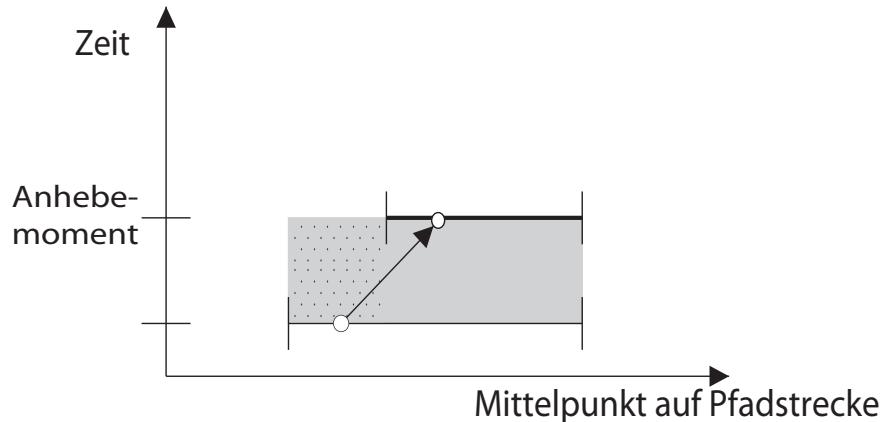
**Abbildung 8.5:** Zulässiger Bereich des Mittelpunktes im Moment des Aufsetzens

weniger am Stützen beteiligt. Größer werden kann die konvexe Hülle keinesfalls. Es ist auch sehr unwahrscheinlich, dass sie ihre Größe beibehält. Auch hier markiert die gepunktete Fläche im Bild 8.6 das Stück Bereich welches mit dem Anheben des Fußes wegfällt.

Ab dem Moment, in welchem der Fuß angehoben wurde, muss der gemeinsame Bereich beider Intervalle (dicke schwarze Linie) eingehalten werden.

### *Problem bei reiner Zufallsauswahl*

Eine rein zufällige Festlegung der neuen Mittelpunkt-Position im gemeinsamen Intervall wäre ungeschickt. Die Wahl würde dann zu oft auf eine Position fallen, die eine Rückwärtsbewegung zur Folge hat. Bei großer Anzahl von Durchlaufzyklen würde diese zwar nicht überwiegen, aber die Folge wäre ein ungewolltes Hin- und Herbewegen. Dieser Fall ist in Abbildung 8.7 zu sehen.

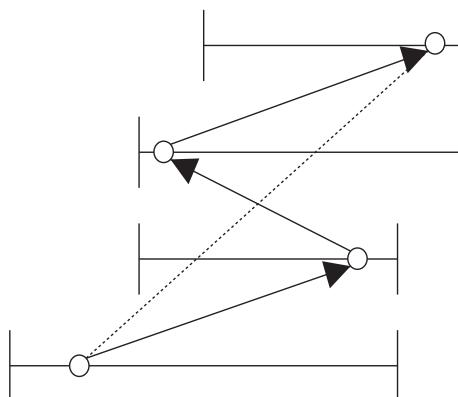


**Abbildung 8.6:** Zulässiger Bereich des Mittelpunktes im Moment des Anhebens [17]

Durch die Vorstellung der Projektion auf die x-Achse kann man auch in diesem Szenario verstehen, dass sich der Körper zwar zum Ziel bewegt, dies aber weniger direkt verläuft.

Bei der Version von A. Herms scheint es so zu sein, dass sich die Wahl der Mittelpunkt-Position neben der Dauer auch auf den minimalen Stability-Margin auswirkt, da dieser immer wieder angepasst wird. Herms erklärt in dem Zusammenhang, dass eine Zufallswahl, welche eine Stelle in Randnähe zum Ergebnis hat, als Folge einen sehr kleinen minimalen Stability-Margin hat.

Wie dies auch immer gemeint ist, es ist in jedem Fall wünschenswert, eine Position zu treffen, die nicht in Nähe der Bereichsränder liegt. Es wurde darum ein Mindestabstand eingeführt, der zwischen Mittelpunkt und Bereichsrand einzuhalten ist. Auf Grund eines Fehlers der beim Konfigurationsübergang mit diesem Verfahren



**Abbildung 8.7:** Verhalten bei Festlegung über reine Zufallsauswahl, [17]

entstehen kann, änderte Ruffler dies in eine Methode wie sie im Folgenden beschrieben wird.

### *Problem bei der Verlagerung des Massenschwerpunktes*

Um die Stabilität, auch während ein Fuß angehoben wird, zu erhalten, wird der Massenschwerpunkt noch vor dem Anheben verlagert. Dafür wird der Roboter bei gleicher Körperposition so weit geneigt, dass der projizierte Massenschwerpunkt auch nach dem Anheben noch auf einer Position liegt, die zulässig ist.

An dieser Stelle hat Ruffler ein Problem festgestellt, denn bislang wurde nicht berücksichtigt, dass der projizierte Massenschwerpunkt gar nicht auf alle beliebigen Stellen versetzt werden kann. Begründet ist dies durch den Arbeitsbereich und der Trajektion der Beine. In Abbildung 8.8 ist beispielhaft eine Bewegung (pinkfarbene Linie mit Anfangs- und Endpunkt) eingezeichnet, mit welcher der Massenschwerpunkt verlagert werden soll. Dies erfolgt mittels der Beine, die sich in der Stützphase befinden. Die schwarze gestrichelte Linie stellt den alten Arbeitsbereich dar, noch bevor er beschnitten wurde. Zu sehen ist, dass die Bewegung zwar gültige Anfangs- und Endpositionen hat, aber dazwischen liegen unerreichbare Positionen. In der Stemmpphase kann dies nicht über die Trajektion der Beine kompensiert werden, da der Fuß während dieser konstant Bodenkontakt hält.

Als Extrembeispiel brachte Ruffler ein Szenario, indem der Roboter alle Beine voll ausgestreckt hat. Hierbei sind alle Arbeitsbereiche der Beine am Anschlag. Der Massenschwerpunkt kann dann gar nicht mehr verlagert werden.

Ruffler löste das Problem, indem er einen Mindestabstand zwischen Körpermitte und dem Arbeitsbereich der Füße definiert hat. Wie in Abbildung 8.8 gezeigt, endet der Arbeitsbereich in Richtung Körper nicht mehr in radialer Form (gestrichelte Linie stellt den ursprünglichen Arbeitsbereich dar), sondern gerade und parallel zur  $x$ -Achse verlaufend. Der Mindestabstand  $y_{ws-dist-min}$  wurde auf mindestens 320 festgelegt. Der Wert entspricht Open-Inventor Koordinaten, die keine Einheit besitzen.

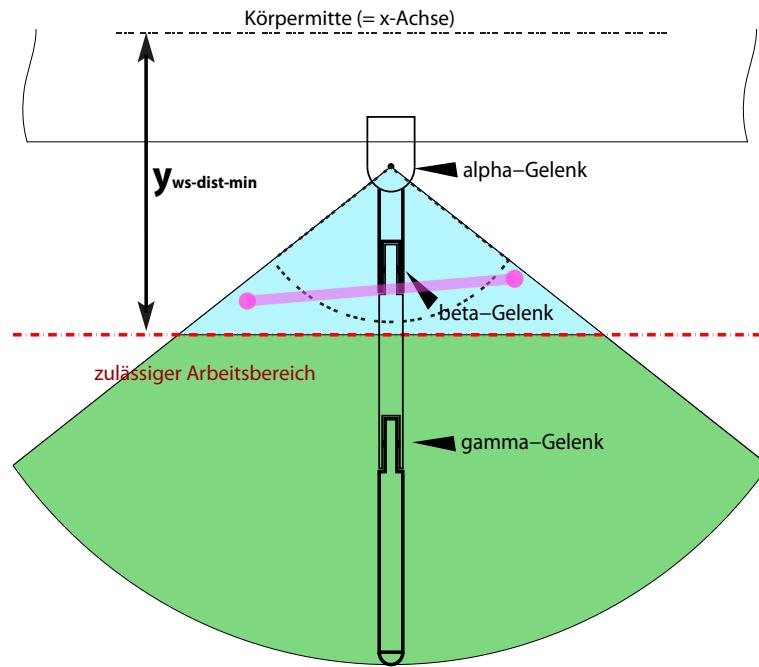


Abbildung 8.8: Mindestabstand zwischen Beinarbeitsbereich und Körpermitte

## 8.4 Listen

Die Definition, die Herms für „Events“ (Ereignisse) festgelegt hat, lautet wie folgt:

Ein Ereignis (Event) beschreibt eine Bewegung des Körpers (Mittelpunkt) oder die eines Fußpunktes. In beiden Fällen besteht ein solches Ereignis aus diesen drei Teilen:

- Dem relativem Startzeitpunkt, ausgehend vom vorherigen Ereignis (diskret).
- Einer Dauer, die für dieses Ereignis vorgesehenen ist.
- Dem Ziel dieser Bewegung.

Basis-Datenstruktur eines einzelnen „Moves“ (Event=Ereignis) ist in Listing 7 zu sehen. Sie ist nicht als eigene Klasse angelegt, sondern als Typ *struct*. Ein „Move“ enthält bereits leere Variablen für die Startzeit, die absolute Startzeit, die Dauer und der Zielposition. Die Ereignisse haben in diskreter Form noch keinerlei Bezug zur Zeit. Lediglich die Machbarkeit der Körperbewegungen und der Fußpositionen kann hierbei erreicht werden (siehe unter ??).

---

**Listing 7** Basis-Datenstruktur eines Events.

File: Movement.h

```
struct Move{
    MyTime          startTime; // sorting key
    Position        dest;
    MyTime          duration;
    MyTime          absStartTime;

    // default constructor
    Move() {}

    // parametrized constructor
    Move(MyTime _start, Position _dest, MyTime _duration)
        : startTime( _start ), dest(_dest), duration( _duration ) {}

    Move(const Move& c) : startTime( c.startTime ),
                           dest( c.dest ),           duration( c.duration )}
```

---

---

**Listing 8** Die fList ist ein Vektor aus FootConfEvents.

File: Movement.h

```
typedef vector<FootConfEvent> FList; //create new datatype FList
FList fList;
```

---

### 8.4.1 Die fList

Die *fList* ist definiert als ein Vektor, dessen Elemente vom Typ *FootConfEvent* sind. Mittels des *typedef*-Befehls wurde dieses Gebilde wiederum als Datentyp mit der Bezeichnung „*FList*“ definiert. Listing 9 zeigt den entsprechenden Teil des Quellcodes. Anschließend wird auch gleich ein Objekt dieses Typs instanziert.

Einzelne Elemente sind also vom Typ „*FootConfEvent*“. Ein einzelnes *FootConfEvent* ist wiederum wie in Listing 9 gezeigt, definiert.

Das Array wird zum Speichern der Foot-Bits (Zustand des Fußes) benötigt. Im Listing 10 wird der *fList* ein neues Element hinzugefügt. Auf Grund der Tatsache, dass im gesamten Quellcode viel mit „*this-Pointern*“ gearbeitet wurde, sind die Funktionsaufrufe sehr lang. Die Darstellung erfolgt daher mehrzeilig.

---

**Listing 9** fList

```
struct FootConfEvent {  
  
    signed char footConf;  
    Position footPos[6];  
  
    float centerStart;  
    float centerEnd;  
    float center;  
  
    FootConfEvent( signed char _footConf,  
                  const Position _footPos[],  
                  float _centerStart,  
                  float _centerEnd ):  
  
        footConf(_footConf),  
        centerStart(_centerStart),  
        centerEnd(_centerEnd) {  
  
            for (int i = 0; i < 6; i++)  
                this->footPos[i] = _footPos[i];  
        }  
};
```

---

---

**Listing 10** Die fList wird um ein Element erweitert.

```
fList.push_back( FootConfEvent(  
    curFootConf,  
    curFootPos,  
    this->seg2Lin( this->startCenterIdx, this->startCenterPoint ),  
    this->seg2Lin( this->endCenterIdx, this->endCenterPoint )  
)  
);
```

---

**Listing 11** Die cList wird zum späteren iterieren der Pfadabschnitte verwendet.

File: Movement.h

```
typedef vector<Movement::Position> CList;
CList cList;

Position startCenterPoint;
Position endCenterPoint;
Position validCenterPoint;

CList::const_iterator startCenterIdx;
CList::const_iterator endCenterIdx;
CList::const_iterator validCenterIdx;
```

---

### 8.4.2 Die cList

Das in Listing 11 gezeigte Konstrukt ist in der Datei Headerdatei (Movement) enthalten, die Definitionen befindet sich in der Datei *Movement\_Random.cpp*. Das Anlegen der cList wurde in Kapitel vier, an Stelle 1 bereits erläutert.

Mit der fList und cList ist eine Bewegung vollständig beschrieben, allerdings nur in diskreter Form. Im nächsten Unterkapitel wird beschrieben wie in eine zeitlich kontinuierliche Form übergegangen wird.

#### *cList in der Version Ruffler*

Ab der Ruffler-Version wird der cList bei jedem Eintrag auch die Zielposition beifügt. Die cList wird ab der Ruffler-Version während des Laufens, nach jedem Strecken-Teilstück erweitert. Die Länge der Teilstücke kann im Source-Code definiert werden. Bei der Herms-Version wurde der Pfad in Segmente unterteilt, doch deren Anzahl und Länge wurde zufällig bestimmt. Ebenso wurde die cList in einem Zug und bereits beim Start des Programmablaufs festgelegt.

## 8.5 Ein Move wird zum Movement

Aus Schritt- und Bewegungsplanung wird das gesamte *Movement* während des Laufens, Stück für Stück aufgebaut. Die zeitlich diskrete Form muss dann verlassen werden, wenn die Bewegung ausgeführt werden soll. Ab diesem Moment müssen die Zeiten zugewiesen sein, sonst kann die Beinregelung auch keine Bewegungen

ausführen, da keine Geschwindigkeiten für diese festgelegt werden können. Aus diesem Grund wird ein „Move“ erst dann zum „Movement“, wenn die Ereignisse die benötigten Zeiten zugewiesen bekommen. Mit dieser Zuweisung kann dann zum Beispiel eine maximale Bewegungsgeschwindigkeit, die von Regler und Aktoren abhängig ist, festgelegt werden. Findet eine Bewegung beispielsweise in der Simulation statt, dann ist die darstellbare Bewegungsgeschwindigkeit von Programm und PC-Komponenten abhängig. In der Version Herms wird dies in einem Stück erledigt, der Bewegungsablauf wird festgelegt noch bevor die eigentliche Bewegung, vom Roboter selbst ausgeführt wird.

---

**Listing 12** Ein Movement besteht aus 7 Move-Vektoren, davon sind 6 für die Füße.  
File: Movement.h

---

```
typedef std::vector<struct Move> CenterList;
CenterList centerMoves;

typedef std::vector<struct Move> FootList;
FootList footMoves[6];
```

---

Dabei besteht das Movement selbst, wie in Listing 12 ersichtlich ist, aus sechs Move-Vektoren für die Füße (FootMove) und einen für Körperbewegungen (CenterMoves). Das „Movement“ wird aus „Moves“ erstellt indem die Zuweisung der Zeiten erfolgt.

---

**Listing 13** Methode um Moves hinzufügen.

---

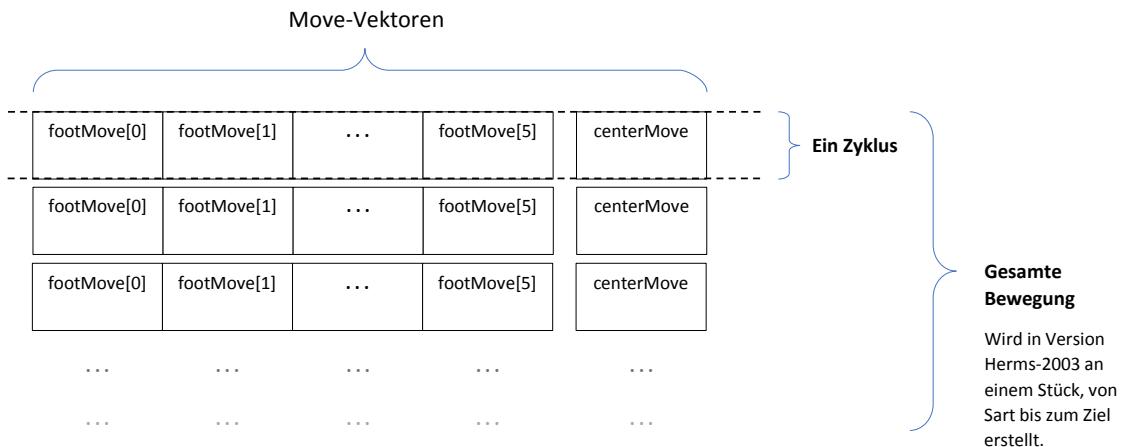
File: Movement\_Random.cpp

---

```
void Movement::addCenterMove(
    MyTime t,           Position dest, MyTime dur
){
    this->centerMoves.push_back( Move( t, dest, dur ) );
}

void Movement::addFootMove(
    int foot, MyTime t, Position dest, MyTime dur
){
    this->footMoves[foot].push_back( Move(t, dest, dur) );
}
```

---



**Abbildung 8.9:** Schema-Darstellung des Movement-Inhalts der Version Herms-2003

### 8.5.1 Movement-Erzeugung in der Version Herms-2003

Da einzelne *Moves* erst dann zum *Movement* werden, wenn die Dauer und Startzeit dafür festgelegt werden. Eine vereinfachte Darstellung eines Movements und dessen Bestandteile, ist für die Herms-2003 Version, in Abbildung 8.9 zu sehen.

In der Version Herms-2003 wird die komplette Bewegung (Movement) erzeugt, welche alle Informationen enthält, um den Roboter von der Startposition zur Zielposition zu bringen. Dies ist nur dann möglich, wenn keine unüberwindbaren Hindernisse die Route stören. Die Bewegung wird dann sozusagen abgespielt und vom Roboter innerhalb der Simulation visuell dargestellt. Möglich ist solches Vorausplanen nur, weil die Höhenkarte die gesamten Gelände-Informationen beinhaltet, die für den Roboter eigentlich noch gar nicht ersichtlich sind. Für Simulationszwecke ist dies in Ordnung, aber in der Realität nur seltenst anwendbar. Im Folgenden wird genauer auf die Zuweisung der Zeiten eingegangen. Die Überführung von Move zum Movement ist von diesen zwei zeitlichen Größen abhängig:

- Von der Mindestdauer der Mittelpunkt-Verschiebung (von alte auf neue Position)
- Von der Mindestdauer der Schwingphase eines Fußes (beim Absetzen die Mindestdauer)

An dieser Stelle sind alle bisherigen Festlegungen der Bewegung als konstant und nicht mehr änderbar anzusehen. Lediglich die Verweildauer kann noch beeinflusst werden.

Mit der Funktion

```
void Movement::construct()
```

wird dann, durch die Ergänzung von Dauer und Startzeit, das vollständige *Movement* erzeugt.

Wie in Listing 14 ersichtlich, wird zuerst die Dauer der Fußbewegung mit einer zeitlichen Toleranz versehen, bevor diese weiterverwendet wird.

Da beide Mindestdauern eingehalten werden müssen und diese auch noch so kurz, wie nur möglich ausfallen sollen, wird das Maximum aus beiden Zeiten ermittelt und dieser Wert als Verweildauer zwischen den Zuständen gewählt.

### 8.5.2 Movement-Erzeugung in Version Ruffler-2006

In der Version Herms-2003 wird die Route noch am Stück erstellt, ebenso wie das Movement selbst. Dies verringert zwar die benötigte Rechenleistung des Programms, verhindert aber auch einen Betrieb im realen Einsatz.

Aus diesem Grund änderte U. Ruffler in [26] den Laufplaner so ab, dass nur noch Movements für Teilstücke der Route erzeugt werden. Dies geschieht auch nicht mehr an einem Stück, sondern findet während des Laufens statt.

Hintergrund dafür ist der Einsatz auf einem echten Roboter, der Umgebungsinformationen eigenständig gewinnt und verarbeitet. Dies muss in Echtzeit während des Laufens erfolgen, da die Umgebung bis auf das Sichtfeld noch unbekannt ist. Somit kann auch kein „Abspielen“ einer vollständigen Bewegung wie in der Herms-2003 Version erfolgen. Die geplanten Movements der Teilstücke müssen dann so lange aneinandergereiht und verkettet werden, bis das Ziel erreicht ist.

Für ein solches Verketten beschreibt Ruffler zwei unterschiedliche Möglichkeiten:

1. Die Möglichkeit fertige Movements der einzelnen Pfad-Abschnitte zu verketten.

---

**Listing 14** Letzter Schritt, Erzeugen des Movements

---

```
void Movement::construct(){
    static const MyTime tolerance(0.1); // for solving numerical problems
    MyTime footTime = Control::minTimeFootMove(
        false,
        upFootPos,
        downFootPos
    )

    footTime += tolerance;

    // deltaT for foot
    MyTime deltaT = footTime;

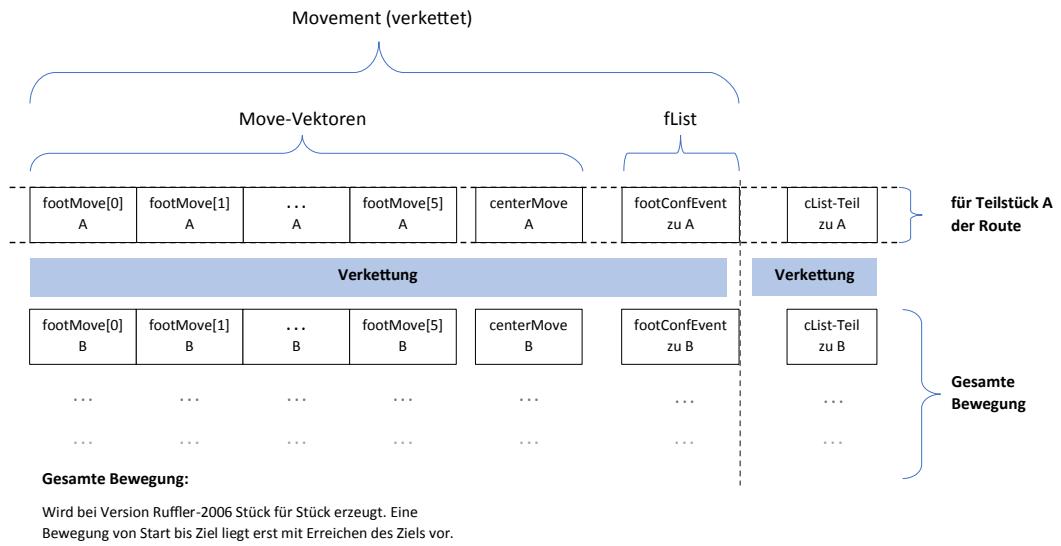
    // if smaller then set deltaT to the maximum
    if (deltaT < curTime + minTime - lastTimeUp[whichFoot]){
        deltaT = curTime + minTime - lastTimeUp[whichFoot];
    }

    this->footMoves[whichFoot].push_back( Move(
        lastTimeUp[whichFoot] - lastTimeDown[whichFoot],
        i->footPos[whichFoot] - newCenterPos,
        deltaT
    )
);
}

// deltaTc for center
MyTime deltaTc = lastTimeUp[whichFoot] + deltaT - curTime;
curTime += deltaTc;

this->centerMoves.push_back(Move(
    MyTime(0.), // start
    newCenterPos,
    deltaTc
)
);
```

---



**Abbildung 8.10:** Verkettung des Movement-Inhalts und der cList, in Version Ruffler-2006

### 2. Die Möglichkeit über FootConfEvents (fList) zu verketten, um dann daraus ein Movement zu erzeugen.

Da die erste Variante mit Nachteilen wie ruckartiges Laufen durch Bewegungspausen behaftet ist, entschied er sich für die zweite Variante. Bei dieser wird die Verkettung, noch vor der eigentlichen Movementerzeugung, über die FootConfEvent-Liste (fList) erreicht.

Eine vereinfachte Darstellung der Movement-Bestandteile ist in Abbildung 8.10 dargestellt.

Um das Movement A mit Movement B zu verketten, wird das abschließende footConfEvent von A mit dem Initial-footConfEvent von B verglichen. Liegt diese als identisches vor, so werden die fListen und die cLists verbunden. Ruffler hat den Code dafür erweitert, wie es in Listing 15 zu sehen ist.

Der zurückgelegte Pfad wurde somit um  $\Delta s_A$  verlängert. Aus diesem Grund müssen die, in der fList von Movement B enthaltenen Informationen (wie Bereiche und Mittelpunkt-Positionen) um die Streckendifferenz  $\Delta s_A$  korrigiert werden.

**Listing 15** Methoden zur Prüfung auf Gleichheit der footConfEvents von Movement Ende-A und Anfang-B.

```
void createRandom(float lengthC = 0.5,
bool initDataAlreadySet = false);
...
MyTime getLastCompleteMove() const; /*RUFFLER*/
MyTime getLastFootEventStartTime() const; /*RUFFLER*/
MyTime getNextFootEventStartTime(MyTime t) const; /*RUFFLER*/

void addEventsFromMovement(Movement *srcMovement); /*RUFFLER*/
void validate(); /*RUFFLER*/
bool isValid(); /*RUFFLER*/
void setNotValid(); /*RUFFLER*/
```

---

## 8.6 Klassen

In diesem Abschnitt werden kurzerhand die wichtigsten Klassen aufgelistet und deren Bezug erläutert, Davon werden einige nur für die Simulationsumgebung und einige nur für den Laufplanungs-Algorithmus verwendet. Leider befinden sich auch Klassen dabei, die von beiden Teilen verwendet werden, was eine Extraktion des reinen Laufplaner-Algorithmus erschwert. Im anschließenden Unterkapitel werden die Klassen aufgelistet, die in der Version Ruffler neu hinzugekommen sind.

### 8.6.1 Herms-Version

Klassen, deren Inhalte überwiegend den Laufplaner betreffen, sind: Terrain, Movement, MovementRandom, MovementNeigh, SimAn

Klassen, deren Inhalt verschiedene Bereiche betreffen und auch die Roboterdimensionierung enthalten, sind: Control, World, Robot, Leg, Head

Klassen, deren Inhalte überwiegend die Simulationsumgebung (Open-Inventor) betreffen, sind: Base, Joint, Screenshot, Eyes, HapticClientInterface<sup>2</sup>, HapticInventor, Trackball, Vispoint, Menu

---

<sup>2</sup>Haptic-Client wurde von Martin Spindler implementiert und ist eine frühe Version der Physik-Engine von Open-Inventor mit der Kraftwirkungen simuliert werden können.

### 8.6.2 Ruffler-Version

GaitPlanner ist die Klasse, durch die der Laufplaner fast echtzeittauglich gemacht wurde, [26]. Die Klasse wurde von Ruffler entwickelt und implementiert und ist in der Herms-Version gar nicht vorhanden. Des Weiteren implementierte Ruffler den Anytime-Algorithmus von Richard Bade [4], dieser arbeitet mit dem Stereo-Vision Verfahren.

Damit sind folgende Klassen hinzugekommen: Util, StereoImage, Matrix, Image, Configure (Headerdatei)

Joel Bout untersuchte in seiner Abschlussarbeit ebenfalls das Stereo-Vision Verfahren. Das Ziel seiner Arbeit war die Erstellung eines Geländeprofils anhand dieses Verfahrens, dass automatisiert auf dem Roboter Lauron betrieben werden kann, um dem Laufplaner die benötigten Umgebungsinformationen zu liefern. Wegen der aufwendigen Arbeiten am Roboter selbst, wurden diese aber nur innerhalb der Simulationsumgebung durchgeführt. Diese wurde dabei nicht verändert, sondern lediglich genutzt, um Stereobild-Ansichten zu generieren die als Basis zu Geländekonstruktion dienten, [7].

#### *Funktion von GaitPlanner*

Durch die Änderung auf die sukzessive Movement-Erzeugung sind Probleme mit einer update()-Funktion entstanden. Diese Funktion dient der Aktualisierung der Darstellung in der Simulationsumgebung und befindet sich eigentlich in der Klasse Control. Doch durch die Änderung kann die update()-Funktion von Control nicht mehr verwendet werden. Auf die genauen Gründe wird hier nicht weiter eingegangen, da diese Arbeit nicht das Ziel verfolgt, die Inventor-Simulationsumgebung zu verwenden. Rufflers Beschreibung nach dient die Klasse aber auch als Ersatz für die fehlerbehaftete (exakte) Mittelpunktbestimmung, die bereits an Stelle 8.3.2 beschrieben wurde. Beim Konstruktorauftruf von GaitPlanner werden zwei Movement-Objekte angelegt. Ein bestes Movement (bestMovement), das scheinbar eine besonders gute Lösung einer Mittelpunktposition beinhaltet und ein weiteres temporäres Movement (tmpMovement) mit welchem aktuell gearbeitet wird. Das bestMove-

ment hat dabei immer die bessere der gefundenen Lösungen parat, natürlich nur sofern auch zwei zulässige Lösungen gefunden wurden.

Falls durch die „mittelmäßige“ Lösung des tmpMovement die Stabilitätsgrenze bei der Mittelpunktfestlegung überschritten wird, dann kommt die „bessere“ Reservelösung des bestMovement zum Einsatz. Dabei wird der temporären Lösung die bessere angehängt. Die bessere Lösung sollte dann so viel besser sein, damit der Massenschwerpunkt, mit dieser noch angehängten Lösung, im zulässigen Bereich liegt. Wurde bislang kein bestMovement gefunden, so muss der Schreitzzyklus unterbrochen und verworfen werden. Ist die Lösung des tmpMovement bereits ausreichend gut, kann diese selbstverständlich verwendet werden und die bessere Reservelösung kommt nicht zum Einsatz.

## Kapitel 9

# Verfahrensvorschlag zur Umgebungserkundung in Echtzeit

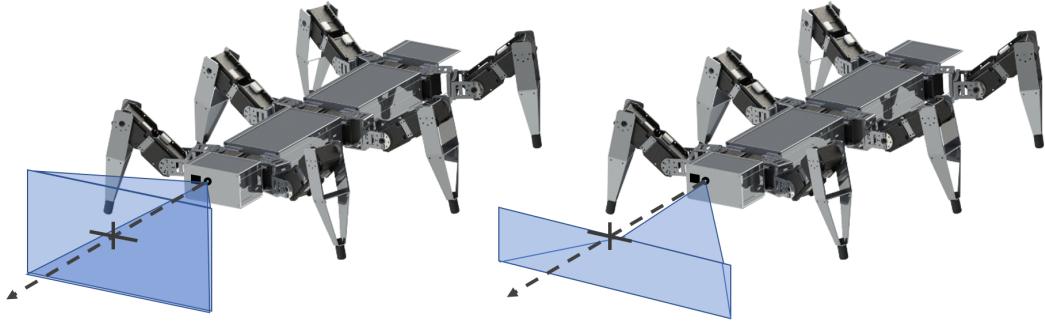
Um dem langfristigen Ziel des autonomen Betriebs entgegen zu kommen, werden in diesem Kapitel alle Vorarbeiten von Uli Ruffler, die eine autonome Betriebsweise begünstigen, aufgeführt. Des Weiteren wird das in Kapitel vier angesprochene Verfahren [11] der ETH-Zürich beschrieben. Zum Vergleich werden einige weitere Echtzeit-Verfahren herangezogen und auf deren Funktionsweise eingegangen. Die Verwendung eines ToF<sup>1</sup>-Sensors bringt deutlich weniger Rechenaufwand mit sich wie die eines Sensors, der auf Basis des Stereovision-Prinzips arbeitet.

Hierbei muss die Tiefen-Information nicht erst durch Berechnung ermittelt werden, sondern wird direkt von der Kamera als Punktfolge bereitgestellt. Die Berechnung findet bei ToF-Verfahren meistens schon auf der Kameraseite statt. Das Erstellen einer Höhenkarte wäre eine der möglichen Datenstrukturen, mit der die gewonnenen Informationen festgehalten werden können. Höhenkarten werden auch als 2,5D-Karten bezeichnet. Sie sind weniger Rechenintensiv als 3D-Karten, bieten dafür aber auch weniger Möglichkeiten. Viele Verfahren zur Kartierung arbeiten mit Höhenkarten. Das vorgestellte Verfahren nutzte diese auch zuvor, wurde jedoch mit der letzten Verbesserung auf einer Kovarianz einer vollständigen 3D-Karte umgestellt, siehe 9.3.

Wie im Bild 9.1 gezeigt wird, kann es vorteilhaft sein, den 3D-Sensor geneigt zu montieren. Auf diese Weise ist eine genauere Erfassung des Untergrundes möglich und weiterhin fallen die Umgebungsinformationen weg, die in Höhen oberhalb des Roboters liegen, denn diese sind in der Regel für die Laufplanung nicht von

---

<sup>1</sup>Ist die Abkürzung für „Time-of-Flight“, ein gängiges Verfahren um 3D-Bilder zu erfassen.



**Abbildung 9.1:** Links: ToF-Sensor vertikal ausgerichtet. Rechts: Mit Neigung ausgerichtet

Relevanz. Ausnahmen sind Merkmale in der Umgebung, die höher als der Roboter selbst liegen und zu seiner Orientierung erforderlich sind. Dies könnte ein bestimmtes Symbol (beispielsweise eine Fahne) sein, das sein Ziel eindeutig signalisiert.

*Aktuell ist der 3D-Sensor ohne Neigung montiert.*

## 9.1 Laufplaner-Vorbereitungen durch Uli Ruffler

An dieser Stelle werden nochmals die Stellen aufgelistet, an welchen Uli Ruffler bereits Änderungen im Laufplaner-Algorithmus von André Herms getroffen hat, um diesen für einen Betrieb mit Echtzeit-Erkundung vorzubereiten.

- Planungsweite des festzulegenden Pfades lässt sich in Stücke beliebiger Länge unterteilen.
- Schrittplanung erfolgt für die Pfadstücke während des Laufens (echtzeitfähig).
- Initial-Stellung der Füße kann nun beliebig sein (Füße müssen nicht alle den Boden berühren).

## 9.2 Ähnliche Verfahren

In [10] wird ein Verfahren mit einer lokalen Karte verwendet, die den Roboter zu jeder Zeit begleitet. Innerhalb dieser werden die gemessenen Umgebungsinformationen während der Bewegung aktualisiert. Bei dem Funktionsprinzip wird die lokale

Höhenkarte aus Stücken zusammengesetzt, die vom Algorithmus als zusammengehörend erkannt werden. Dies wird mit „feature matching“-Algorithmen erreicht, welche die erfassten Umgebungsinformationen auf Gleichheit anhand von Merkmalen prüfen. Anschließend wird die nötige Transformation angewendet, mit der das neue Stück der Umgebung auf die richtige Orientierung gebracht wird um es der lokalen Karte anzufügen. Auf diese Weise wird Stück für Stück die globale Karte aufgebaut. Die Lokalisierung des Roboters wird durch die aufgezeichneten Bewegungen erreicht, indem die zurückgelegte Strecke über die Kinematik berechnet wird. In der genannten Arbeit wird jedoch nicht berücksichtigt, dass die Parameter zur Beschreibung der Körperposition und Orientierung langsam über die Dauer abweicht. Solche Ungenauigkeiten der Roboter-Position und -Orientierung spiegeln sich direkt in der erzeugten Umgebungskarte als ungewollte Höhenunterschiede wieder.

### 9.3 Vorgeschlagenes Verfahren zur Echtzeit-Erkundung

Das Funktionsprinzip beschreibt ein wahrscheinlichkeitstheoretisches, lokales Verfahren zum Erkunden und Kartieren der Umgebung ausgehend von propriozeptiven Sensorinformationen<sup>2</sup>, [16]. Die Entwickler haben sich auf lokale Ungenauigkeiten und deren Entstehung spezialisiert. Das Verfahren [11] erstellt eine fusionierte Umgebungskarte, anhand von zwei Einzelkarten, die von separaten Informationsquellen stammen, siehe Abbildung 9.2. Eine davon wird rein visuell über beispielsweise einen ToF-3D-Sensor erstellt, wobei die andere aus den Bewegungsinformationen des Roboters erzeugt wird. Die meisten Verfahren basieren auf sogenannten 2,5D-Rasterkarten (=Höhenkarten), die weniger Möglichkeiten als 3D-Karten aufweisen, dafür aber weniger Rechenressourcen verbrauchen. Das verbesserte Verfahren [11] arbeitet nicht mehr mit 2,5D-Rasterkarten, sondern mit einer dreidimensionalen Kovarianz einer Karten-Repräsentation.

Die Messwerte, die über die Sensor-Abstandsmessung gewonnen wurden, werden mit geschätzten Tiefenwerten in Bezug gesetzt, wobei die Abweichung der Körperposition jedoch berücksichtigt wird. Das Verfahren bildet dann ein geschätztes Abbild der Umgebung, wobei sich die Schätzwerte nur bis zu einer bestimmten Ge-

---

<sup>2</sup>Die propriozeptive Wahrnehmung wird auch als kinästhetische Wahrnehmung oder als Tiefen-Sensibilität bezeichnet.

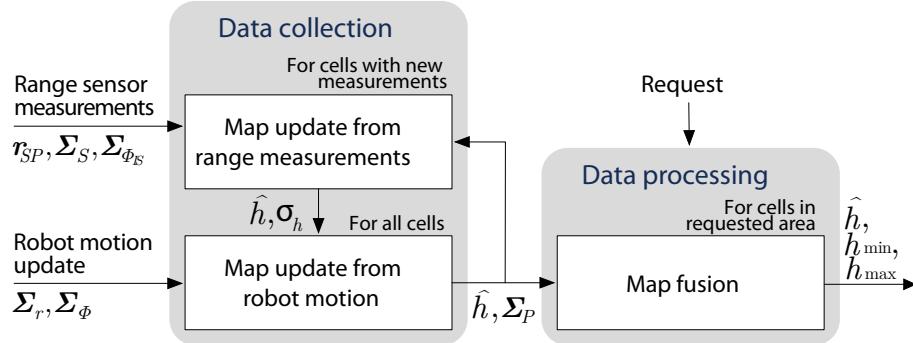


Abbildung 9.2: Funktionsprinzip der fusionierten Umgebungskarte, [11]

nauigkeit festlegen lassen. Beim Erstellen dieses Abbildes sind keine Informationen des Tiefen-Sensors eingeflossen.

Die getrennt erstellten Karten werden dann nach dem Prinzip aus Abbildung 9.3 verbunden. Das Prinzip beruht auf vier unterschiedlichen Koordinatensystemen (im Folgenden als KS abgekürzt), wie in Abbildung 9.4 dargestellt. Die vier Koordinatensysteme sind:

- Inertial-KS (I)
- Robot base-KS (B)
- Sensor-KS (S)
- Map-KS (M)

Das Inertiale-KS ist fest mit der Umgebung (global) verbunden, während das Robot base-KS und das Sensor-KS fest mit dem Körper-Mittelpunkt (lokal) verbunden sind.  $B$  wird über  $\Phi_{IB}$  und  $r_{IB}$  mit  $I$  in Bezug gesetzt und es entsteht der folgende Zusammenhang:

$$\Phi_{IB} = \Phi_{I\tilde{B}}(\psi) \circ \Phi_{\tilde{B}B}(\theta, \phi) \quad (9.1)$$

Rotationen um die z-Achse (Gieren) werden beispielsweise durch  $\Phi_{I\tilde{B}}$  beschrieben. Einzelheiten sind in [11] zu finden.

Letztlich werden die Höhenwerte von  $M$  durch die Beziehung mit einem der lokalen KS ( $B$  oder  $S$ ) definiert. Dabei ist die Beziehung über die Translation  $r_{BM}$  und der Rotation  $\Phi_{IB}$  gegeben, die beide vom Benutzer festgelegt werden.

**Listing 16** Subscribed Topics des ROS-Interfaces, über dass der Algorithmus verfügt.

File: Movement.h

---

```
# Subscribed Topics:  
/points (sensor_msgs/PointCloud2)  
# The distance measurements  
  
/pose (geometry_msgs/PoseWithCovarianceStamped)  
# The robot pose and covariance  
  
/tf (tf/tfMessage)  
# The transformation tree
```

---

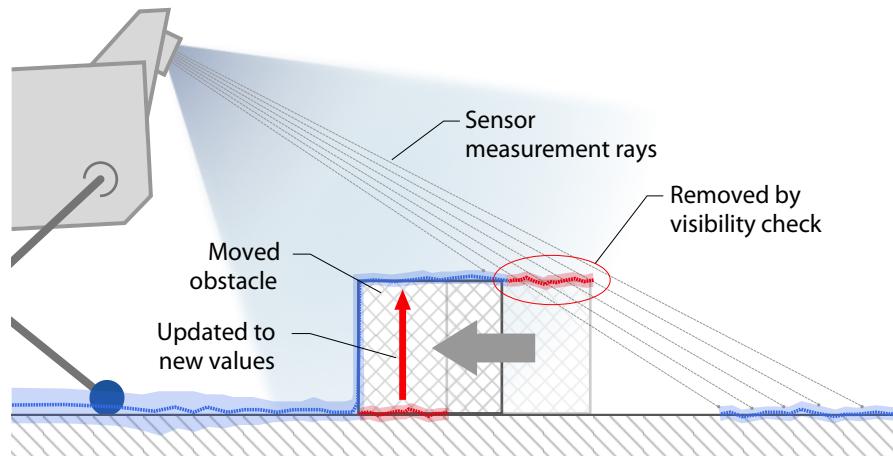
## 9.4 ROS-Kompatibilität

Der in [11] beschriebene Algorithmus ist als C++ Bibliothek geschrieben und besitzt bereits ein ROS-Interface. Er ist als open-source verfügbar und unterstützt die gängigen Hersteller von Sensoren. Es werden einige zusätzliche Bibliotheken und Pakete benötigt wie:

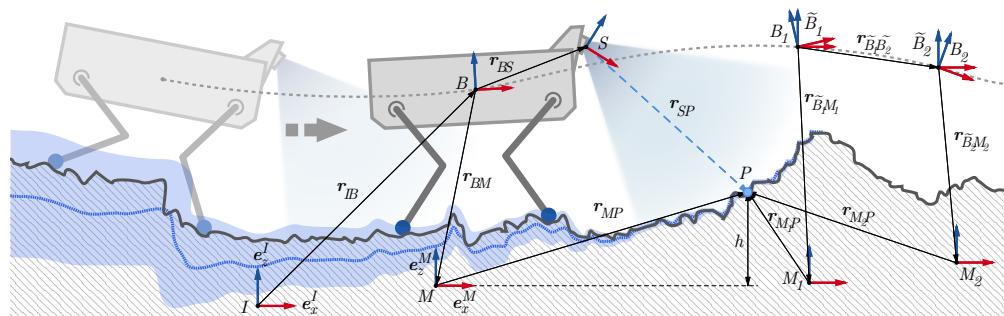
- Grid-Map (library)
- Point-Cloud-Library (PCL)
- kindr
- kindr\_ros (ROS-wrapper)
- ...

Quelle des Projektes: [https://github.com/ANYbotics/elevation\\_mapping](https://github.com/ANYbotics/elevation_mapping)

Das ROS-Interface, kann selbst als Subscriber der folgenden Topics fungieren, um die benötigten Daten zu erhalten:



**Abbildung 9.3:** Funktionsprinzip des Karten-Updatevorgangs, [11]



**Abbildung 9.4:** Zusammenhänge der vier Koordinatensysteme, [11]

# Kapitel 10

## System des Akrobat-Roboters

Um den aktuellen Stand des Akrobat-Laufroboters hinsichtlich der Technik zusammenzufassen, sind in diesem Kapitel die Parameter in Form von Tabellen aufgeführt. Zuerst werden die mechanischen und anschließend die elektronischen Eigenchaften aufgelistet. Die dritte Tabelle enthält Informationen über die aktuellen Betriebssysteme und die vierte Tabelle, die Zugangsinformationen für diese.

### 10.1 Mechanik

Die wichtigsten mechanischen Parameter sind in Tabelle 10.1 zusammengefasst.

### 10.2 Elektronik

Bei der Elektronik gab es diverse Erweiterungen in letzter Zeit, welche sich nicht alle dafür eignen, um tabellarisch aufgeführt zu werden. So wurden beispielsweise die Kabel im Roboterinneren zum Großteil auf Leiterplatten überführt. Die letzten Änderungen, Optimierungen und Messungen wurden in [14] und [23] durchgeführt.

In Tabelle 10.2 sind die wichtigsten elektronischen Parameter zusammengefasst.

Die Abbildung 10.2 zeigt die drei Servomotoren eines Beins. Jedes Bein führt die vieradrige Busverbindung, an der die Servomotoren angeschlossen sind. Die Busleitungen, der sechs Beine verlaufen über Schmelzsicherungen und werden anschließend im Innenraum wieder zu einer gemeinsamen Leitung zusammengeführt.

**Tabelle 10.1:** Mechanische Parameter

<b>Mechanik im Gesamtsystem</b>		
Anzahl Beine	6	Stk.
Lokale Freiheitsgrade (RKS)	18	-
<b>Maße der Gehäusesegmente</b>		
Höhe ca.	56	mm
Breite ca.	102	mm
Länge innere Segmente	je 160	mm
Länge vorderes Segment	etwa 75	mm
Länge hinteres Segment	60	mm
<b>Beine</b>		
Freiheitsgrade pro Bein	3	-
<b>Arbeitsbereich der Gelenke um Z-Achse</b>		
alpha-Gelenk	-50 bis +50	grad
<b>Arbeitsbereich der Gelenke um X-Achse</b>		
beta-Gelenk	-106 bis +106	grad
gamma-Gelenk	-135 bis + 135	grad
<b>Maße der Beinsegmente</b>		
Gesamt (ausgestreckt)	332	mm
A-Segment	72	mm
B-Segment	97	mm
C-Segment	163	mm
Masse eines Beines	ca. 0,8	kg

Die Ansteuerung der Motoren erfolgt über eine RS485 Schnittstelle und zugewiesenen Bezeichnungen, wie sie in Abbildung 10.2 zu sehen sind.

### 10.3 Betriebssysteme

Die Tabelle 10.3 enthält die Bezeichnung und Version von beiden Betriebssystemen, die auf den Boards installiert sind.

**Tabelle 10.2:** Elektronische Parameter**Elektronik im Gesamtsystem****Aktoren**

Servomotoren pro Bein	18	Stk.
Typ	3	Stk.
	Robotis Dynamixel	Rx-64

**Sensoren**

3D-Kamera	PMD picoFlex	ToF
Power-Monitor	SOT23-IC	INA-219

**Recheneinheiten**

Embedded-PC	Raspberry PI	3 B+
Embedded-PC	Intel UP-Board	Version 32GB

**Energieversorgung**

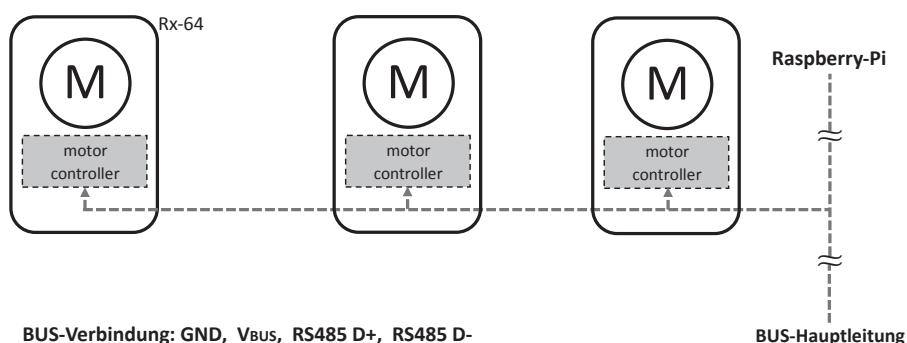
Autark	Lithium-Polymer Akkumulator	2 Stk.
Extern	DC Netzteil	12 V ... 21 V

**Äußere Anschlüsse**

- USB-2.0 (RPi)
- HDMI (RPi)
- Audio 3,5 mm (RPi)

**Interne Anschlüsse**

- I<sup>2</sup>C-Bus
- GPIO-Port (RPi)
- SPI-Bus (RPi)
- RS485/UART (RPi)
- USB2.0 (RPi)
- Ethernet (LAN)

**Abbildung 10.1:** Servomotoren eines Beins

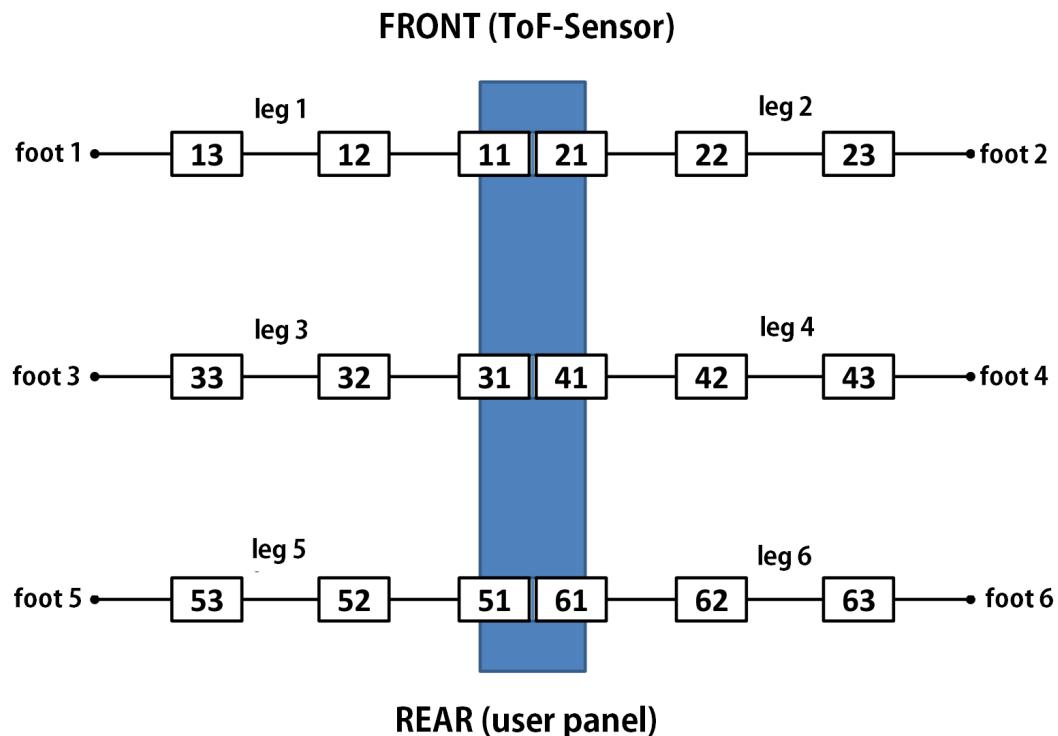


Abbildung 10.2: Bezeichnungen der Servomotoren

Tabelle 10.3: Betriebssysteme

---

### Betriebssysteme im Gesamtsystem

---

#### Raspberry Pi 3B+

OS	Ubuntu(GUI von Lubuntu)	Xenial 16.04
Meta-OS	ROS	Kinetic Kame

#### Intel UP-Board

OS	Ubuntu(GNOME)	Xenial 16.04
Meta-OS	ROS	Kinetic Kame

---

Tabelle 10.4: Zugangsdaten Betriebssysteme

---

### Zugangsdaten der Betriebssysteme

---

#### Raspberry Pi 3B+

user: ubuntu  
login-password: ubuntu

OS

Ubuntu

#### Intel UP-Board

user: robot  
login-password: robot

Ubuntu(GNOME)

login-password: robot

# Kapitel 11

## Akrobot Hardware-Upgrade

### 11.1 System zur 3D-Bild-Erfassung/-Verarbeitung

Im Rahmen dieser Arbeit, wurde der Roboter Akrobot (Front-Gehäusesegment) um ein 3D-Bildverarbeitungssystem erweitert worden. Dieses beinhaltet einen ToF-Sensor, der auf der Innenseite der Aluminium-Abdeckung angebracht ist und einen Embedded-PC, das Intel UP-Board.

#### 11.1.1 3D-Sensor

Die Abbildung 11.1 zeigt den 3D-Sensor, der im Frontsegment des Gehäuses verbaut ist. Dieser arbeitet mit dem ToF-Verfahren (Time-Of-Flight) und ist ressourcenschonender, als ein Sensor auf Basis des Stereo-Vision Prinzips. Der Hersteller des verwendeten Sensors nennt sich PMD, was für „Photonic-Mixing-Device“ (Photonischer Misch Detektor) steht.

Die wichtigsten technischen Eigenschaften sind in Abbildung 11.2 zusammengefasst.



Abbildung 11.1: Time-of-Flight 3D-Kamera, PMD picoFlex

Parameter	CamBoard pico flexx
Dimensions <sup>1)</sup>	68mm x 17mm x 7.35 mm
Weight <sup>1)</sup>	8g
ToF-Sensor	IRS1145C Infineon® REAL3™ 3D Image Sensor IC based on pmd intelligence
Measurement range	0.1 – 4 m
Framerate	Up to 45fps (3D frames); 8 pre-defined operation modes
Acquisition time per frame	4.8 ms typ. @ 45 fps / 30 ms typ. @ 5 fps
Power consumption	USB2.0 compliant, average 300mW for IRS chip and illumination
Illumination	850 nm, VCSEL, Laser Class 1
Software	Royale SDK (C/C++ based, supports Matlab, DotNet, CAPI, OpenCV, OpenNI2, ROS)
Resolution	224 x 171 (38k)px
Viewing angle (H x V)	62° x 45°
Interface	USB2.0 / USB3.0 (data & power)
Depth resolution <sup>2)</sup>	<= 1% of distance (0.5 – 4m @ 5fps) =< 2% of distance (0.1 – 1m @ 45fps)
Operating System	macOS

**Abbildung 11.2:** Technische Spezifikation, 3D-ToF-Sensor PMD PicoFlex

### Front-Abdeckung

Abbildung 11.3 zeigt die angefertigte Front-Abdeckung, die aus Aluminium gefertigt ist. Auf der Innenseite ist der 3D-Sensor montiert. Für diesen wurden in der Abdeckung zwei Öffnungen vorgesehen, über eine davon werden Messpunkte ausgesendet, über die andere diese erfasst und ausgewertet.

#### 11.1.2 Recheneinheit: Intel UP-Board

Das UP-Board, so wie es in Abbildung 11.4 zu sehen ist, basiert auf einer AMD64-Architektur anders als die des Raspberry-Pi, der mit einer ARMv7.1-Architektur arbeitet. Architekturen wie die AMD64 lassen auch die Verwendung von Betriebssystem wie beispielsweise Windows 10 zu. Der Roboter Akrobat wurde in [3] bereits auf das Meta-Betriebssystem/Framework ROS ausgerichtet. Da dieses ausschließ-



**Abbildung 11.3:** Frontblende mit ToF-Sensor

lich innerhalb von Linux funktioniert, wurde auf das Intel UP-Board als Betriebssystem die Linux-Distribution Ubuntu (16.04 Xenial, GNOME) installiert.

Die interne Kommunikation zwischen den Board, den Sensoren und den Aktoren übernimmt das Robot Operation System (kurz ROS) das, in der Version Kinetic-Kame, auf beiden Boards installiert wurde. Hierbei spielt es keine Rolle, dass sich die Architekturen und die exakten Ubuntu-Versionen von UP-Board und Raspberry-Pi unterscheiden. Durch die identischen ROS-Version beider Boards funktioniert die Kommunikation auf ROS-Ebene einwandfrei. Das UP-Board besitzt eine deutlich höhere Leistung als der Raspberry-Pi, wodurch es sich besonders für Bildverarbeitungsprozesse, die viel Rechenleistung erfordern, eignet. Mit der stärkeren Rechenleistung kann es auch gezielt dazu verwendet werden, um den Raspberry-Pi zu entlasten, indem belastende Prozesse auf das UP-Board ausgelagert werden.

Der ToF-Sensor (PMD PicoFlex) ist über ein kurzes Micro-USB Kabel mit dem UP-Board verbunden. Hierfür wurde der USB3.0 Port verwendet, da sich dieser an einer günstigen Stelle auf dem Board befindet. Das Kabel ist *sehr kurz* konfektioniert worden, um den 3D-Sensor, das Kabel und das UP-Board im Inneren des eng bemessenen Front-Gehäusessegmentes unterzubringen. Die LAN-Buchse war durch die zu nahe gelegene Wand des Aluminium-Gehäuses nicht mehr zugänglich, aber für die interne Kommunikation zwingend notwendig.

Aus diesem Grunde wurde das Board an dieser Stelle modifiziert, indem die LAN-Buchse ausgelötet und per Kabel in das nächste Gehäusesegment verlegt wurde. In der Studienarbeit [14] wurde übrigens das Gleiche auch beim Raspberry-Pi, im hintersten Gehäuse-Segment, durchgeführt.

### 11.1.3 Ethernet-Anbindung

Das UP-Board ist über die modifizierte LAN-Buchse an einem Miniatur-Ethernet Switch, der sich im nächstgelegenen Gehäusesegment befindet, angeschlossen. Das Gleiche wurde am Raspberry-Pi, der im hintersten Gehäusesegment liegt, durchgeführt. Auf diese Art wurde eine Netzwerkverbindung zwischen beiden Boards hergestellt.

Um von Extern die Anbindung eines dritten Netzwerk-Teilnehmers (Debugging, etc.) zu ermöglichen, wurde ein Switch mit drei Netzwerkanschlüssen gewählt. Der Miniatur Switch wird vom DC-DC-Wandler, der sich in Front-Richtung befindet, versorgt.

*Bislang wurde die dritte Netzwerkbuchse noch nicht auf die Außenseite des Robotergehäuses geführt.* Es gibt kaum eine Stelle, an der die Buchse positioniert werden könnte, ohne dass sich Einschränkungen der Roboterbewegung ergeben würden.

Als alternative Möglichkeit, um bei geschlossenem Gehäuse dennoch lokalen Netzwerkzugriff von Extern zu ermöglichen, kann auf dem Raspberry-Pi ein WIFI-Access-Point aktiviert und genutzt werden. Eine drahtlose Verbindung kann allerdings nur zum Raspberry-Pi aufgebaut werden, da nur dieser sich hinter einer nicht-metallischen Gehäuse-Abdeckung befindet. An dieser Stelle sei auch erwähnt, dass das UP-Board von Werk ab mit kein WLAN-Interface ausgestattet ist.

### 11.1.4 Verwenden des 3D-Sensors

Grundsätzlich verläuft die Kommunikation mit dem ToF-Sensor über das UP-Board, an dem dieser angeschlossen ist. Folglich muss dieses auch in Betrieb sein, denn nur so kann eine ROS-Verbindung aufgebaut werden, über die der Sensor dann ansprechbar ist.

Um das UP-Board zu aktivieren, bieten sich folgende zwei Möglichkeiten an:

1. Das UP-Board wird immer gleichzeitig mit dem Raspberry-Pi hochgefahren.
2. Das UP-Board wird erst bei Bedarf, vom Raspberry-Pi ausgehend aktiviert.

In der Variante eins kann das Bildverarbeitungssystem, falls es nicht benötigt wird, mit Hilfe eines SSH-Kommandos heruntergefahren oder in einen Power-Save-Mode

---

**Listing 17** Befehl zum lokalen Start des Nodelets

---

```
#lokal auf dem UP-Board starten  
roslaunch pico_flexx_driver pico_flexx_driver.launch publish_tf:=true
```

---

versetzt werden. Das UP-Board ist von Grund auf so eingestellt, dass es beim Anlegen einer Versorgungsspannung automatisch hochfährt. Dies sollte sich in den BIOS-Einstellungen aber auch ändern lassen.

Bei der zweiten Variante kann die Aktivierung zum Beispiel mit der „WakeUp-over-LAN“ Funktion erreicht werden. Die wohl bessere Lösung wäre aber das UP-Board mit einer Kabelverbindung zu aktivieren. Die Verbindung sollte parallel zum Einschalt-Taster des UP-Boards liegen, das andere Ende am GPIO-Port des Raspberry-Pi. Vom GPIO-Port ausgehend wird dann das Signal zum Aktivieren des Bildverarbeitungssystems gegeben.

*Bislang wird die erste Variante verwendet, diese sollte jedoch nicht auf längere Dauer beibehalten werden.*

Der Sensor stellt über das, auf dem UP-Board gestartete PMD-Nodelet als Publisher<sup>1</sup> verschiedene Nodes zur Verfügung. Über Kanäle (Topics) können die Messwerte dann von Subscribers, wie beispielsweise der Raspberry-Pi oder das UP-Board selbst, verwendet werden.

Ein lokaler Start des Nodelets erfolgt mit dem Befehl in Listing 17, zum Beispiel über eine Tastatur. An dieser Stelle sei angemerkt, dass am UP-Board keine USB-Anschlüsse für Eingabegeräte nach außen geführt wurden.

*Um die USB-Ports des UP-Boards zu erreichen, muss dieses aus dem Rahmenprofil des Front-Segments gezogen werden !*

Eine Kommunikation mit dem UP-Board ist nur über ROS vorgesehen. *Der Befehl aus Listing 18 zeigt lediglich eine der Möglichkeiten, um das PMD-Nodelet zu starten.* Ab dem Moment kann die Kommunikation über ROS weitergeführt werden.

Listing 19 zeigt die Nodes und Topics die vom Raspberry-Pi aus erreichbar sind. Aktuell wird das Nodelet über das Kommando aus Listing 18 gestartet. Die Eingabe erfolgt dabei im Terminal auf dem Raspberry-Pi. Dafür wurde „plink“ von „putty-

---

<sup>1</sup>ROS verwendet das sogenannte „Publisher and Subscriber“ Prinzip zur Kommunikation unter den Teilnehmern.

**Listing 18** Single-Line Befehl zum starten des Nodelets per SSH-Kommando vom Raspberry-Pi aus.

```
#global, von RPi aus starten
#ssh login @ up-board with plink (from putty-tools)
plink robot@192.168.178.79 -pw robot &&
    roslaunch pico_flexx_driver pico_flexx_driver.launch publish_tf:=true
```

---

tools“ und ein open-SSH Server installiert. Mit plink kann der SSH-Login samt Passworteingabe und dem ROS-Befehl in einem Befehl ausgedrückt werden.

---

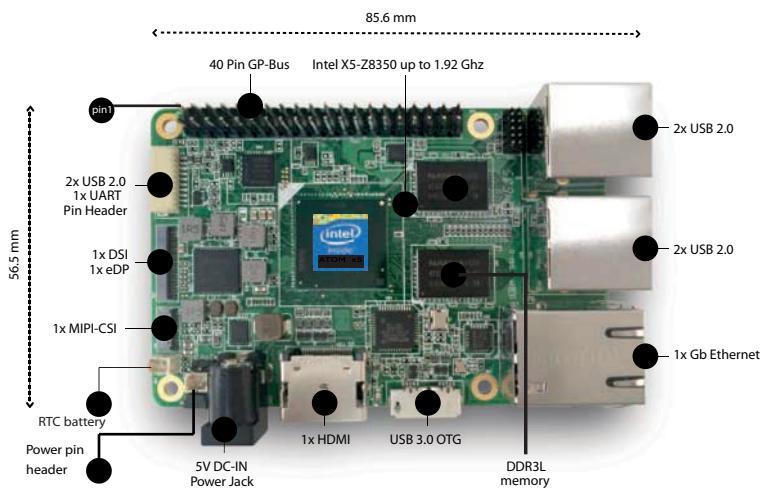
**Listing 19** NODES und TOPICS die über ROS vom PMD-Nodelet zur Verfügung gestellt werden.

---

```
# NODES PMD PicoFlex
ubuntu@masterboard:~$ rosnodes list
/pico_flexx
/pico_flexx_driver
/pico_flexx_static_tf
/rosout

# TOPICS PMD PicoFlex
ubuntu@masterboard:~$ rostopic list
/pico_flexx/bond
/pico_flexx/camera_info
/pico_flexx/image_depth
/pico_flexx/image_mono16
/pico_flexx/image_mono8
/pico_flexx/image_noise
/pico_flexx/points
/pico_flexx/stream2/camera_info
/pico_flexx/stream2/image_depth
/pico_flexx/stream2/image_mono16
/pico_flexx/stream2/image_mono8
/pico_flexx/stream2/image_noise
/pico_flexx/stream2/points
/pico_flexx_driver/parameter_descriptions
/pico_flexx_driver/parameter_updates
/rosout
/rosout_agg
/tf
```

---



**Abbildung 11.4:** Komponenten auf dem Intel UP-Board

## **Kapitel 12**

# **Zusammenfassung und Weiterentwicklungen**

Dieses Kapitel ist das letzte und abschließende und beinhaltet neben einer Zusammenfassung auch Ausblicke für mögliche Weiterentwicklungen.

### **12.1 Zusammenfassung**

Ziel der Arbeit war die Vorbereitung eines Roboters auf die Integration des Laufplaners. Zukünftig soll der eigentliche Laufplaner-Algorithmus auf dem Roboter Akrobot und mit ROS betrieben werden. Hierfür ist zuerst notwendig beide Software Versionen einer Analyse zu unterziehen. Die aktuellere Version des Laufplaners versuchte bereits zu berücksichtigen, dass der vom Roboter erfassbare Bereich eingeschränkt ist. Bei der Modifikation des Laufplaners standen keine echten Sensordaten zur Verfügung, dennoch konnte innerhalb der Simulation ein Anytime-Algorithmus für das Stereo-Vision Verfahren implementiert werden. Die simulierte Umgebung kann zwar mit Texturen versehen werden, dennoch basiert das Geländeprofil auf Höhenkarten.

Mit dieser Arbeit wurde eine vergleichende Analyse zwischen ursprünglicher und modifizierter Laufplaner Version durchgeführt. Modifikationen sind an den betreffenden Stellen kenntlich gemacht und beschrieben worden. Der Aufbau der Analyse spiegelt den Programmablauf des Algorithmus wieder. Beide Software Versionen konnten auf einem virtuellen Betriebssystem lauffähig und kompilierbar gemacht werden. Beim Roboter wurde ein 3D-Sensor integriert um echte Sensordaten, der

Umgebung, zu gewinnen. Die Rechenkapazität des Gesamtsystems wurde erhöht, indem ein zusätzlicher Embedded-PC integriert wurde. Der Sensor kann über diesen, in Verbindung mit einem ROS Nodelet, erreicht werden. Die aktuelle Version des Laufplaners basiert noch immer auf Höhenkarten. In dieser Arbeit wurden beide Laufplaner Versionen nur so weit verändert, bis diese innerhalb des virtuellen Betriebssystems kompilierbar und lauffähig waren.

### 12.2 Weiterentwicklungen

Nach der Analyse des Algorithmus wäre der nächste Schritt die Extraktion von diesem. Der Algorithmus müsste hierfür vollständig von der Inventor-Simulationsumgebung getrennt werden. Hierbei sollte allerdings eine klar definierte, unabhängige und gut durchdachte Schnittstelle angelegt werden. Genau genommen sollten es zwei Schnittstellen sein, davon eine für die Eingabeinformationen und eine weitere für die Ausgabe des Movements. Im Quellcode ist bereits ein XML-Parser eingebaut, mit dem Schrittfolgen aus Dateien gelesen werden können. Ebenso ist die Ausgabe in eine XML-Datei bereits vorhanden. In Hinsicht des Roboters Acrobat muss dann eine Möglichkeit gefunden werden, um die Schritt-Positionen aus dem „Movement“ (XML-Datei) zu lesen und diese über ROS zum Beinregler zu übertragen. Die Steuerung, die Wilen Askerow bereits in [3] implementiert hat, besitzt bereits die notwendigen Transformationen (inverse Kinematik) um Bewegungen der Beine auszuführen.

Ab der Version von Ruffler werden nur noch kleine Teilstrecken des Pfades, mit vordefinierter Länge, vom Algorithmus abgearbeitet. Dies erfolgt während des Lauflens. An dieser Stelle muss die Umgebung, oder zumindest der Untergrund, ausgewertet werden. Im Idealfall wird auf intelligente Art und Weise die Umgebung auf Hindernisse (Obstacles) untersucht und ein Pfad gesucht, der größtmöglichen Abstand zu kritischen Stellen hält. Der dafür notwendige 3D-Sensor wurde in dieser Arbeit bereits zusammen mit einem weiteren Embedded-PC (UP-Board) integriert. Hierbei muss zwischen Lösungsfindung vom Algorithmus und Umgebungserkundung ein Kompromiss hinsichtlich der Geschwindigkeit und Genauigkeit gefunden werden. Das Ziel wäre eine kontinuierliche und flüssige Laufbewegung mit beiläufiger Erkundung der Umgebung in Echtzeit.

Das UP-Board sollte in Zukunft über ein Steuersignal per Kabel und nur nach Bedarf gestartet werden. Dieses kann beispielsweise über den GPIO-Port des Raspberry Pi erfolgen, der in [14] bis in die inneren Gehäusesegmente verlängert wurde. Des Weiteren sollte ein Software-Schalter mittels eines Ubuntu-Services oder auf anderweitige Art eingerichtet werden, um ein automatisches Start-Verhalten an- oder ausschalten zu können. Bei der aktuell verwendeten Methode wird über das Tool „plink“ und SSH-Kommandos der Startbefehl für das PMD-ROS Nodelet erteilt. Der Nodelet-Aufruf, der über den ROS Launch-Befehl erfolgt, setzt einen aktiven Master im lokalen Netzwerk voraus, dies sollte unbedingt berücksichtigt werden.





x

# **Tabellenverzeichnis**

2.1	Körperteile des Stabschreckenbeins . . . . .	11
2.2	Gelenke und Freiheitsgrade . . . . .	11
4.1	Werte der Höhenkarte [17] . . . . .	37
10.1	Mechanische Parameter . . . . .	98
10.2	Elektronische Parameter . . . . .	99
10.3	Betriebssysteme . . . . .	100
10.4	Zugangsdaten Betriebssysteme . . . . .	100



# Abbildungsverzeichnis

2.1	Roboterhund Cheetah3 verfügt nur über taktile Wahrnehmung [5] . . . . .	6
2.2	Chimpansenroboter Charlie, kann auf zwei und vier Füßen laufen [19] . . . . .	7
2.3	Sherpa-UW beim Testeinsatz unter Wasser [19] . . . . .	7
2.4	Schlangenroboter Kairo-3 des FZI-Karlsruhe . . . . .	8
2.5	Laufroboter Lauron-5 des FZI-Karlsruhe . . . . .	8
2.6	Sechsbeiniger autonomer und hydraulisch betriebener Laufroboter Comet IV [24] . . . . .	9
2.7	Stabschrecke ( <i>Carausius Morosus</i> ) . . . . .	9
2.8	Abstraktion zum kinematischen Modell, [28] . . . . .	11
2.9	Schema der Bewegungssteuerung . . . . .	12
2.10	Schematische Darstellung der Bein-Oszillatoren, [1] . . . . .	13
2.11	Merkmale des Schrittzzyklus . . . . .	13
2.12	Grundformen: Arachnid(links), Reptil(mitte), Zirkular(rechts), [15]	15
2.13	Berechnung von Höhenkarten im Stereo-Vision System, [30] . . . . .	19
2.14	Prinzip der Triangulation, [4] . . . . .	20
2.15	3D-Kamera, basierend auf dem Stereo-Vision-Verfahren . . . . .	20
2.16	Funktionsprinzip ToF (Time-Of-Flight), [20] . . . . .	21
3.1	Übersicht zum Programmablauf . . . . .	25
3.2	Die sechs Freiheitsgrade eines starren Körpers innerhalb des Raumes, [13] . . . . .	28
3.3	C-Space bei zwei Freiheitsgraden durch x- und y-Translation . . . . .	29
3.4	C-Space bei drei Freiheitsgraden durch x-, y-Translation und z-Rotation . . . . .	30
3.5	(Obstacle-Based)-Probabilistic-Road-Map [29] . . . . .	31
3.6	Zusammenhang zwischen WKS(global) und RKS(lokal) . . . . .	33
4.1	Terrain-map zur Eingabe der Umgebungsinformation, [17] . . . . .	36
5.1	Umwege der Pfadführung auf Grund von Hindernissen . . . . .	40
5.2	Einschränkung durch Untergrundeigenschaften und Arbeitsbereich der Füße . . . . .	42
5.3	Einschränkung auf stabilitätserhaltenden Bereich . . . . .	43
5.4	Einschränkung durch Pfadverlauf . . . . .	44

5.5	Grenzen des Pfadbereichs . . . . .	44
5.6	Mehrfaches Auftreten von Grenzen des Pfadbereichs . . . . .	46
6.1	Stützpolygon mit Projektion des Roboter-Massenschwerpunktes . .	50
6.2	Links: statisch instabil, Mitte: gerade noch instabil, Rechts: statisch stabil . . . . .	51
6.3	Änderung der konvexen Hülle durch Anheben eines Fußes. Folge: Die Projektion des Massenschwerpunktes befindet sich danach im unzulässigen Bereich . . . . .	52
6.4	Steigungen eines Rasterfeldes . . . . .	55
6.5	Zeitlicher Verlauf der Ereignis-Kette . . . . .	57
6.6	Kollisionsverhinderung durch (imaginäre) Begrenzung . . . . .	59
7.1	Stützzustände mit mindestens drei stützenden Füßen (schwarze Krei- se), [17] . . . . .	63
7.2	Zustands-Kombinationen stützender Füße, [17] . . . . .	64
8.1	Zusammenhang zwischen zurückgelegten Strecken . . . . .	70
8.2	Suchspirale zum Finden nahegelegener Alternativ-Positionen, [17] .	72
8.3	Unzulässiges Anheben eines Fußes . . . . .	73
8.4	Sonderfall beim Anheben eines Fußes . . . . .	74
8.5	Zulässiger Bereich des Mittelpunktes im Moment des Aufsetzens .	76
8.6	Zulässiger Bereich des Mittelpunktes im Moment des Anhebens [17]	77
8.7	Verhalten bei Festlegung über reine Zufallsauswahl, [17] . . . . .	77
8.8	Mindestabstand zwischen Beinarbeitsbereich und Körpermitte . .	79
8.9	Schema-Darstellung des Movement-Inhalts der Version Herms-2003	84
8.10	Verkettung des Movement-Inhalts und der cList, in Version Ruffler- 2006 . . . . .	87
9.1	Links: ToF-Sensor vertikal ausgerichtet. Rechts: Mit Neigung aus- gerichtet . . . . .	92
9.2	Funktionsprinzip der fusionierten Umgebungskarte, [11] . . . . .	94
9.3	Funktionsprinzip des Karten-Updatevorgangs, [11] . . . . .	96
9.4	Zusammenhänge der vier Koordinatensysteme, [11] . . . . .	96
10.1	Servomotoren eines Beins . . . . .	99
10.2	Bezeichnungen der Servomotoren . . . . .	100
11.1	Time-of-Flight 3D-Kamera, PMD picoFlex . . . . .	101
11.2	Technische Spezifikation, 3D-ToF-Sensor PMD PicoFlex . . . . .	102
11.3	Frontblende mit ToF-Sensor . . . . .	103
11.4	Komponenten auf dem Intel UP-Board . . . . .	108

# Literatur

- [1] J. Schmidt A. Büschges. „Neuronale Kontrolle des Laufens – Einblicke aus Untersuchungen an Insekten“. In: *Neuroforum* (2015).
- [2] Nancy Amato, O. Burchan Bayazit und Lucia Dale. *OBPRM: An Obstacle-Based PRM for 3D Workspaces*. 1998.
- [3] Wilen Askerow. „Konstruktion, Aufbau und Inbetriebnahme einer sechsbeinigen Laufmaschine unter Verwendung inverser Kinematik.“ Bachelor Thesis. Hochschule Mannheim, 2014.
- [4] Richard Bade. „Modifikation einer geeigneten Stereobildverarbeitungsmethode für die Anwendung in einer Echtzeitumgebung“. Diploma. Otto-von-Guericke-Universität Magdeburg, 2003.
- [5] Paul Bandelin. „Tastsinn statt Kamera: Roboter-Hund vom MIT ertastet sich den Weg“. In: *modernes mobiles Leben* (2018).
- [6] Karsten Berns. „Steuerungsansätze auf der Basis Neuronaler Netze für sechsbeinige Laufmaschinen“. Hochschulschrift (Dissertation). Forschungszentrum Informatik, Karlsruhe (FZI), 1994.
- [7] Joel Bout. „Erstellung eines Geländeprofils anhand von Stereobildern“. Diploma. Hochschule Mannheim, 2008.
- [8] John F. Canny. „The Complexity Of Robot Motion Planning“. Dissertation. MIT Press, 1987.
- [9] H. Cruse. „The Function of the Legs in the Free Walking Stick Insect, *Carausius morosus*“. In: *Journal of Comparative Physiology A* (1976).
- [10] P. Łabecki D. Belter und P. Skrzypczynski. „Estimating terrain elevation maps from sparse and uncertain multi-sensor data“. In: *International Conference on Robotics and Biomimetics* (2012).

- [11] Péter Fankhauser, Michael Bloesch und Marco Hutter. „Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization“. In: *ETH Zürich Research Collection* (2018).
- [12] FZI-Karlsruhe. *Projekt Lauron V.* Hrsg. von FZI-Karlsruhe. 2019. URL: <https://www.fzi.de/forschung/projekt-details/lauron/>.
- [13] GregorDS. *The six degrees of freedom*. 2015. URL: <https://upload.wikimedia.org/wikipedia/commons/2/2a/6DOF.svg>.
- [14] Daniel Groß. *Hardwareoptimierung eines Laufroboters und Überführung der Elektronik auf Leiterplatten*. Techn. Ber. Hochschule Mannheim, 2018.
- [15] Canberk Suat Gurel. „Hexapod Modelling, Path Planning and Control“. Dissertation. University of Maryland, 2017.
- [16] Claude Heischbourg. „Gewinnung eines Geländeprofils mittels Sensordaten und Bewegungsplanung“. Bachelor-Thesis. Hochschule Mannheim, 2007.
- [17] André Herms. „Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren“. Diploma. Institut für Verteilte Systeme der Otto-von-Guericke-Universität Magdeburg, 2003.
- [18] Thomas Ihme. „Steuerung sechsbeiniger Laufroboter unter dem Aspekt technischer Anwendungen“. Dissertation. Universität Magdeburg, 2002.
- [19] Daniel Kuhn. „An ape-inspired robot“. In: *ROBOTS* (2012).
- [20] Maike Kuhnert. *Vergleich und Bewertung von 3D-Sensoren für mobile Roboter*. Techn. Ber. TU Dortmund, 19. Okt. 2009.
- [21] Daniel Leidner. „Planung und Ausführung von alltäglichen zweihändigen Manipulationsaufgaben“. Diploma. Deutsches Zentrum für Luft- und Raumfahrt, Hochschule Mannheim, 2010.
- [22] Uwe Müller-Wilm u. a. „Kinematic Model of a Stick Insect as an Example of a Six-Legged Walking System“. In: *Volume 1, Number 2* (1992).
- [23] Simon Müller, Vinoth Kumar und Sascha Moos. *Autarkisierung des sechsbeinigen Roboters*. Techn. Ber. Institut für Robotik, Hochschule Mannheim, 2018.
- [24] Kenzo Nonami u. a. *Hydraulically Actuated Hexapod (Robots Design, Implementation and Control)*. Springer, 2014. Kap. 1, S. 1. 285 S.

- [25] Friedrich Pfeiffer, Jürgen Eltze und Hans-Jürgen Weidemann. „Six-legged technical walking considering biological principles“. In: *Robotics and Autonomous Systems* 14 (1995).
- [26] Uli Ruffler. „Laufplanung basierend auf realitätsnahen Umgebungsdaten für einen sechbeinigen Laufroboter“. Diploma Thesis. Institut für Robotik der Fakultät Informatik an der Hochschule Mannheim, 2006.
- [27] Jan-Ullrich Schamburek. *Informationsverarbeitung in Lebewesen (Bewegungssteuerung bei Insekten)*. Hrsg. von F. Beutler u. a. 2003. URL: <http://docplayer.org/26561188>  
-Informationsverarbeitung-in-lebewesen.html (besucht am 30.04.2019).
- [28] Malte Schilling u. a. „Walknet, a bio-inspired controller for hexapod walking“. In: *Biological Cybernetics* (2013).
- [29] Seth Teller. *Configuration Space for Motion Planning*. Hrsg. von MIT. 2015. URL: <http://courses.csail.mit.edu/6.141/spring2013/pub/lectures/Lec1-2-ConfigurationSpace.pdf>.
- [30] Bernhard Wirnitzer. *Bildverarbeitung und Mustererkennung (Vorlesungsskript)*. Hrsg. von Bernhard Wirnitzer. 2013.

