Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren

Diplomarbeit

André Herms

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Verteilte Systeme

Aufgabenstellung: Prof. Dr. Thomas Ihme
Betreuung: Dipl.-Inform. Richard Bade, Prof. Dr. Thomas Ihme

Kurzfassung

Diese Arbeit befasst sich mit der Planung von Laufbewegungen für sechsbeinige Laufroboter. Ziel ist, durch Verwendung von Umgebungsinformationen bessere Laufbewegungen zu erreichen. Dabei sollen bestimmte Kriterien eingehalten und möglichst gut erfüllt werden. Um die hierbei geforderte Optimierung zu ermöglichen, wird das Laufplanungsproblem als Optimierungsproblem betrachtet. Dazu erfolgt eine formale Beschreibung über Eingabe, Lösungsraum und Bewertungsfunktion. Zum Lösen des gestellten Problems werden allgemeine Lösungsverfahren betrachtet. Anhand der Anforderungen zeigen sich Random Sampling, Lokale Suche und Simulated Annealing als geeignet. Die Verfahren werden implementiert und getestet. Dabei stellt sich nur das Random Sampling als praktisch geeignet heraus und wird im weiteren verwendet. Abschließend erfolgen Tests des Laufplaners anhand unterschiedlicher Szenarien. Es wird dabei ersichtlich, dass der entwickelte Laufplaner die gestellten Anforderungen erfüllt.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In der Robotik nehmen *autonome mobile Roboter* eine besondere Rolle ein. Sie sind in der Lage, sich selbständig, ohne externe Steuerung, durch ein Gelände zu bewegen. Von besonderem Interesse sind hierbei die *Laufroboter*. Sie besitzen eine höhere Mobilität und können Gelände bewältigen, die für rad- oder kettengetriebene Roboter unpassierbar sind. Es können Hindernisse überschritten oder Halt auf einem Untergrund gefunden werden, der nur spärlich sichere Stellen aufweist.

Diese Vorteile bedingen aber eine erheblich höhere Komplexität bei der Ansteuerung. Laufroboter verfügen über viele Freiheitsgrade, die während der Bewegung kontrolliert werden. Es existieren einige einfache Verfahren zur Steuerung. Sie basieren auf der Annahme, dass der Roboter auf ebenem Untergrund läuft. Diese Verfahren erzielen sehr gute Ergebnisse. Der entscheidende Nachteil ist dabei, dass unebenes Gelände gleichzeitig ausgeschlossen wird. Die Vorteile von Laufrobotern gegenüber radgetriebenen Robotern kommen somit nicht zum Tragen.

Diese einfachen Steuerungen wurden erweitert, um auch unebenes Gelände behandelt werden kann. Dabei werden reaktive Komponenten eingesetzt. Stößt der Roboter bei der Bewegung auf ein Hindernis, erfolgt eine Ausnahmenbehandlung. Somit ist auch eine Bewegung möglich, wenn gelegentlich Hindernisse auftreten. Dieser Ansatz hat jedoch den Nachteil, dass Störungen im Bewegungsablauf immer erst dann korrigiert werden, wenn sie bereits aufgetreten sind. Erst bei einer Kollision erkennt der Roboter das Hindernis und reagiert entsprechend.

Wünschenswert wäre aber anstelle einer reaktiven Hindernisbehandlung eine vorausschauende Hindernisvermeidung. Dabei nimmt der Roboter ein Hindernis rechtzeitig war und steuert die Bewegung so, dass es gar nicht erst zu einer Störung des Bewegungsablaufs kommt. Grundlage hierfür ist die Verwendung von Sensoren. Hindernisse können damit frühzeitig erkannt werden. Zusätzlich wird aber auch ein *Laufplaner* benötigt. Er berechnet basierend auf den Sensordaten eine geeignete Bewegung, wobei die vorher erkannten Hindernisse berücksichtigt werden.

Bisher ist kein Verfahren bekannt, das eine exakte Laufplanung durchführt, ohne dabei die möglichen Bewegungen einzuschränken. Aufgrund der Komplexität des Problems ist es auch unwahrscheinlich, dass ein ausreichend schnelles und exaktes Verfahren existiert. Soll die Planung online erfolgen, kann daher nur eine suboptimale Lösung geliefert werden. Hierfür gibt es unterschiedliche Verfahren, die auf verschiedenen Ansätzen basieren. In dieser Arbeit wird ein Verfahren gesucht, das für ein gegebenes System angepasst ist, und somit möglichst gute Ergebnisse erzielt.

1.2 Aufgabenstellung

Die Zielsetzung dieser Arbeit ist der Entwurf eines Laufplaners für den Laufroboter LAURON III. Durch Sensoren wird ein Modell der Umgebung gewonnen. Die darin enthaltenen Informationen sollen genutzt werden, um eine Laufbewegung zu planen. Um möglichst gute Lösungen zu erzielen, soll das Problem als Optimierungsproblem aufgefasst werden. Dadurch werden hinsichtlich gegebener Kriterien optimale Ergebnisse bestimmt. Da eine exakte Lösung nur mit unverhältnismäßig hohem Aufwand zu finden ist, soll eine geeignete Heuristik zum Einsatz kommen. Die absolute Optimalität der Lösungen geht damit zwar verloren, es sollten sich dennoch hinreichend gute bestimmen lassen. Zur Verbesserung der Rechenleistung ist eine zusätzliche Verwendung der Mikrocontroller zu erwägen.

Der Laufplaner ist so zu entwerfen, dass er in die gegebene Steuerungsarchitektur des Roboters integriert werden kann. Das Verfahren ist zu implementieren und an geeigneten Szenarien zu testen.

1.3 Ergebnisse

Die Arbeit stellt einen Laufplaner für den Laufroboter LAURON III vor. Es wird gezeigt, wie sich das Problem der Laufplanung auf ein entsprechendes Optimierungsproblem abbilden lässt. Durch Lösen dessen kann eine geeignete Laufbewegung gefunden werden. Zusätzlich ermöglicht eine Bewertungsfunktion die individuelle Anpassung der Lösungen. Durch höhere Bewertung einzelner Eigenschaften der Bewegung werden diese bevorzugt. Außerdem gelingt es, zulässige aber extrem schlechte Lösungen auszuschließen.

Zum Lösen des Optimierungsproblems wurden mehrere Ansätze theoretisch und zum Teil auch praktisch untersucht. Dabei stellte sich der Laufplaner basierend auf Random Sampling als geeignet heraus. Der entwickelte Laufplaner ist in der Lage, in kurzer Zeit hinreichend gute Lösungen zu finden. Steht mehr Rechenleistung oder Zeit zur Verfügung, kann das Ergebnis stetig verbessert werden. Der Laufplaner lässt sich parallelisieren, was durch den Einsatz in einem Mosix-Cluster gezeigt wurde. Da nur sehr wenig Speicher benötigt wird, können durch verteilte Berechnung auch alle Prozessoren des LAURON III genutzt werden. Dies führt potentiell zu besseren Lösungen.

Der Laufplaner wurde in unterschiedlichen Szenarien getestet. Flaches Gelände, wie es auch von einfacheren Steuerungen bewältigt wird, stellt dabei kein Problem dar. Es wird sehr schnell eine gute Lösung gefunden. Kleinere Hindernisse werden ebenfalls problemlos behandelt. Dies wäre auch noch mit reaktiven Steuerungen zu erreichen. Beachtenswert ist jedoch, dass mit dem hier vorgestellten Verfahren selbst extreme Situationen bewältigt werden. So findet der Laufplaner sogar eine Lösung für das Überschreiten großer Lücken. Mit einem der klassischen Ansätze wäre dies nicht möglich gewesen.

1.4 Gliederung

Die Grundlagen für diese Arbeit werden in Kapitel 2 behandelt. Es wird der aktuelle Stand der Technik beschrieben, wobei auf die entsprechenden Arbeiten verwiesen wird. Anschlie-

ßend wird die Problemstellung näher betrachtet. Die Systemarchitektur des LAURON III wird vorgestellt, und es werden die Anforderungen beschrieben, die an den Laufplaner zu stellen sind.

Die Darstellung des Laufplanungsproblems als Optimierungsproblem wird in Kapitel 3 erläutert. Zunächst wird die Struktur der Eingabe des Algorithmus definiert. Anschließend erfolgt eine Beschreibung der einzelnen Elemente des Lösungsraums. Eine mögliche Bewertungsfunktion wird vorgestellt. Es folgt die Betrachtung einzelner Terme, die bestimmte Kriterien der Bewegung widerspiegeln.

Für das Lösen des vorher definierten Optimierungsproblems werden in Kapitel 4 geeignete Heuristiken untersucht. Betrachtet werden dabei Greedy-Verfahren, Branch and Bound, Random Sampling, Lokale Suche, Tabu-Suche, Simulated Annealing und Genetische Algorithmen. Die Beurteilung erfolgt vorerst nur theoretisch anhand der in Kapitel 2 beschriebenen Voraussetzungen.

Die in Kapitel 4 als mögliche Kandidaten ermittelten Verfahren Random Sampling, Lokale Suche und Simulated Annealing werden in den Kapiteln 5 und 6 konkret betrachtet. Kapitel 5 beschreibt den für das Random Sampling nötigen Algorithmus zum Erzeugen zufälliger Lösungen. Anschließend findet eine Bewertung des Random-Sampling-Algorithmus hinsichtlich der Eignung als Laufplaner statt. In Kapitel 6 werden die nötigen Algorithmen für Lokale Suche und Simulated Annealing dargestellt. Dabei wird vor allem auf die Definition der Nachbarschaft eingegangen. Auch hier erfolgt eine Bewertung der beiden Verfahren.

Die Funktionalität des Laufplaners wird in Kapitel 7 überprüft. Dazu werden einige Szenarien mit dem Laufplaner berechnet. Das Kapitel beginnt mit der Beschreibung der Testumgebung. Es folgt eine Erläuterung der einzelnen Szenarien sowie die Bewertung der Ergebnisse.

Das letzte Kapitel bietet eine Zusammenfassung der Arbeit und einen Ausblick auf weitere Ideen, die hierauf aufbauen können.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen der Arbeit näher erläutert werden. Zunächst wird ein Überblick zum Stand der Technik gegeben. Im Anschluss daran wird der Laufroboter LAURON III näher vorgestellt. Es folgt die Erläuterung der verwendeten Hardware und Softwareumgebung, in der später der Laufplaner zum Einsatz kommen soll. Die daraus resultierenden Anforderungen werden im folgenden Abschnitt 2.3 noch konkret dargestellt.

2.1 Stand der Technik

Dieser Abschnitt gibt einen Überblick zu den bisherigen Arbeiten im Bereich der Steuerung sechsbeiniger Laufroboter. Dabei werden die Verfahren grob in drei Klassen gegliedert: statische Verfahren, reaktive Verfahren und planende Laufsteuerungen. Jeder Klasse werden entsprechende Arbeiten zugeordnet und kurz auf deren Inhalt eingegangen.

2.1.1 Statische Verfahren

Statische Verfahren erzeugen eine einfache Laufbewegung nach einem fest vorgegebenen Schema. Äußere Einflüsse werden dabei nicht beachtet. Kollisionen, Hindernisse oder unebenes Gelände können meist nur unzureichend behandelt werden. Für flaches Gelände sind die Verfahren aber gut geeignet. Außerdem lassen sie sich dank ihrer Einfachheit mit wenig Aufwand umsetzen.

Laufmuster

Ein sehr einfaches Verfahren sind *Laufmuster* (gait pattern). Ein Laufmuster definiert die Abfolge, nach der die einzelnen Füße angehoben und abgesetzt werden. Die Endstellung der Füße entspricht dabei der am Anfang. So lässt sich das Laufmuster beliebig oft hintereinander ausführen, wodurch eine kontinuierliche Bewegung erreicht wird.

Es wurden zahlreiche Untersuchungen zu geeigneten Laufmustern durchgeführt, und die besten Lösungen hinsichtlich Geschwindigkeit oder Stabilität ermittelt [42]. Durch Betrachtung von Laufmustern als Optimierungsproblem und anschließendem Lösen mit Hilfe von Genetischen Algorithmen [30, 36, 37] und Backtracking [35] wurden diese bestätigt.

Laufmuster werden in vielen Arbeiten verwendet. Dies kann direkt geschehen, wie in [21, 15, 38]. In einigen Arbeiten wie [18, 10, 27] wurden Laufmuster aber auch auf Neuronale

Netze übertragen. Dabei wurden die Berechnung des Laufmusters und die Berechnung der Gelenkwinkel zusammengefasst. Das Ergebnis ist eine Optimierung der Rechengeschwindigkeit, wenn auch zu Lasten der Genauigkeit.

Zelluläre Automaten

Bereits in den frühen 70er Jahren wurde das Laufverhalten von Insekten wie der indischen Stabheuschrecke (Carausius morosus) untersucht [9, 32]. Dabei wurden sehr einfache Regeln [11] gefunden, die ein korrektes Laufverhalten ermöglichen. Diese Regeln werden meist in Form eines zellulären Automaten beschrieben. Dabei entspricht die Struktur des Automaten der des strickleiterförmigen Nervensystems von Insekten. Einzelne Beine sind nur mit ihren direkten Nachbarn verbunden. Über diese Verbindungen werden Reize ausgelöst, die ein Anheben oder Absenken der Beine unterdrücken beziehungsweise aktivieren. Die durch Beobachtung gefundenen Regeln wurden bestätigt, indem ein entsprechendes Optimierungsproblem mittels eines Genetischen Algorithmus gelöst wurde [13].

Dieses Verfahren hat gewisse Vorteile gegenüber normalen Laufmustern. So können Unregelmäßigkeiten in der Synchronität der Beine durch die Regeln automatisch ausgeglichen werden. Die Laufbewegung ist aber auch hier nicht mit äußeren Einflüssen gekoppelt und kann deshalb nicht auf Störungen wie Hindernisse reagieren.

2.1.2 Reaktive Verfahren

Der entscheidende Nachteil aller statischen Laufsteuerungen ist, dass sie nur für ebenes Gelände geeignet sind. Hindernisse können zu Störungen der Laufbewegung führen. Daher wurden diese Ansätze um reaktive Komponenten erweitert. Durch eine Art Reflex werden zusätzliche Verhaltensweisen erreicht, die die bestehenden Nachteile ausgleichen sollen.

Sehr häufig wird ein *Elevator-Reflex* [17, 18] verwendet. Er dient dem Überwinden kleinerer Hindernisse. Stößt ein Bein bei einer Bewegung gegen ein Hindernis, wird der Fuß automatisch etwas angehoben. Durch eventuell mehrmaliges Anwenden kann die entsprechende Stelle überwunden werden. So ist beispielsweise auch das Laufen über Stufen möglich.

Für Vertiefungen im Untergrund wird ein *Suchreflex* [17] angewendet. Findet das Bein beim Aufsetzen keinen festen Untergrund, wird in der Nachbarschaft des Aufsetzpunktes nach einer geeigneten Position gesucht. Wird eine solche gefunden, kann der Lauf fortgesetzt werden.

In [18] wird ein *Aufstandreflex* vorgestellt. Er dient dem Verhindern zeitlicher Störungen, indem sichergestellt wird, dass sich zu jedem Zeitpunkt die richtige Anzahl von Füßen auf dem Boden befindet. In einigen Fällen führt dies aber zu Störungen im Bewegungsablauf. Fordert das Laufmuster ein gleichzeitiges Aufsetzen mehrerer Beine und können diese bedingt durch die technische Umsetzung nicht absolut synchron gesteuert werden, wird ständig der Reflex ausgelöst. Damit dominiert die Ausnahmebehandlung das normale Verhalten, was zu erheblichen Störungen der Laufbewegung beiträgt.

Die Reflexe sind ein einfacher Ansatz, den bestehenden statischen Laufsteuerungen eine Anwendung auf unebenem Untergrund mit Hindernissen zu ermöglichen. Hierzu werden beispielsweise Drucksensoren verwendet. Die resultierende Steuerung geht dabei jedoch sehr optimistisch vor. Das Laufverhalten basiert immer noch auf der Annahme von ebenem Gelände. Die Reflexe dienen lediglich als Notlösung. In stark unebenem Gelände mit vielen Hindernissen können die Reflexe jedoch das normale Laufmuster dominieren. Der Roboter „tastet" sich dann nur noch vorwärts.

2.1.3 Planende Laufsteuerungen

Der Nachteil der reaktiven Laufsteuerungen ist, dass sie auf ein Hindernis immer erst dann reagieren, wenn eine Störung des Laufens auftritt. Diesen Nachteil umgehen planende Laufsteuerungen. Sie versuchen *vorausschauend* mit Hindernissen umzugehen und damit eine Störung der Laufbewegung zu verhindern. Hierzu werden Sensoren verwendet, mit denen die Umgebung untersucht wird. Mit Hilfe der gewonnenen Daten erfolgt eine Planung, die ein Vermeiden der Kollision mit Hindernissen ermöglicht.

Die Planung kann auf unterschiedlichen Ebenen erfolgen. In [18] dienen die Sensorinformationen lediglich der Vermeidung des Elevator-Reflex, indem das Bein vorausschauend auf die erforderliche Höhe gehoben wird. Die restliche Steuerung erfolgt wie bei reaktiven Laufmustern.

Ähnlich wird in [25] vorgegangen. Die Umgebungsdaten werden verwendet, um gewisse Parameter wie Hubhöhe und -weite eines Beins im Laufmuster anzupassen. Reaktive Komponenten kommen nicht zum Einsatz. Eine wichtige Einschränkung dieses Ansatzes ist, dass jede Fußposition im Gelände erreichbar sein muss. Es darf keine zu hohen oder zu tiefen Bereiche geben.

In [14] wird anhand der Umgebungsdaten die gesamte Roboterbewegung inklusive der Beinpositionen geplant. Dazu dient ein mehrstufiges Verfahren basierend auf Genetischen Algorithmen und Backtracking. Als Nachteil wird erwähnt, dass unter Umständen keine Lösung gefunden wird. Außerdem werden keine Aussagen über die benötigte Rechenzeit von Backtracking und Genetischem Algorithmus gemacht. Es bleibt offen, ob sie für die Planung zur Laufzeit geeignet sind.

Der simulierte Laufroboter Simpod [44] verwendet Untergrundinformationen, um zwischen gegebenen Laufmustern umzuschalten. So wird versucht, stets eine geeignete Fußposition zu finden. Dies ist aber nur insoweit möglich, wie es die zur Wahl stehenden Laufmuster zulassen.

Konzepte zur Steuerung von Robotern finden auch in anderen Bereichen Verwendung. So existieren ähnliche Verfahren für die Steuerung von Charakteren in Computerspielen. Dort werden für realistische Laufbewegungen Gelenkmodelle betrachtet. Der Charakter muss sich wie ein gelenkbasierter Roboter durch eine virtuelle Landschaft bewegen. Ein entsprechendes Planungsverfahren wird in [26] vorgestellt. Hier werden optimale Bewegungen durch Backtracking berechnet.

Ein Laufplaner basierend auf dem heuristischen Optimierungsverfahren Ordinal Optimization wird in [8] vorgestellt. Dieses ähnelt sehr stark dem in dieser Arbeit verwendeten

Random Sampling. Allerdings wurden dort erhebliche Einschränkungen für die Bewegungen vorgesehen. Es müssen sich nach jedem Schritt alle sechs Füße auf dem Boden befinden. Andere Bewegungen werden ausgeschlossen. Existieren nur Bewegungen, die dieser Einschränkung nicht genügen, kann keine Lösung gefunden werden.

2.2 Aufbau des LAURON III

2.2.1 Entwicklungsgeschichte

In den 90er Jahren konstruierte die Gruppe Interaktive Diagnose- und Servicesysteme am Forschungszentrum Informatik der Universität Karlsruhe den sechsbeinigen Laufroboter LAURON [5, 39](Abbildung 2.1). Ziel war die Entwicklung und praktische Erprobung von Laufsteuerungen. Dabei wurden ursprünglich diverse Arten von Neuronalen Netzen untersucht [6, 23].

Abbildung 2.1: LAURON (aus [16])

Bei der Forschung zeigten sich einige Unzulänglichkeiten in Bezug auf Sensorausstattung und Mechanik. Diese Schwachstellen wurden bei der Entwicklung des Nachfolgers LAURON II (Abbildung 2.2) behoben. Das System wurde im Laufe der Jahre ständig weiterentwickelt. Mittlerweile existiert der LAURON III (Abbildung 2.3). Dieser zeichnet sich durch verbesserte Kraftsensoren an den Beinen und weitere kleine Details in der Mechanik aus.

2.2.2 Hardware des LAURON III

Die Hardwarearchitektur des LAURON III ist hierarchisch aufgebaut. Dabei wird zwischen dem Gelenksystem, dem Beinsystem, dem Kopfsystem und dem Gesamtsystem unterschieden. Dies ermöglicht es, die Steuerung auf einer Ebene getrennt von den anderen zu betrachten. Erst hierdurch wird dieses recht komplexe System handhabbar. Einzelne Komponenten können unabhängig voneinander entwickelt und getestet werden. Teile der Hardware können einfach ausgetauscht werden, um beispielsweise mehr Leistung für gewisse Aufgaben zur erhalten. Außerdem lassen sich leicht neue Komponenten hinzufügen.

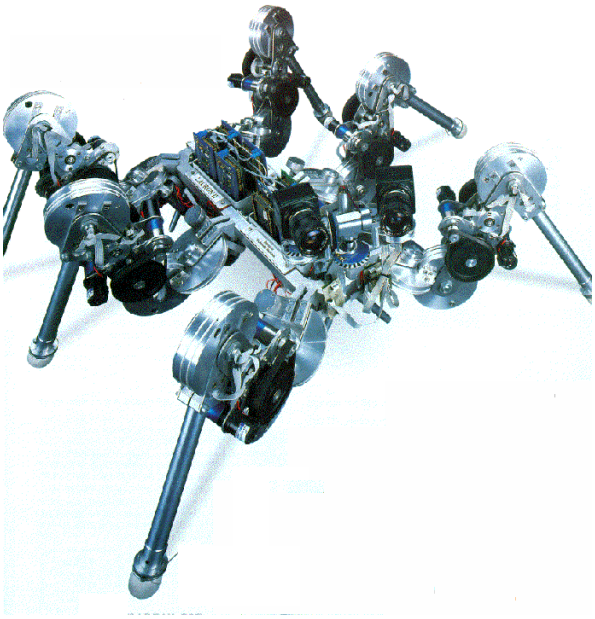


Abbildung 2.2: LAURON II (aus [16])

Gesamtsystem

Der LAURON III besteht aus einem Hauptkörper mit sechs Beinen. Die Konstruktion ist in ihrer Struktur dem Körperbau der indischen Stabheuschrecke nachempfunden. Die Beine sind seitlich am Körper angebracht und in ihrem Aufbau mit Ausnahme der Winkelstellung identisch. Ein PC/104 übernimmt die höheren Steuerungsaufgaben. Die Befehle werden über CAN-Bus [41] an die Beine und den Kopf gesendet. Zum Empfang von externen Anweisungen steht eine WLAN-Karte zur Verfügung. Die Video-Signale des Kamerakopfes werden über einen Framegrabber ausgewertet.

Beinsystem

Die Beine des LAURON III sind alle identisch aufgebaut. Sie verfügen über jeweils drei Drehgelenke. Die daraus resultierenden drei Freiheitsgrade gestatten eine freie Positionierung des Fußpunktes innerhalb seiner Reichweite. Außerdem steht ein Kraftsensor zur Verfügung, der die Kräfte am Fußpunkt misst. Die Ansteuerung erfolgt über einen c167 Mikrocontroller [24]. Alle c167 des Roboters sind untereinander und mit dem PC/104 über CAN-Bus verbunden.

Gelenksystem

Jedes der drei Gelenke eines Beins wird über einen Elektromotor bewegt. Durch Encoder kann der aktuelle Winkel mit einer Auflösung von 0,4° ermittelt werden. Die Spannung der Motoren wird vom Mikrocontroller über Pulsweiten-Modulation gesteuert.

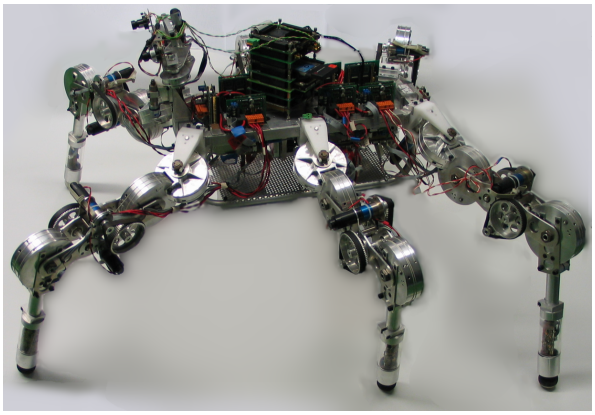


Abbildung 2.3: LAURON III

Kopfsystem

Das Kopfsystem dient dem Erfassen der Umgebung. Dazu verfügt es über zwei Kameras, die ein Bild aus leicht versetzter Perspektive aufnehmen. Durch eine Stereobildverarbeitung kann hieraus ein Tiefenprofil gewonnen werden. Die Kameras sind auf einem beweglichen Kamerakopf montiert. Die Blickrichtung kann über zwei Drehgelenke bestimmt werden. Der Winkel zwischen den Kameras lässt sich über ein drittes Gelenk beeinflussen. Zur Steuerung wird wie bei den Beinen ein c167 eingesetzt. Die Auswertung der Kamerabilder erfolgt nicht auf dem Mikrocontroller, da dieser zu wenig Rechenleistung bietet. Stattdessen werden die Bilder über einen Framegrabber auf dem PC/104 aufgenommen und dort ausgewertet.

2.2.3 Steuerungssoftware

Ähnlich der Hardware des LAURON III ist auch die Steuerungssoftware hierarchisch aufgebaut (vgl. Abbildung 2.6). Sie läuft verteilt auf den Mikrocontrollern und dem PC/104. Dabei kommt auf der PC-Seite Linux mit RTLinux-Erweiterung als Betriebssystem zum Einsatz. Die Mikrocontroller verfügen über kein eigenes Betriebssystem. Tasks werden über Interruptroutinen erzeugt. Code wird in Form von Modulen vom PC/104 über den CAN-Bus geladen.

Die Hierarchie der Steuerungssoftware spiegelt zum Teil die der Hardware wieder. Die Software besteht aus Modulen, die für die unterschiedlichen Teilaufgaben zuständig sind. Durch die Modularisierung können Teilaufgaben getrennt voneinander behandelt werden. Auch ein späteres Austauschen einzelner Module wird möglich.

Gelenkregler

Die Gelenke der Roboterbeine werden durch einen Gelenkregler gesteuert. Er läuft als periodische Task auf den Mikrocontrollern. Als Eingabe wird der gewünschte Winkel vorgegeben. Der Regler sorgt dann dafür, dass dieser vom Gelenk angenommen wird.

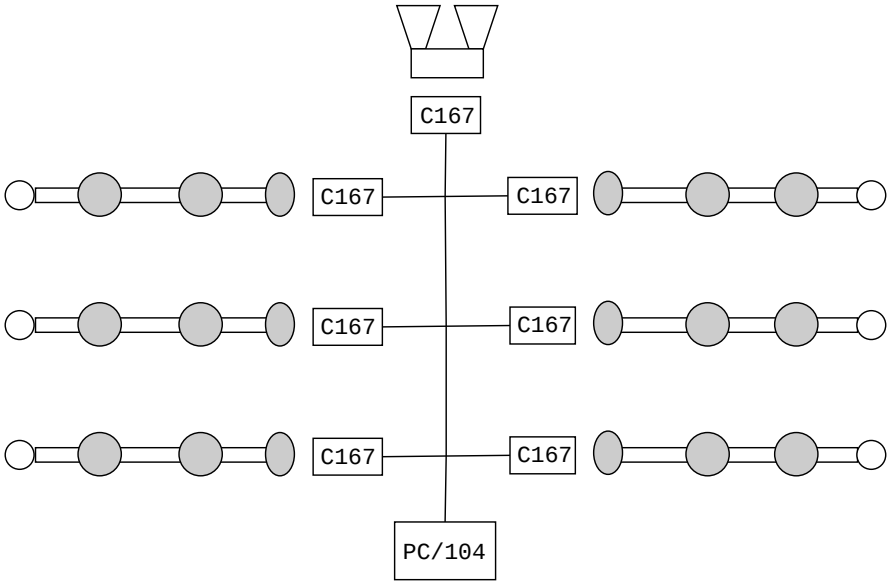


Abbildung 2.4: Gesamtsystem des LAURON III

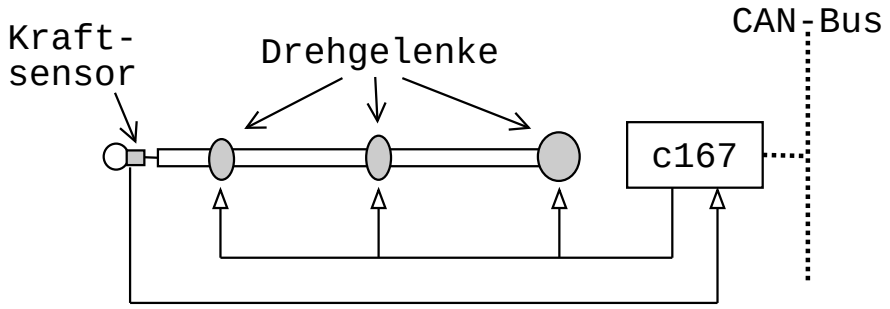


Abbildung 2.5: Beinsystem des LAURON III

Beinregler

Der Beinregler dient der Steuerung eines kompletten Beins. Geregelt wird die Position des Fußpunktes (Endpunkt des Beins). Sie wird als dreidimensionale Koordinate vorgegeben. Der Beinregler stellt die Gelenke so ein, dass die gewünschte Position erreicht wird. Zur Bestimmung der Gelenkwinkel aus einer gegebenen Position wird eine kinematische Transformation benötigt. Dies kann wie in [21] analytisch geschehen, aber auch Approximationen wie in [18] über neuronale Netze sind denkbar.

Die Laufsteuerung stellt bestimmte Anforderungen an den Beinregler. Wird eine neue Zielposition gegeben, darf der Regler diese nicht direkt ansteuern. Wichtig ist auch der Verlauf der Bewegung. Hierbei werden zwei Modi unterschieden:

Stützmodus: Das Bein befindet sich auf dem Untergrund und stützt den Roboter. In diesem Modus sind die Beine für die Verschiebung des Roboterkörpers verantwortlich.

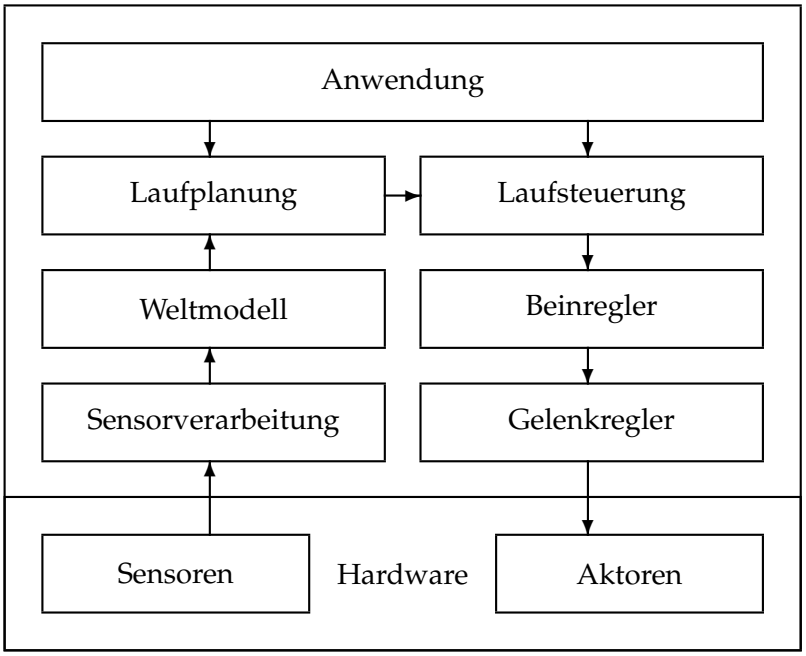


Abbildung 2.6: Hierarchie der Steuerungssoftware

Eine Positionsänderung in eine bestimmte Richtung wird erreicht, indem alle stützenden Fußpunkte in die entgegengesetzte Richtung bewegt werden. Da diese Bewegung relativ zum Körper stattfindet, die Fußpunkte aber fest auf dem Untergrund stehen, wird der Roboterkörper verschoben. Damit dies auch auf einer wohldefinierten Bahn geschieht, müssen sich alle Fußpunkte mit gleicher Geschwindigkeit in die gleiche Richtung bewegen. Als Vorgabe dienen die Endposition und der Zeitpunkt, zu dem die Bewegung abgeschlossen sein muss.

Hebemodus: Das Bein wird vorwärts gesetzt. Dabei hebt es sich vom Untergrund ab, bewegt sich zur vorgegebenen Position und setzt dort wieder auf. Der exakte Verlauf der Bewegung ist hierbei unerheblich. Beim Aufsetzten wird nicht nur die vorgegebene Position erreicht, sondern mit Hilfe des Drucksensors aktiv der Bodenkontakt gesucht. Zusätzlich zur Endposition wird auch hier ein Zeitpunkt vorgegeben, zu dem die Bewegung abgeschlossen sein muss.

Laufsteuerung

Die Aufgabe der *Laufsteuerung* ist die Koordination aller Beine. Durch eine geeignete Abfolge von Beinbewegungen wird ein Laufen des Roboters erreicht. Für die Stabilität ist es nötig, dass dabei bestimmte Bedingungen eingehalten werden. Beispielsweise müssen immer mindestens drei Beine den Roboter stützen. Die Bewegungen der Beine werden hier vom Laufplaner geliefert. Die Aufgabe der Laufsteuerung ist lediglich die Umsetzung der geplanten Bewegung.

Es ist sinnvoll, die Laufplanung und die Laufsteuerung getrennt voneinander zu betrachten. Durch diese Trennung wird eine Unabhängigkeit von der geplanten Bewegung erreicht. Die Laufsteuerung kann sie umsetzen, muss dies aber nicht. Tritt eine Störung auf, wie etwa ein plötzliches Hindernis, muss die Bewegung nicht fortgesetzt werden. Der Laufsteuerung kann stoppen und eventuell eine alternative Bewegung planen. Es wird also möglich, reaktive Elemente zu integrieren. Die Laufplanung könnte damit nicht umgehen, da Informationen über plötzliche Hindernisse während der Planung nicht vorliegen.

Sensorverarbeitung

Die Sensorverarbeitung dient der Aufbereitung der Sensordaten. In diesem Fall sind Daten des Kamerakopfes von Interesse. Die beiden Kameras liefern zeitgleich zwei Bilder, die in der Perspektive leicht versetzt sind. Durch geeignete Verfahren [3, 22] ist es möglich, aus diesen Bildern eine Tiefeninformation für die einzelnen Pixel zu extrahieren.

Weltmodell

Auf Grundlage der Sensorinformationen ist das System in der Lage, ein Modell seiner Umgebung zu erstellen. Dieses Modell enthält Hindernisse und Höheninformationen des Untergrundes. Auf Basis dieser Informationen wird eine Planung von Bewegungen erst möglich. Ohne Wissen könnte dies nicht geschehen.

Die Informationen werden hier aus dem vorherigen Schritt, der Sensorverarbeitung, gewonnen. Dabei müssen die Daten noch transformiert werden, um dem Format des Weltmodells zu entsprechen. Die gewonnenen Informationen werden nicht verworfen, sondern beibehalten und durch neue Sensorwerte ergänzt. So wird das Weltmodell stetig vervollständigt. Grundsätzlich könnten auch statische Daten der Umgebung, wie Karten, verwendet werden. Diese müssen dann aber in einer entsprechend hohen Auflösung vorliegen, wobei der Roboter zusätzlich seine Position in der Karte ermitteln muss.

Anwendung

Die Anwendung legt fest, wohin sich der Roboter bewegen soll. Wie diese Entscheidung genau getroffen wird, ist hier nicht vorgegeben und hängt vom jeweiligen Einsatzzweck ab. Die unterschiedlichsten Anwendungen sind denkbar. So kann ein autonomes System aufgesetzt werden, das aktiv die Umgebung erkundet. Aber auch eine Steuerung durch einen Menschen ist möglich. Von Bedeutung ist hier nur die Ausgabe. Es wird eine Menge von Zielpunkten vorgegeben, für deren Erreichen die unteren Schichten verantwortlich sind.

Laufplanung

Die *Laufplanung* schließlich ist Thema dieser Arbeit. Die Laufsteuerung sendet Befehle an die Beinregler, um eine Laufbewegung zu erreichen. Diese Bewegungen können fest vorgegeben

sein, wie beispielsweise bei Laufmustern. In dieser Arbeit werden sie aber zur Laufzeit berechnet. Mittels der im Weltmodell enthaltenen Informationen wird eine Bewegung gesucht, die zunächst einmal korrekt sein sollte. Das heißt, der Roboter muss durch die Bewegung der Beine laufen, ohne den Halt zu verlieren. Hinzu kommt noch, dass die Bewegung gewisse Kriterien, wie ein frühes Erreichen des Zielpunktes, anstreben soll.

Die Stabilität und die anderen gewünschten Kriterien hängen stark von der Umgebung ab. So kann das Treten auf schrägem Untergrund zum Verlust des stabilen Standes und damit zu Beschädigungen führen. Derartige Fälle lassen sich durch Verwendung von Umgebungsinformationen vermeiden, die aus dem Weltmodell gewonnen werden.

Die Vorgaben für die Planung werden von der Anwendung gestellt. Nach diesen wird eine geeignete Bewegung berechnet und anschließend an die Laufsteuerung übergeben.

2.3 Anforderungen an das Verfahren

Die allgemeine Funktionsweise des Laufplaners wurde im vorhergehenden Abschnitt beschrieben. Dabei wurde nur die Ein- und Ausgabe betrachtet. In diesem Teil sollen weitere Anforderungen an den Laufplaner herausgearbeitet werden.

2.3.1 Optimierung der Bewegung

Der Laufplaner soll eine Bewegung zu einem vorgegebenen Zielpunkt liefern. Dabei werden Umgebungsinformationen zusätzlich herangezogen, um eine bessere Lösung zu erhalten. Grundsätzlich gibt es viele mögliche Bewegungen, mit denen das Ziel erreicht werden kann. Viele davon sind nicht allzu sinnvoll. So kann der Roboter beliebig oft vor und zurück laufen, bevor er das Ziel erreicht. Auch andere Eigenschaften der Bewegung, wie etwa die Kippgefahr des Roboters, können in unerwünschten Bereichen liegen. Solche schlechten Lösungen will man generell ausschließen. Außerdem ist es wünschenswert, eine möglichst gute Laufbewegung zu erzielen. Was unter einer „guten Bewegung" zu verstehen ist, mag je nach Standpunkt anders beantwortet werden. Zumindest sollte sich die jeweilige Güte objektiv über ein geeignetes Maß ausdrücken lassen.

Mit dieser Forderung stellt sich das Problem der Laufplanung als ein Optimierungsproblem dar. Gesucht wird eine Lösung, die das vorgegebene Maß am besten erfüllt. Eine solche Lösung stellt sicher, dass sie zulässig ist und gleichzeitig werden alle schlechten Ergebnisse ausgeschlossen. Um das Problem als ein Optimierungsproblem behandeln zu können, muss es formal definiert werden. Die Menge der möglichen Lösungen sowie eine Bewertungsfunktion werden benötigt. Anschließend kann ein geeigneter Algorithmus auf das Problem angewendet werden.

2.3.2 Komplexität des Problems

In der Literatur finden sich sehr wenige Ansätze zur Laufplanung basierend auf Sensorinformationen. In der Regel wurden einfacherer Steuerungsansätze gewählt, die allerdings auch

nicht so leistungsfähig sind. Für die Laufplanung wurde bisher kein exaktes und effizientes Verfahren gefunden. Dies gestaltet sich schwierig, weil Laufroboter eine sehr hohe Anzahl zu kontrollierender Freiheitsgrade haben. Intuitiv ist klar, dass sich damit ein höherer Aufwand ergibt. Im folgenden Abschnitt soll dies formal begründet werden. Es soll ersichtlich werden, dass tatsächlich die hohe Anzahl der Freiheitsgrade ein exaktes und effizientes Lösen verhindert. Daraus folgt die Notwendigkeit einer guten Heuristik.

C-Space

Das Problem der Bewegungsplanung lässt sich formal mit dem Konzept des *Configuration Space* [31](auch *C-Space*) beschreiben und lösen. Der C-Space ist der Raum über den Freiheitsgraden. Die Anzahl der Dimensionen entspricht nicht der Anzahl in der realen Welt, sondern der Anzahl der Freiheitsgrade des Roboters. Damit entspricht eine Konfiguration des Roboters genau einem Punkt im C-Space. Eine Konfiguration enthält die Position des Roboters, die Orientierung und bei einem Laufroboter die Stellung der Gelenke. Die Repräsentation von Hindernissen im C-Space wird durch „Ankleben" der Roboterdimension gewonnen.

Exakt ausgedrückt bedeutet das:
Sei $W \subset \mathbb{R}^3$ der Raum, in dem sich der Roboter bewegt.
Sei $\mathcal{R}$ eine abstrakte Beschreibung des Roboters.
Sei R eine Funktion

$$R : \mathcal{R} \times \mathbb{R}^d \mapsto 2^W, \tag{2.1}$$

die eine d-dimensionale Konfiguration des Roboters auf die Teilmenge von W abbildet, die vom Roboter unter der Konfiguration eingenommen wird. Die ermittelte Teilmenge enthält beispielsweise die Bereiche, in denen sich der Roboterkörper und die Beine befinden.
Sei $O_i \subset W$ ein Hindernis. In diesen Bereichen darf sich kein Teil des Roboters befinden, sonst läge eine Kollision vor. Ein solches Hindernis im C-Space wird beschrieben über:

$$C_i(\mathcal{R}) = \left\{ p \in \mathbb{R}^d | R(\mathcal{R}, p) \cap O_i \neq \emptyset \right\}. \tag{2.2}$$

Dies sind genau die Konfigurationen, die zu einer Kollision mit dem Hindernis führen würden.

Alle unzulässigen Bereiche im CSpace sind beschrieben über:

$$C - Space(\mathcal{R}) = \bigcup_i C_i(\mathcal{R}) \cup \left\{ p \in \mathbb{R}^d | \text{outOfRange}(\mathcal{R}, p) \right\}. \tag{2.3}$$

Dabei ist outOfRange eine Funktion

$$\text{outOfRange} : \mathcal{R} \times \mathbb{R}^d \mapsto \{\text{true}, \text{false}\}. \tag{2.4}$$

Sie liefert true, falls einer der Freiheitsgrade außerhalb des zulässigen Bereichs liegt (z.B. ein Gelenkwinkel) oder es zu einer Kollision von Gelenken des Roboters bei der gegebenen Konfiguration kommt.

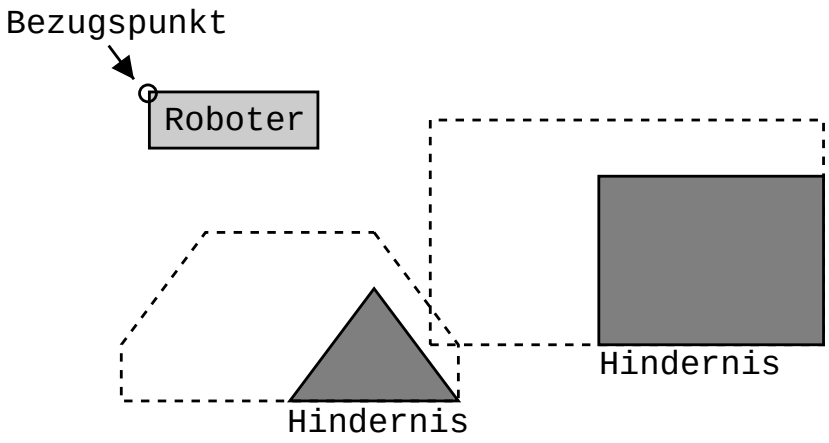


Abbildung 2.7: Beispiel C-Space

In Abbildung 2.7 ist dies anhand eines kleinen Beispiels dargestellt. Der Roboter besitzt nur zwei Freiheitsgrade, die x- und y-Position. Dementsprechend ist der C-Space auch nur zweidimensional und lässt sich somit darstellen. Die reale Welt und deren Hindernisse sind hier Teil des $\mathbb{R}^2$. Zur Veranschaulichung wurden hier die beiden Räume übereinander dargestellt. Die dunkelgrauen Bereiche sind Hindernisse im realen Raum, die gestrichelte Linie beschreibt sie im C-Space. Man kann so deutlich erkennen, wie die Hindernisse um die Ausmaße des Roboters erweitert wurden.

Eine Bewegung des Roboters entspricht einer stetigen Änderung der Konfiguration. Eine Konfiguration wird durch einen Punkt und eine Bewegung durch einen Kurvenverlauf im C-Space beschrieben. Bei einer Bewegung wird ein Hindernis genau dann nicht berührt, wenn die Kurve das Äquivalent im C-Space nicht schneidet. In Abbildung 2.7 ist etwa deutlich erkennbar, dass keine Kurve zwischen den beiden Hindernissen verlaufen kann. Somit kann auch der Roboter nicht kollisionsfrei zwischen den Hindernissen hindurch bewegt werden. Man beachte, dass in diesem Beispiel nur eine x- und y-Bewegung, aber keine Rotation zulässig ist.

Das Problem der Laufplanung wird also auf ein geometrisches Problem zurückgeführt. Gesucht wird eine Kurve im C-Space, die vom Start- zum Zielpunkt führt. Für eine konstante Dimension d des C-Space kann dies in Polynomialzeit berechnet werden. Allerdings ist die Laufzeit auch exponentiell in der Anzahl der Freiheitsgrade [7].

Bei Laufrobotern ist gerade die Anzahl der Freiheitsgrade extrem hoch. Im vorliegenden Fall sind dies drei Freiheitsgrade in je sechs Beinen, sowie zwei in der Roboterposition und eventuell die Orientierung. Dies ergibt 20 Freiheitsgrade. Das Verfahren würde zwar polynomiale Laufzeit besitzen, aber mit einem sehr großen Faktor. Dabei wurde noch nicht die Güte betrachtet, sondern nur das Finden einer zulässigen Lösung. Das Lösen des zugehörigen Optimierungsproblems ist mindestens genauso schwer. Für eine Online-Planung ist das Verfahren daher nicht geeignet.

Um das Problem dennoch handhaben zu können, muss es über eine *Heuristik* gelöst werden. Dies hat zur Folge, dass in der Regel nur eine suboptimale Lösung ermittelt wird. Dafür kann

die Laufzeit aber polynomial beschränkt werden. Erst dadurch wird eine Online-Planung möglich.

2.3.3 Laufzeitkontrolle der Algorithmen

Die Planung der Bewegung nimmt eine bestimmte Zeit in Anspruch. Solange sie nicht abgeschlossen ist, kann die Bewegung auch nicht ausgeführt werden. Würde die Berechnung immer erst dann stattfinden, wenn eine neue Zielposition vorgegeben wird, entstünden sehr ruckartige Bewegungen. Wünschenswert ist daher, dass die Planung bereits abgeschlossen ist, wenn sie ausgeführt werden soll.

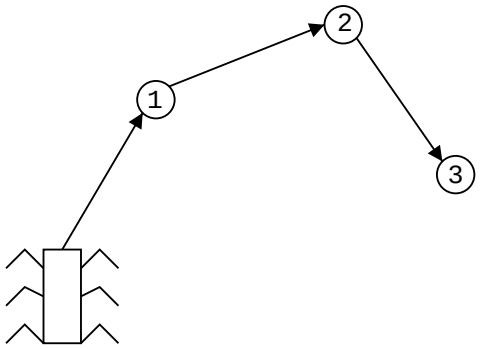


Abbildung 2.8: Vorgabe mehrerer Zielpunkte

Um dies zu ermöglichen, wird nicht nur ein Zielpunkt vorgegeben, sondern wie in Abbildung 2.8 mehrere, die nacheinander erreicht werden sollen. Noch während ein Wegstück abgelaufen wird, erfolgt die Berechnung des darauf folgenden. Läuft der Roboter zum Zielpunkt 1, berechnet er gleichzeitig die Bewegung von Zielpunkt 1 zu Zielpunkt 2. Damit steht dem Algorithmus zur Berechnung die Zeit zur Verfügung, die der Roboter zum Erreichen des nächsten Ziels braucht.

Diese Zeit ist sehr variabel. Sie hängt von der Dauer der Bewegung ab und ist damit nicht fest vorgegeben. Wird nun ein Algorithmus gesucht, der diese Zeit unabhängig von ihrer Länge immer einhält, ergeben sich zwei Möglichkeiten. Es kann ein Verfahren gewählt werden, das sehr schnell arbeitet und damit ein rechtzeitiges Beenden garantiert. Die zweite und hier bevorzugte Möglichkeit sind die so genannten *Anytime-Algorithmen*.

Anytime-Algorithmen

In der Regel liefern Heuristiken eine bessere Lösung, je mehr Rechenzeit sie zur Verfügung haben. Wird diese nun wie im ersten Fall extrem eingeschränkt, sind auch die Ergebnisse von eingeschränkter Qualität. Anytime-Algorithmen erlauben dagegen, die Rechenzeit flexibel anzupassen.

Das Konzept wurde erstmals von *Boddy* und *Dean* in [12] vorgestellt. Ein Anytime-Algorithmus garantiert demnach mindestens:

- Eine Lösung liegt zu jedem Zeitpunkt der Berechnung vor.
- Die Qualität der Lösung verbessert sich, je mehr Rechenzeit zur Verfügung steht.

Zusätzlich wird oft gefordert:

- Die Lösung kann nicht schlechter werden, wenn mehr Rechenzeit verwendet wird.
- Der Algorithmus kann an einer beliebigen Stelle unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden.
- Für eine gegebene Rechenzeit kann die Qualität der Lösung deterministisch bestimmt werden.

Die letzte Forderung wird in dieser Arbeit nicht aufrechterhalten. Bei den hier betrachteten Heuristiken kann selten überhaupt eine Qualität der Lösung garantiert werden. Sie kann demzufolge auch nicht berechnet werden.

Durch Verwendung eines Anytime-Algorithmus kann die zur Verfügung stehende Rechenzeit zur Laufplanung maximal ausgenutzt werden. Sobald der Roboter den aktuellen Zielpunkt erreicht hat, wird die Berechnung abgebrochen. Normalerweise liegt dann eine geplante Bewegung zum nächsten Zielpunkt vor, die direkt verwendet werden kann. Somit kann die Bewegung ohne weiteres fortgesetzt werden.

Es ist unter Umständen möglich, dass beim Erreichen des alten Zielpunktes noch keine gültige Bewegung ermittelt wurde. In diesem Fall kann die Bewegung nicht fortgesetzt werden. Die Anytime-Definition verlangt, dass immer eine Lösung vorliegt. Hier wird deshalb zwischen Lösungen und gültigen Bewegungen unterschieden. Solange keine gültige Bewegung gefunden wurde, hat die Lösung den Wert *unbekannt*. Die Berechnung wird in solch einem Fall wieder aufgenommen und solange fortgeführt, bis ein positives Ergebnis vorliegt. Der Roboter muss somit ausnahmsweise stehen bleiben und kann erst später weiterlaufen.

2.3.4 Berechnung auf den Mikrocontrollern

Ein Anytime-Algorithmus liefert potentiell bessere Lösungen, je mehr Rechenzeit ihm zur Verfügung steht. Die Rechenzeit wird durch die Dauer der aktuellen Bewegung vorgegeben. Durch Verwendung mehrerer Prozessoren kann die gesamte Rechenleistung aber erhöht werden.

Der LAURON III verfügt über insgesamt acht Prozessoren – sieben Mikrocontroller und der des PC/104. Die Mikrocontroller werden lediglich zu Steuerung der Beine verwendet. Damit sind sie aber kaum ausgelastet und können zusätzlich zur Planung genutzt werden. Durch *Parallelisierung* eines Algorithmus kann die zur Verfügung stehende Rechenkapazität zusätzlich verwendet werden.

Entscheidend ist dabei die *Parallelisierbarkeit* des Algorithmus. So müssen Bereiche im Code existieren, die nicht auf Ergebnisse von vorhergegangenen Berechnungen zurückgreifen. Außerdem dürfen die Daten, auf die gleichzeitig zugegriffen wird, nicht zu groß sein. Sonst

ist eventuell zu viel Kommunikation zwischen den Prozessoren nötig, was wiederum eine Verschlechterung der Laufzeit bewirkt.

Eine zusätzliche Anforderung entsteht durch die Verwendung der Mikrocontroller. Sie verfügen lediglich über 512 KByte RAM. Was für einen Embedded Prozessor sehr viel ist, kann für einige komplexe Algorithmen jedoch nicht ausreichen. Es ist also auch auf den Speicherbedarf zu achten.

3 Laufplanung als Optimierungsproblem

In Abschnitt 2.3.1 wurde festgestellt, dass die Betrachtung der Laufplanung als Optimierungsproblem sinnvoll ist. Es ist dazu allerdings notwendig, die Laufplanung in der entsprechenden Form als Optimierungsproblem zu beschreiben. Dies geschieht im folgenden Kapitel. Zunächst wird dargestellt, wie ein Optimierungsproblem formal beschrieben wird. Anschließend werden die einzelnen Teile, die Eingabe, der Lösungsraum und die Bewertungsfunktion, für das Laufplanungsproblem definiert.

3.1 Optimierungsprobleme

Ein *Optimierungsproblem* ist definiert durch ein Paar (S, ϕ). S ist die Menge der möglichen Lösungen und ϕ die Bewertungsfunktion

$$\phi : S \mapsto \mathbb{R}. \tag{3.1}$$

Bei einem *Maximierungsproblem* gilt: Gesucht wird ein s_{opt} mit

$$s_{\text{opt}} \in S, \phi(s_{\text{opt}}) = \max_{s \in S} \phi(s). \tag{3.2}$$

Für ein *Minimierungsproblem* lautet es entsprechend:

$$s_{\text{opt}} \in S, \phi(s_{\text{opt}}) = \min_{s \in S} \phi(s). \tag{3.3}$$

Gesucht wird also ein Element des Lösungsraums mit entsprechend der Bewertungsfunktion optimaler Bewertung. Da sich durch Negation der Bewertungsfunktion jedes Minimierungsproblem in ein äquivalentes Maximierungsproblem transformieren lässt, wird in dieser Arbeit stets von Maximierungsproblemen ausgegangen.

Meist wird der Lösungsraum S nicht explizit angegeben, sondern über einschränkende Bedingungen als Teil einer Obermenge $U \supseteq S$ beschrieben. Damit ist es erst einmal notwendig, gültige Lösungen $s \in S$ zu bestimmen. Zusätzlich wird über diesen noch das Optimum bestimmt. Für viele Probleme ist bereits das Finden einer zulässigen Lösung schwierig. Durch die zusätzliche Einschränkung der Optimalität lassen sich die Problemstellungen nicht einfacher, meist sogar noch schwieriger zu lösen.

3.2 Eingabe

Der *Lösungsraum* ist nicht konstant, sondern wird über eine *Eingabe* bestimmt. Im Fall des hier vorliegenden Laufplanungsproblems sind dies:

- die Startposition,
- die Zielposition,
- die Umgebungsinformationen.

Die Start- und die Zielposition geben an, wohin sich der Roboter ausgehend vom Start bewegen soll. Die Angaben beziehen sich dabei auf den Robotermittelpunkt – ein festgelegter Bezugspunkt auf dem Roboter. Sie werden als zweidimensionale Koordinaten in einem vorgegebenen Weltkoordinatensystem dargestellt.

Die Umgebungsinformationen dienen der Beschreibung der Umwelt. Durch diese Informationen ist der Roboter erst in der Lage, im Voraus eine geeignete Bewegung zu planen. Sie sollten in einer abstrakten Datenstruktur verwaltet werden, die es gestattet, unterschiedliche Quellen zur Lieferung der Daten zu verwenden. Dadurch wird eine gewisse Unabhängigkeit von den verwendeten Sensoren erreicht. Zusätzlich muss aber auch die geringe Speicherausstattung der Mikrocontroller berücksichtigt werden. Die Datenstruktur sollte daher möglichst kompakt sein.

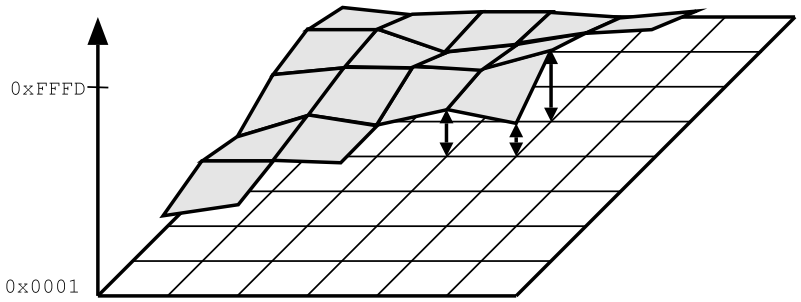


Abbildung 3.1: Terrain Map

In dieser Arbeit wird eine im Weiteren als *Terrain Map* bezeichnete Struktur verwendet. Wie in Abbildung 3.1 dargestellt, wird die Umgebung in äquidistante Bereiche unterteilt. An den Eckpunkten wird jeweils ein Höhenwert abgetragen. Die Größe der Bereiche richtet sich nach der gewünschten Auflösung. Hier wurde eine Kantenlänge von 100 mm gewählt. Als Bezug dient der Roboter, der in der Mitte des Feldes platziert wird. Pro Eckpunkt wird ein ganzzahliger 16-Bit-Wert als Höheninformation verwendet. Die Anzahl der Eckpunkte richtet sich nach dem Speicherplatz des Mikrocontrollers. Hier sind es 256×256 Werte, was zu einem Speicherbedarf von 131 072 Byte oder 128 KByte führt.

In den 16 Bit der Eckpunkte werden die Informationen zu den jeweiligen Koordinaten gespeichert. Damit stehen 65 536 Werte zur Verfügung. Diese werden auf einen geeigneten

Zahlenbereich abgebildet, der die Höhe repräsentiert. Werte außerhalb des Zahlenbereichs können nicht dargestellt werden. Um spezielle Informationen zu hinterlegen, werden einige Zahlenwerte reserviert und gesondert interpretiert. Diese sind:

Wert	*Bedeutung*	*Beschreibung*
0x0000	$-\infty$	Der Höhenwert ist kleiner als 0x0001.
0xFFFE	∞	Der Höhenwert ist größer als 0xFFFD.
0xFFFF	`undefined`	Es liegen keine Informationen vor.

Die Werte $-\infty$ und ∞ werden verwendet, wenn die Höhe an den jeweiligen Stellen nicht über den gewählten Zahlenbereich dargestellt werden kann. Die Interpretation für $-\infty$ ist eine Art „Loch". Der Bereich darf überquert werden, ein Betreten ist aber nicht möglich. Dem Wert ∞ kommt die Bedeutung einer Wand am nächsten. Dieser Bereich kann weder betreten noch überquert werden. Der Wert `undefined` wird immer angenommen, wenn keine Informationen über den Bereich vorliegen. Hierdurch wird es möglich, auch unvollständige Sensorinformationen zur Planung zu verwenden.

Die Zuordnung von Eckpunkten zu globalen Koordinaten erfolgt über die Position des Roboters $\vec{r}$. Dieser befindet sich immer in der Mitte des Koordinatensystems, also bei 256 $\times$ 256 Werten auf der Position (128, 128). Damit lässt sich die Weltkoordinate eines Eckpunktes (i, j) mit

$$pos(i, j) = (\vec{r}_x + (i - 128), \vec{r}_y + (j - 128)) \tag{3.4}$$

berechnen.

3.3 Lösungsraum

Der Lösungsraum stellt die Bewegungen des Roboters über der Zeit dar. Dabei ist darauf zu achten, dass wirklich *alle* möglichen Bewegungen enthalten sind und nicht bereits durch die Darstellung bestimmte Lösungen ausgeschlossen werden.

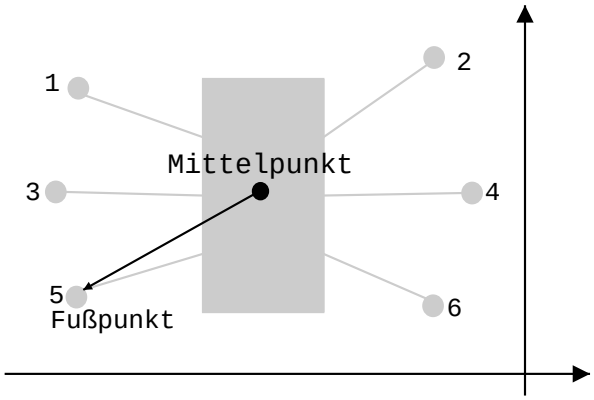


Abbildung 3.2: Roboterkoordinaten

Die Beschreibung der Roboterbewegung geschieht, wie in Abbildung 3.2 dargestellt, über den *Robotermittelpunkt* und die *Fußpunkte*. Der Mittelpunkt ist ein zweidimensionaler Vektor, der als absolute x- bzw. y-Koordinate im Weltkoordinatensystem zu interpretieren ist. Die Fußpunkte dagegen sind zweidimensionale Koordinaten relativ zum Mittelpunkt. Durch eine einfache Addition von Fußpunkt und Mittelpunkt lässt sich aber auch die absolute Position bestimmen. Eine Angabe der z-Koordinate ist nicht nötig, da sie aus der Terrain Map abgeleitet werden kann.

Zur Modellierung eines zeitlichen Verlaufs wurde in dieser Arbeit eine ereignisbasierte Darstellung gewählt. Dies hat gewisse Vorteile gegenüber einer zeitdiskreten Variante. Die Zeit kann kontinuierlich dargestellt werden. Außerdem wird weniger Speicher benötigt, da die Anzahl der Ereignisse viel geringer ist als die Zahl der diskreten Zeitabschnitte.

Ein *Ereignis* beschreibt eine Bewegung von Körper oder Fußpunkt über:

- den relativen *Startzeitpunkt* zum vorherigen Ereignis (vgl. Abbildung 3.3),
- die vorgesehene *Dauer* der Bewegung,
- das *Ziel* der Bewegung.

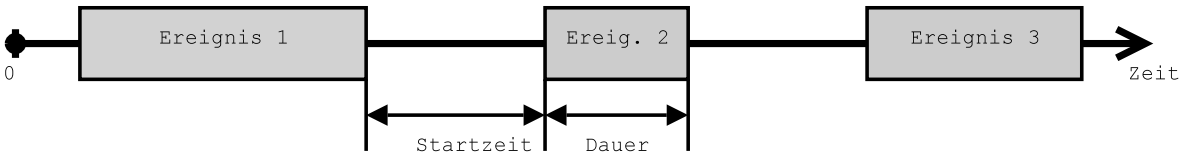


Abbildung 3.3: Ereignisse

Sowohl für den Mittelpunkt als auch für die sechs Fußpunkte wird eine *Ereignisliste* verwaltet. Aus dieser Liste lässt sich zu einem beliebigen Zeitpunkt die entsprechende Position ableiten.

3.3.1 Mittelpunkt-Ereignis

Ein Ereignis in der Mittelpunkt-Liste repräsentiert eine Bewegung des Roboterkörpers. Der Startzeitpunkt bestimmt sich dabei relativ zum vorhergehenden Ereignis beziehungsweise für das erste Ereignis zum Zeitpunkt 0. Die Bewegung erfolgt in Richtung Zielpunkt und ist genau nach der über die Dauer angegebenen Zeit abgeschlossen.

Das Verhalten der Fußpunkte während eines Mittelpunkt-Ereignisses hängt davon ab, ob sie gerade selbst ein Ereignis ausführen. Es gibt damit genau zwei Fälle:

Der Fußpunkt führt ein Ereignis aus. Wenn dies der Fall ist, wird das Verhalten ausschließlich von dem Fußpunkt-Ereignis bestimmt.

Der Fußpunkt führt kein Ereignis aus. In diesem Fall befindet sich der Fuß auf dem Untergrund und stützt den Roboter. Durch eine Fußbewegung entgegen der Roboterbewegung schiebt der Fuß den Roboter in Richtung Ziel. Führt also der Mittelpunkt zwischen zwei Zeitpunkten eine Verschiebung um den Vektor $\vec{m}_\Delta$ aus, bewirken alle stützenden Fußpunkte eine Verschiebung um $\vec{f}_\Delta = -\vec{m}_\Delta$.

Dabei kann es zu nicht zulässigem Verhalten kommen. Die Fußpunkte können durch die physikalische Reichweite der Roboterbeine nur in bestimmten Bereichen liegen. Ein Verlassen dieser Bereiche ist nicht möglich und muss beim Erzeugen der Ereignisfolgen vermieden werden. Dazu muss rechtzeitig ein Fußpunkt-Ereignis auftreten, das zu einem Verlassen des Stützverhaltens führt.

3.3.2 Fußpunkt-Ereignis

Ein Fußpunkt-Ereignis ist für das Umsetzen eines Fußes auf eine andere Position zuständig. Auch hier bestimmt sich der Startzeitpunkt relativ zum vorhergehenden Ereignis bzw. zum Zeitpunkt 0. Die Zielposition ist aber ein Vektor relativ zum Robotermittelpunkt. Während des Ereignisses stützt der Fuß den Roboter nicht, sondern bewegt sich auf einer entsprechenden Bahn zum Zielpunkt. Die Dauer des Vorgangs entspricht dabei der im Ereignis angegebenen. Anschließend befindet sich der Fußpunkt wieder auf dem Untergrund und stützt den Roboter.

Auch hierbei können unzulässige Zustände auftreten. Die Zielkoordinate könnte außerhalb des erreichbaren Bereichs liegen. Es ist ferner möglich, dass zu einem Zeitpunkt weniger als drei Fußpunkte den Körper stützen. Dadurch wäre er nicht mehr stabil und könnte kippen. Auch dies sollte beim Erzeugen der Ereignisfolge verhindert werden.

Wie sich der Fußpunkt während eines Ereignisses bewegt, wird hier nicht betrachtet. Dies ist Aufgabe der darunter liegenden Schicht, der Beinsteuerung. Es ist allerdings möglich, dass der Roboter die Bewegung nicht innerhalb der geforderten Dauer ausführen kann. Zu diesem Zweck wird ein Modell der Beinsteuerung benötigt, anhand dessen die Zulässigkeit der Bewegung überprüft wird.

3.3.3 Beispiel Ereignislisten

Zur Verdeutlichung soll der so genannte *Dreifußgang* über Ereignislisten dargestellt werden. Der Dreifußgang ist ein Laufmuster für Sechsbeiner. Er erlaubt eine Bewegung auf ebenem Gelände in Vorwärtsrichtung.

Beim Dreifußgang werden jeweils drei Beine zum Stützen des Roboterkörpers verwendet, während sich die verbleibenden drei vorwärts bewegen. Sobald die Vorwärtsbewegung der Beine beendet ist, wechseln die Tripel ihre Funktion. Nun stützen die vorher nach vorn gesetzten Beine, und die ehemaligen Stützbeine führen die Vorwärtsbewegung aus. Dies wird solange wiederholt, bis das Ziel erreicht ist. Es wurde gezeigt [36, 42], dass der Dreifußgang für ein gerades Vorwärtslaufen auf ebenem Untergrund bei periodischer Ausführung eine optimale Lösung hinsichtlich der Geschwindigkeit darstellt.

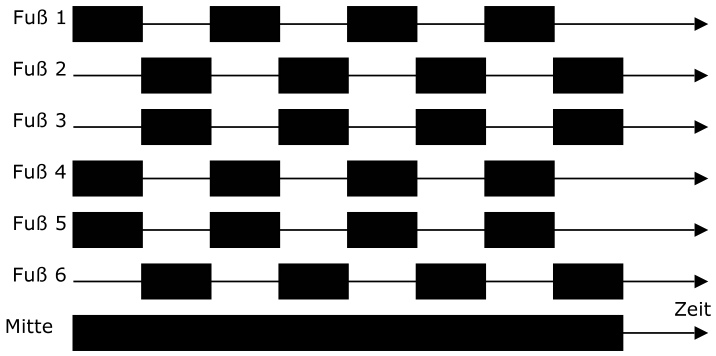


Abbildung 3.4: Beispiel Dreifußgang

Die entsprechenden Ereignislisten sind in Abbildung 3.4 dargestellt. Die Zielpositionen der Ereignisse sind nicht enthalten. Für den Mittelpunkt ist dies die Position des Ziels. Die Fußpunkte werden immer in der Mitte des möglichen Bereichs platziert. Der Robotermittelpunkt führt permanent ein Ereignis aus, nämlich die Vorwärtsbewegung. Die Fußpunkte 1, 4 und 5 führen ein Ereignis aus, welches dem Vorwärtssetzen entspricht. Die anderen drei Fußpunkte verbleiben im Stützverhalten. Nach einer gewissen Zeit tauschen die Tripel ihre Funktion, indem die Ereignisse der Fußpunkte 1, 4 und 5 enden und die Fußpunkte 2, 3 und 6 neue Ereignisse starten. Der Vorgang wiederholt sich, bis die Bewegung des Roboters beendet ist.

3.4 Bewertungsfunktion

Die *Bewertungsfunktion*

$$\phi : \mathrm{S} \mapsto \mathbb{R} \tag{3.5}$$

dient der Bewertung einer Lösung durch Abbildung auf einen Zahlenwert. Durch die Darstellung als Zahlenwert ist ein direkter Vergleich der Güte mehrerer Lösungen möglich. Dabei gilt bei einem Maximierungsproblem: Je höher der Wert desto besser die Lösung. Eine Lösung $s_1 \in \mathrm{S}$ ist genau dann besser als $s_2 \in \mathrm{S}$, wenn $\phi(s_1) > \phi(s_2)$.

Hier dient die Bewertungsfunktion der Beurteilung einer Laufbewegung. Dazu müssen unterschiedliche Kriterien betrachtet werden, die die Güte einer Lösung beschreiben. Die Güte ist kein rein objektiver Begriff, sondern hängt stark von der Anwendung ab. Einzelne Aspekte können einen mehr oder weniger großen Einfluss auf die Güte haben. Um dem gerecht zu werden, wird die Bewertungsfunktion aus einzelnen Teiltermen zusammengesetzt, welche die jeweiligen Aspekte repräsentieren:

$$\phi(s) = \lambda_1\phi_1(s) + \lambda_2\phi_2(s) + \ldots + \lambda_n\phi_n(s). \tag{3.6}$$

Über die Faktoren λ_i lassen sich die einzelnen Aspekte gewichten. Dies ermöglicht eine Anpassung der Funktion an unterschiedliche Bedürfnisse. Im Weiteren werden einige Kriterien vorgestellt.

3.4.1 Dauer der Bewegung

Wohl eines der wichtigsten Kriterien für die Güte einer Bewegung ist ihre Dauer. Diese Größe entspricht der Zeit, die vom Start der Laufbewegung bis zum Erreichen des Ziels vergeht. Auch für nicht zeitkritische Bewegungen ist es sinnvoll, dass die Dauer betrachtet wird. Sonst sind Lösungen möglich, die unverhältnismäßig viel Zeit benötigen. Eine Bewegung, bei der der Roboter eine lange Zeit einfach still stehen bleibt, hätte die gleiche Bewertung, wie die entsprechende Bewegung ohne Pause. Ein solches Verhalten ist in keinem Fall gewünscht und wird durch die Bewertung der Dauer mit einem niedrigen Wert bestraft.

Zum Ermitteln des Funktionswertes muss die Dauer der Bewegung berechnet werden. Dies ist sehr einfach möglich, indem von allen Mittelpunktereignissen die Dauer und Startzeit addiert werden:

Algorithmus 1 Berechnung der Gesamtdauer t_{dur}

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

Da die Bewertung für die Dauer größer sein soll, je besser die Lösung ist, kann t_{dur} nicht direkt verwendet werden. Deshalb ist mindestens eine Negation nötig.

$$\phi_{-\mathrm{dur}} = -t_{\mathrm{dur}} \tag{3.7}$$

Dieser Term besitzt die gewünschte Eigenschaft. Ein Nachteil ist jedoch, dass die mögliche Dauer auch von der Entfernung des Zielpunktes abhängt. Somit ist es schwer, den Wert ins Verhältnis zu den anderen Bewertungstermen zu setzen. Eine Normalisierung erscheint daher erstrebenswert. So könnte die zurückgelegte Strecke durch die Dauer dividieren werden, was der Geschwindigkeit entspricht. Dies würde allerdings auch unnötig lange Wege bevorzugen, die zwar eine hohe Durchschnittsgeschwindigkeit, aber auch eine hohe Gesamtdauer besitzen.

Als sinnvolle Lösung erscheint hier, nicht die zurückgelegte Strecke, sondern die Entfernung des Zielpunktes vom Startpunkt zu verwenden. Dem entspricht die Geschwindigkeit zum Ziel ohne Umwege. Damit ergibt sich der endgültige Term als:

$$\phi_{\mathrm{dur}} = \frac{|\vec{p}_{\mathrm{ziel}} - \vec{p}_{\mathrm{start}}|}{t_{\mathrm{dur}}}. \tag{3.8}$$

3.4.2 Kippstabilität

Ein weiterer wichtiger Aspekt bei der Bewertung einer Bewegung ist die Stabilität des Roboters. Ist diese nicht gewahrt, kann es zum Wegrutschen oder Kippen kommen. Daraus können Beschädigungen am Roboter selbst und auch an der transportierten Last resultieren. Für Instabilitäten kann es unterschiedliche Gründe geben. Hier werden die Kippstabilität und die Untergrundstabilität betrachtet.

Der Roboterkörper wird während der Laufbewegung von den Beinen gestützt. Gleichzeitig dienen diese aber auch der Fortbewegung. Damit erfüllen sie zwei Aufgaben und es ist Ziel der Bewegungsplanung zu entscheiden, welcher Fußpunkt zu welchem Zeitpunkt welche Aufgabe übernimmt. Bei Fehlentscheidungen kann es zum „Kippen" kommen.

Ein einfaches Beispiel ist die Regel, dass immer mindestens drei Fußpunkte den Körper stützen müssen. Dadurch kann ein stabiler Stand erzeugt werden. Sind zu einem Zeitpunkt aber drei Fußpunkte auf der gleichen Seite des Körpers ausgewählt, kippt der Roboter zur anderen Seite weg.

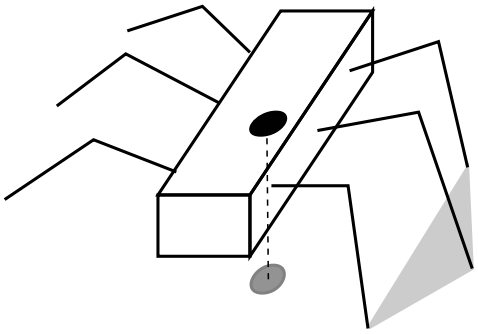


Abbildung 3.5: Kippen

Der Grund ist, dass der Massenschwerpunkt des Roboters außerhalb des von den Fußpunkten aufgespannten Dreiecks liegt. Genauer gesagt: Werden der Massenschwerpunkt und die stützenden Fußpunkte entlang des Gravitationsvektors auf eine darunter liegende Ebene projiziert, muss mindestens ein von den Fußpunktprojektionen aufgespanntes Dreieck existieren, das den projizierten Massenschwerpunkt enthält [44]. In Abbildung 3.5 ist dies für den Fall der drei Fußpunkte auf der gleichen Seite dargestellt. Der Massenschwerpunkt liegt nicht im von den Fußpunkten aufgespannten Dreieck. Deshalb kippt der Roboter.

Durch diese Regel erhält man eine Möglichkeit zu bestimmen, ob der Roboter kippt. Allerdings ist eine reine Ja-Nein-Aussage sehr empfindlich gegenüber Störungen. Auch wenn der Massenschwerpunkt dicht am Rand des stützenden Dreiecks liegt, steht der Roboter stabil. Jedoch können schon leichte Schwankungen eine Änderung der Situation hervorrufen und ein Kippen des Roboters verursachen.

Man beurteilt daher nicht nur, ob der Roboter stabil steht, sondern auch wie stabil er steht. Dies entspricht gerade dem Abstand des Massenschwerpunktes von dem Bereich, in dem alle stützenden Dreiecke verlassen sind und das Kippen einsetzt. Man nennt diesen Wert *Stability Margin* [42].

Der Stability Margin ließe sich prinzipiell berechnen, indem man den maximalen Wert betreffend aller aufgespannten Stützdreiecke bildet. Es geht aber auch einfacher, indem man statt der einzelnen Dreiecke die Vereinigung dieser betrachtet. Sie entspricht gerade der konvexen Hülle aller stützenden Fußpunkte. Die konvexe Hülle lässt sich sehr leicht berechnen [29] und damit auch der Wert des Stability Margin, wie in Abbildung 3.6 dargestellt. Es muss lediglich der Abstand des Massenschwerpunktes vom Rand der konvexen Hülle ermittelt werden, was mit einfachen geometrischen Berechnungen gelingt.

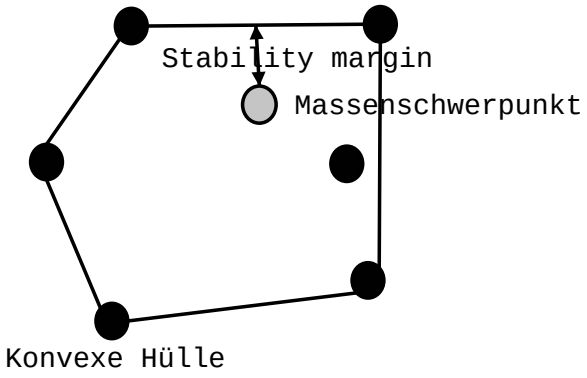


Abbildung 3.6: Stability Margin

Für die Berechnung wird der Massenschwerpunkt benötigt. Dieser Punkt hängt von der Masseverteilung im Roboter ab. Seine Position bestimmt sich relativ zum Bezugspunkt des Roboters, also dem Mittelpunkt. Da die Position des Mittelpunktes frei gewählt werden kann, soll zur Vereinfachung davon ausgegangen werden, dass der Massenschwerpunkt gleich dem Mittelpunkt ist. Durch Bewegungen der Beine werden auch deren Massen verschoben. Dies führt zu einer Verschiebung des Massenschwerpunktes. Simulationen [21] haben allerdings ergeben, dass die Abweichungen vernachlässigbar gering sind. Der Massenschwerpunkt darf daher als konstant angenommen werden und mit dem Mittelpunkt identisch sein.

Mit den bisherigen Möglichkeiten kann der Stability Margin für einen konkreten Zeitpunkt ermittelt werden. Für die Bewertung der Laufbewegung ist es jedoch nötig, den minimalen Wert für die gesamte Dauer zu ermitteln. Da die Zeit kontinuierlich ist, kann nicht zu jedem möglichen Zeitpunkt der Wert berechnet werden. Daher werden bestimmte Zeitpunkte ermittelt, an denen der Stability Margin den minimalen Wert annehmen kann.

Für die Änderung des Stability Margin kann es genau zwei Ursachen geben: Die konvexe Hülle ändert sich oder der Mittelpunkt bewegt sich. Die konvexe Hülle ändert sich, wenn Fußpunkte hinzukommen oder entfernt werden. Durch Hinzufügen eines Fußpunktes wird die konvexe Hülle höchstens größer. Damit kann der Stability Margin ebenfalls nur größer werden und braucht somit für das Minimum nicht betrachtet werden. Beim Anheben eines Fußpunktes wird die konvexe Hülle möglicherweise kleiner. Hier muss überprüft werden, ob ein neues Minimum vorliegt. Das Anheben eines Fußpunktes entspricht gerade dem Startzeitpunkt eines Fußpunkt-Ereignisses.

Der zweite Fall ist das Verschieben des Mittelpunktes innerhalb der konvexen Hülle. Dies geschieht durch ein Mittelpunkt-Ereignis. Aufgrund der Konvexität und der linearen Bewegung kann während der Bewegung kein Wert kleiner als am Anfang oder Ende angenommen werden. Damit genügt eine Betrachtung dieser Zeitpunkte. Da die Position des Mittelpunktes am Start eines Ereignisses gleich der Position eines vorherigen Ereignisses ist, reicht es aus, das Ende der Ereignisse zu betrachten. Eine Ausnahme bildet nur das erste Mittelpunkt-Ereignis. Hier wird die Startposition zum Zeitpunkt 0 verwendet.

Es gibt also insgesamt drei Fälle, in denen ein minimaler Stability Margin angenommen

Algorithmus 2 Berechnung des minimalen Stability Margin ϕ_{stab}

```
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 ∈ FootEvents_i do
    t := startTime(e)
    smargin := min{ smargin, stabilityMargin(t) }
  end for
end for
```

werden kann:

1. zum Beginn der Bewegung,
2. zum Ende eines Mittelpunkt-Ereignisses,
3. zum Beginn eines Fußpunkt-Ereignisses.

Damit lässt sich der minimale Stability Margin wie in Algorithmus 2 berechnen. Der Wert besitzt auch die für ein Maximierungsproblem nötige Eigenschaft, dass er für bessere Bewegungen größer ist. Er kann daher direkt als Bewertungsterm ϕ_{stab} genutzt werden.

3.4.3 Untergrundstabilität

Die Stabilität des Roboters hängt nicht allein von der Stellung der Beine ab, sondern auch von der Beschaffenheit des Untergrundes. Dieser muss fest genug sein, um einen Fußpunkt zu stützen. Zudem muss die Oberfläche möglichst horizontal sein, um ein Wegrutschen zu verhindern.

Die Festigkeit des Untergrundes vor dem Aufsetzen zu bewerten ist äußerst schwierig, weshalb es in dieser Arbeit nicht betrachtet wird. Die Ebenheit des Untergrundes ist dagegen leicht zu ermitteln, wenn ein Höhenprofil vorliegt.

Wie bereits im Abschnitt 3.2 beschrieben, existiert ein Höhenprofil in Form der Terrain Map. Damit ist es möglich, einem bestimmten Punkt p vier umgebende Höhenwerte $U(p)$ zuzuordnen.

$$U(p) = \left\{ \begin{pmatrix} \lfloor p_x \rfloor \\ \lfloor p_y \rfloor \end{pmatrix}, \begin{pmatrix} \lceil p_x \rceil \\ \lfloor p_y \rfloor \end{pmatrix}, \begin{pmatrix} \lfloor p_x \rfloor \\ \lceil p_y \rceil \end{pmatrix}, \begin{pmatrix} \lceil p_x \rceil \\ \lceil p_y \rceil \end{pmatrix} \right\} \tag{3.9}$$

Aus diesen vier Werten lässt sich die maximale Höhendifferenz $s(p)$ berechnen.

$$\mathrm{s}(p) = \max_{u \in U(p)} T_{u_x,u_y} - \min_{u \in U(p)} T_{u_x,u_y} \tag{3.10}$$

Sie stellt einen repräsentativen Wert für den Anstieg im Punkt p dar.

Für die Bewertungsfunktion wird ein Term benötigt, der gute Lösungen mit hohen Bewertungen versieht und schlechte mit entsprechend niedrigen. Die Funktion s besitzt diese Eigenschaften noch nicht. Deshalb müssen die Werte noch geeignet umgerechnet werden.

Ein stabiler Stand ist bis zu einem gewissen Anstieg $s_{\min}$ immer gegeben. Es gibt zudem einen Wert $s_{\max}$, ab dem ein Wegrutschen unvermeidbar ist. Dazwischen findet ein zunehmender Verlust der Stabilität mit größer werdendem Anstieg statt. Eine Bewertung kann über die Funktion

$$\sigma(p) = \begin{cases} 0 & : \quad \mathrm{s}(p) < s_{\min} \\ -\mathrm{s}(p) & : \quad s_{\min} < \mathrm{s}(p) < s_{\max} \\ -\mathrm{M} & : \quad s_{\max} < \mathrm{s}(p) \end{cases} \tag{3.11}$$

erfolgen. Die Konstante $-M$ ist eine sehr kleine negative Zahl. Tritt der zugehörige Fall $s_{\max} < \mathrm{s}(p)$ auf, sollte die Bewertung so schlecht werden, dass die Lösung nie optimal ist. Hierdurch erfolgt ein indirektes Ausschließen dieser unzulässigen Lösung.

Für eine Gesamtbewertung der Bewegung wird die minimale Stabilität ermittelt. Die Ziele aller Fußpunkt-Ereignisse enthalten die entsprechenden Positionen, so dass sich der Wert sehr einfach, wie in Algorithmus 3, berechnen lässt.

Algorithmus 3 Berechnung der minimalen Untergrundstabilität

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

3.4.4 Zulässigkeit der Lösung

Bereits in Abschnitt 3.3 wurde erwähnt, dass einige darstellbare Lösungen nicht zulässig sind. Mögliche Gründe sind:

1. Ein Fußpunkt bewegt sich, während er den Körper stützt, außerhalb seiner physikalischen Reichweite.
2. Das Ziel eines Fußpunkt-Ereignisses liegt außerhalb der physikalischen Reichweite.
3. Die gewünschte Dauer einer Bewegung kann von der Beinsteuerung nicht erbracht werden.
4. Eine Kollision zwischen zwei Beinen tritt auf.

Idealerweise werden unzulässige Lösungen bei den entsprechenden Verfahren gar nicht erst betrachtet. Es ist für einige Verfahren aber empfehlenswert, auch unzulässige Lösungen vorerst zuzulassen. Der Ausschluss findet dann nicht über den Lösungsraum, sondern über die Bewertungsfunktion statt. Einer unzulässigen Lösung wird eine extrem schlechte Bewertung gegeben. Damit kommt sie nicht mehr als Optimum in Frage. Die negative Bewertung wird meist als *Strafterm* bezeichnet. Die oben genannten Fälle müssen also überprüft und entsprechend behandelt werden.

Stützfuß außerhalb des möglichen Bereichs

Die Position eines stützenden Fußpunktes lässt sich sehr einfach aus der Bewegung des Mittelpunktes ableiten. Nachdem der Fuß aufgesetzt hat, bewegt er sich im globalen Koordinatensystem nicht mehr. Wenn der Mittelpunkt verschoben wird, bewegt er sich aber im lokalen Bezugssystem des Roboters, und zwar genau entgegengesetzt zum Mittelpunkt.

Um nun zu berechnen, wie weit sich der Fußpunkt seit dem letzten Aufsetzen bewegt hat, muss nur die Differenz der Mittelpunktposition beim Aufsetzen und zum aktuellen Zeitpunkt berechnet werden. Addiert zu der alten Fußposition erhält man den gesuchten Wert. Dieser kann auf Zulässigkeit hinsichtlich der physikalischen Reichweite des Roboterbeins geprüft werden.

Ähnlich der Berechnung des Stability Margin müssen Zeitpunkte gefunden werden, die repräsentativ für die Gesamtdauer der Bewegung sind. Die kritischen Zeitpunkte sind die Start- und Endzeiten von Fußpunkt- und Mittelpunkt-Ereignissen. Die Endzeit eines Fußpunkt-Ereignisses wird im folgenden Fall bereits behandelt. Die Startposition eines Mittelpunkt-Ereignisses entspricht der Endposition des vorherigen. Somit genügt eine Betrachtung eines der beiden. Die Zulässigkeit muss also immer zu Beginn eines Fußpunkt- oder Mittelpunkt-Ereignisses getestet werden.

Setzen eines Fußes außerhalb des möglichen Bereichs

Die Zielposition eines Fußpunkt-Ereignisses ist in der Datenstruktur auf keine Weise beschränkt. Es können auch Werte außerhalb der physikalischen Reichweite eines Beins angegeben werden. Dies würde aber zu einer unzulässigen Bewegung führen. Die Endposition eines jeden Beins muss also auf Zulässigkeit getestet werden. Dies ist für alle Fußpunkt-Ereignisse nötig.

Unzulässige Dauer einer Bewegung

Auch die Dauer eines Fußpunkt- oder Mittelpunkt-Ereignisses ist in der Datenstruktur in keiner Weise eingeschränkt. Der Roboter besitzt jedoch entsprechende Einschränkungen. Eine Position kann durch den Beinregler nicht beliebig schnell erreicht werden. Für die Bewegung der Beine gibt es eine Dauer, die mindestens benötigt wird. Diese hängt stark von der zurückzulegenden Strecke ab.

Um entscheiden zu können, ob die Dauer für die gewünschte Bewegung zulässig ist, wird ein Modell des Beinreglers benötigt. Dieses Modell stellt eine Funktion bereit, die für einen Anfangs- und Endpunkt die Mindestdauer liefert. Über den Wert kann dann die Zulässigkeit der Dauer in den einzelnen Ereignissen überprüft werden.

Die Dauer muss für alle Fußpunkt-Ereignisse geprüft werden, aber auch während der Mittelpunkt-Ereignisse. Für ein Fußpunkt-Ereignis wird der Startpunkt aus der Position des letzten Aufsetzens ermittelt. Die Endposition des letzten Fußpunkt-Ereignisses entspricht dem Punkt, ab dem sich der Fuß auf dem Boden befand. Die Differenz zwischen aktueller

Mittelpunktposition und der zum Zeitpunkt des letzten Aufsetzens entspricht der Verschiebung. Durch Addition erhält man die neue Position. Der Endpunkt und die Dauer können direkt aus dem Ereignis verwendet werden. Über diese drei Größen und die Funktion zum Ermitteln der minimalen Dauer lässt sich dann die Korrektheit der angegebenen Dauer überprüfen.

Für ein Mittelpunkt-Ereignis geschieht dies ähnlich. Hierbei müssen die Bewegungen aller stützenden Fußpunkte auf Korrektheit überprüft werden. Deshalb werden diese zuerst ermittelt. Ein Fußpunkt ist stützend, wenn er während des Mittelpunkt-Ereignisses kein eigenes Ereignis ausführt. Dies lässt sich leicht über die Zeiten prüfen. Anschließend erfolgt eine Überprüfung der Dauer wie bei einem Fußpunkt-Ereignis.

Kollision von Beinen

Die Reichweiten der Beine des LAURON III überlappen sich in bestimmten Bereichen. Damit ist eine Kollision während der Bewegung möglich. Solch ein Verhalten ist allerdings nicht zulässig und muss bei der Planung berücksichtigt werden.

Die Kinematik der Beine ist äußerst komplex und nur schwer analytisch zu beschreiben. Eine exakte Kollisionserkennung ist daher auch nur schwer möglich. Hier wurde deshalb eine Vereinfachung vorgenommen.

Für die Beine wird gefordert, dass ein Fuß stets *hinter* dem vorderen sein muss. Konkret bedeutet dies, dass sich wie in Abbildung 3.7 entlang des Körpers immer der Fußpunkt 3 hinter 1, 5 hinter 3, 4 hinter 2 und 6 hinter 4 befinden muss.

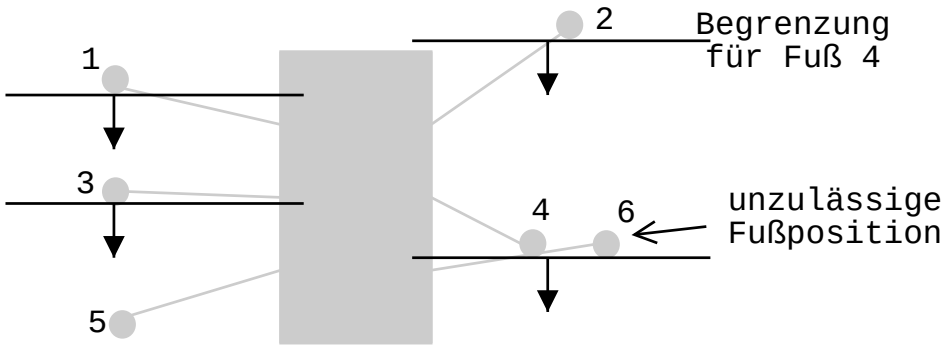


Abbildung 3.7: Ordnung der Fußpositionen

Eine solche Bedingung kann sehr leicht geprüft werden. Für die Fußpunkte wird ein Radius von 50 mm angenommen, der eine Kollision in der Praxis wirkungsvoll verhindern sollte. Durch diese Vereinfachung werden auch nur wenige Lösungen ausgeschlossen. Durch Experimente am realen Roboter zeigte sich, dass Konstellationen, bei denen diese Ordnung nicht eingehalten wird, äußerst problematisch sind. Es kann schon bei kleinen Abweichungen zu einem Verhaken der Beine kommen. Diese Fälle sind, wenn auch zulässig, nicht wünschenswert.

Das Einhalten dieser Regel muss immer dann geprüft werden, wenn ein Fuß aufsetzt. Dies ist immer am Ende eines Fußpunkt-Ereignisses der Fall. War eine Fußstellung vorher gültig, so kann sie beim Anheben nicht ungültig werden. Dieser Fall braucht deshalb nicht betrachtet werden.

Erfüllt eine Bewegung die beschriebenen Kriterien nicht, so wird sie mit einem Strafterm versehen. Dies ist bei dem hier verwendeten Maximierungsproblem eine sehr kleine negative Zahl. Die Bewegung kann damit nie eine bessere Bewertung erhalten als eine korrekte Lösung. Als Optimum sollte sie daher nicht auftreten. Wird letztlich doch eine Lösung mit Strafterm ermittelt, so wurde keine bessere, also auch keine korrekte, gefunden.

3.4.5 Weitere mögliche Kriterien

Die beschriebenen Bewertungskriterien wurden in dieser Arbeit verwendet. Natürlich sind noch weitere denkbar. Ein allgemeines Optimierungsverfahren kann nach beliebigen Bewertungen optimieren. In Betracht kommen auch:

- Gleichförmigkeit der Bewegung,
- Energieverbrauch,
- Nutzen des Weges beim Aufbau des Weltmodells,
- Fehlertoleranz beim Ausfall eines Beins,
- usw.

Diese Kriterien müssen lediglich durch einen geeigneten Bewertungsterm beschrieben werden. Anschließend können sie über eine entsprechende Gewichtung zur Bewertung hinzugezogen werden.

Das Verfahren lässt beliebige Anpassungen zu. Auch für besondere Anwendungen muss nur die Bewertung verändert werden. Denkbar ist sogar ein Umschalten zur Laufzeit. So kann auf besondere Ereignisse mit einer angepassten Laufbewegung reagiert werden.

4 Untersuchung geeigneter Heuristiken

Meist sind Optimierungsprobleme nur mit großem Rechenaufwand zu lösen. Die gilt auch für das hier vorliegende Laufplanungsproblem. In Abschnitt 2.3.2 wurde gezeigt, dass das Finden *einer beliebigen* Lösung bereits sehr schwierig ist. Soll diese auch noch optimal sein, wird das Problem nicht einfacher. Es ist also nicht möglich, zuverlässig die optimale Lösung zu bestimmen. Um dennoch Ergebnisse zu erzielen, können aber Heuristiken angewendet werden. Es wird damit zwar keine optimale, aber immer noch eine relativ gute Lösung gefunden. Dies entspricht auch der ursprünglichen Forderung.

Für viele Optimierungsprobleme sind nur exakte Lösungsalgorithmen mit exponentieller Laufzeit bekannt [20, 2, 43]. Da sich derartige Probleme aber häufig in der Praxis stellen, wurden diverse Verfahren zum heuristischen Lösen entwickelt. Eine Auswahl der prominentesten wird in diesem Kapitel darauf untersucht, inwieweit sie sich für das hier gestellte Problem eignen. Dabei werden besonders die in Abschnitt 2.3 ermittelten Anforderungen betrachtet. Eine umfassende Beschreibung der Verfahren ist in [20, 2, 43] zu finden.

4.1 Greedy Verfahren

Greedy-Verfahren gehören zu den einfacheren Lösungsansätzen. Voraussetzung für die Anwendung ist, dass sich eine Lösung aus einzelnen Teillösungen $(s_1, s_2, \ldots, s_n) = s \in S$ zusammensetzen lässt. Bei dem Verfahren werden nun nacheinander Teillösungen bestimmt. Beginnend mit der leeren Lösung wird in jedem Schritt genau ein Teil ermittelt. Hierzu dient ein lokales Kriterium, mit welchem die beste Teillösung s_i gesucht wird. Bei Greedy-Verfahren wird im Gegensatz zu Backtracking-Verfahren eine einmal gefundene Teillösung beibehalten. Eine falsche Entscheidung kann damit auch nicht rückgängig gemacht werden. Hieraus resultiert auch die Bezeichnung „greedy" - gierig.

Da nur lokal gute Teillösungen ermittelt werden, ist nicht sichergestellt, dass auch die gesamte Lösung gut bewertet ist. Die Heuristik geht aber davon aus, dass dies der Fall ist. Es ist weiterhin nicht selbstverständlich, dass überhaupt eine zulässige Lösung gefunden wird. So lässt sich nicht ausschließen, dass im i-ten Schritt keine zulässige Teillösung s_i mehr bestimmt werden kann, die zu den vorher festgelegten $s_1, \ldots, s_{i-1}$ passt. Für Greedy-Verfahren muss dies aber *immer* gewährleistet sein. Andernfalls kann nicht zuverlässig eine Lösung geliefert werden.

Parallelisierbarkeit

Das Konzept der Greedy-Verfahren basiert auf einem sequentiellen Aufbau der Lösung. Daher ist allein durch das Greedy-Konzept keine Parallelisierung möglich. Auf Grund der kurzen Laufzeit ist es auch meist nicht nötig, mehrere Prozessoren zur Verarbeitung zu verwenden.

Speicherbedarf

Da bei Greedy-Verfahren nur eine Lösung sequentiell aufgebaut wird, müssen keine größeren Datenmengen gespeichert werden. Daher ist in der Regel nur sehr wenig Speicher vonnöten.

Anytime

Die Lösung eines Greedy-Verfahrens liegt immer erst am Ende der Verarbeitung vor. Daher ist es nicht möglich, den Algorithmus vorzeitig zu Gunsten einer suboptimalen Lösung abzubrechen. Da die Verfahren normalerweise sehr kurze Laufzeiten haben, ist ein solches Vorgehen aber auch selten nötig.

Anwendbarkeit

Um ein Greedy-Verfahren auf ein Problem anzuwenden, müssen sich die Lösungen sequentiell aufbauen lassen. Ist dies nicht der Fall, scheidet dieser Ansatz von vornherein aus. Außerdem muss noch eine Heuristik zum Aufbau der Lösung gefunden werden. Insgesamt ist dieser Ansatz deshalb schlecht allgemein verwendbar.

4.2 Branch and Bound

Bei *Branch and Bound* handelt es sich um ein Verfahren, das immer die exakte Lösung liefert. Damit ist es keine Heuristik im eigentlichen Sinne. Das Verfahren ist eine modifizierte Variante des Backtracking. Es werden wie beim normalen Backtracking die Lösungen über einen Suchbaum bestimmt. In der Regel sind diese Suchbäume sehr groß, so dass sich meist exponentielle Laufzeiten ergeben.

Bei Branch and Bound wird nun versucht, die Laufzeit durch geschicktes Stutzen des Suchbaums zu verkürzen. Hierzu dient eine Schätzfunktion ψ, die einer Teillösung S_i eine maximal erreichbare Endbewertung zuordnet.

$$\psi : S_i \mapsto \mathbb{R} \tag{4.1}$$

$$\forall s = (s_1, s_2, \ldots, s_n) \in S : \psi(s_1, s_2, \ldots s_i) \geq \phi(s_1, s_2, \ldots, s_n) \tag{4.2}$$

Ist in einem Schritt $\psi(s_i)$ kleiner als die Bewertung der besten bisher gefundenen Lösung, so kann anhand dieser Teillösung keine bessere Lösung gefunden werden. Die Suche wird daher in diesem Zweig unterbrochen, und mit einer anderen Teillösung fortgesetzt. Der Suchbaum wird also durch den Funktionswert der besten gefundenen Lösung begrenzt (bound).

Die Funktion ψ ist von entscheidender Bedeutung für die Qualität des Branch and Bound. Der Funktionswert sollte so niedrig wie möglich sein, damit der Baum möglichst früh gestutzt wird. Das Finden einer solchen Funktion verlangt meist tiefgehende Kenntnisse über die Beschaffenheit des Problems.

Ein starkes Stutzen des Suchbaums kann auch erreicht werden, indem möglichst früh eine gute Lösung gefunden wird. Dies geschieht, indem in einem Schritt geschickt der nächste zu verfolgende Zweig gewählt wird. Hierzu dient, wie schon bei Greedy-Verfahren, eine Heuristik zur Bestimmung der nächsten Teillösung. Alle möglichen Teillösungen werden gemäß der Heuristik bewertet, sortiert und beginnend mit der besten abgearbeitet.

Branch and Bound hat eine Worst-Case-Laufzeit, die der eines Backtracking entspricht. Dies kann aber viel zu lange für praktische Anwendungen sein. Daher gibt es oft die Möglichkeit, die Berechnung vorzeitig abzubrechen. Die beste bisher gefundene Lösung wird dann zurückgegeben. Da diese meist suboptimal ist, handelt es sich hiermit wieder um eine Heuristik.

Parallelisierbarkeit

Branch and Bound lässt sich gut parallelisieren. Hierzu wird auf jedem Prozessor ein Zweig verfolgt. Kommunikation tritt nur beim Austausch der Bewertung der besten bisher gefundenen Lösung auf. Für die Korrektheit des Verfahrens ist es auch nicht notwendig, dass dies in jedem einzelnen Schritt geschieht. Der Kommunikationsaufwand kann damit entsprechend angepasst werden.

Speicherbedarf

Bei dem Branch-and-Bound-Verfahren wird immer nur eine Lösung zu einem Zeitpunkt betrachtet. Dazu werden keine großen Speichermengen benötigt. Das Verfahren selbst bedingt somit keinen hohen Speicherbedarf. Lediglich die Schätzfunktion oder die Verzweigungsheuristik könnten mehr Speicher benötigen, was aber meist nicht der Fall ist. Der Speicherbedarf ist daher in der Regel sehr gering.

Anytime

Die Laufzeit von Branch and Bound wird durch Heuristiken beschränkt. Im schlechtesten Fall ist sie immer noch sehr groß. Bei dem Verfahren liegt jedoch sehr schnell eine gültige Lösung vor, die im Verlauf nur noch verbessert wird. Es kann daher jederzeit abgebrochen werden, wobei die beste bisher gefundene Lösung verwendet wird.

Anwendbarkeit

Zur Anwendung von Branch and Bound benötigt man sowohl eine Verzweigungsheuristik als auch eine Schätzfunktion ψ. Während sich eine Heuristik für die Verzweigung oft leicht finden lässt, ist dies für die Schätzfunktion ψ nicht der Fall. Triviale Funktionen lassen sich zwar konstruieren, nur führen diese nicht zu dem gewünschten frühen Stutzen des Suchbaums. Daher kann Branch and Bound nicht ohne weiteres für beliebige Probleme benutzt werden.

4.3 Random Sampling

Die wohl einfachste Heuristik zum Lösen von Optimierungsproblemen ist das *Random Sampling*. Die Idee ist ziemlich simpel und auch sehr leicht umzusetzen: Es werden große Mengen von zufälligen Lösungen generiert. Das entsprechend der Bewertungsfunktion beste Ergebnis wird übernommen.

Algorithmus 4 Random Sampling

htb!

```
best := random solution
while not good enough do
   s := random solution
   if  φ(s) > φ(best)  then
      best := s
   end if
end while
return best
```

Dieser scheinbar triviale Ansatz kann zu guten Ergebnissen führen. In einem einzelnen Schritt wird einfach nur eine Lösung aufgebaut. Da dies sehr schnell geht, ist es möglich, in kurzer Zeit sehr viele Lösungen zu generieren. Hierdurch wird der Lösungsraum intensiv durchsucht.

Besonders gut funktioniert der Ansatz, wenn viele gute Lösungen im Lösungsraum vorhanden sind. Dies erhöht die Wahrscheinlichkeit, dass eine solche tatsächlich gefunden wird.

Die Wahrscheinlichkeit einer guten Lösung lässt sich berechnen, wenn die Größe der Menge der guten Lösungen S_{good} bekannt ist. Bei einem gleichverteilten zufälligen Generieren von Lösungen ist die Wahrscheinlichkeit einer guten Lösung in einem Schritt

$$P(good) = \frac{|S_{\text{good}}|}{|S|}. \tag{4.3}$$

Die Wahrscheinlichkeit, keine gute Lösung zu finden, ist entsprechend

$$P(bad) = 1 - P(good) = 1 - \frac{|S_{\text{good}}|}{|S|}. \tag{4.4}$$

Bei n generierten Lösungen ist die Wahrscheinlichkeit, keine gute Lösung zu finden, demzufolge

$$P(n \times bad) = \left(1 - \frac{|S_{\text{good}}|}{|S|}\right)^n. \tag{4.5}$$

Soll es nicht auftreten, dass nie eine Lösung gefunden wird, so entspricht dies:

$$P(best \in S_{\text{good}}) = 1 - \left(1 - \frac{|S_{\text{good}}|}{|S|}\right)^n. \tag{4.6}$$

Damit ist der Fall erfasst, dass mindestens ein gültiges Ergebnis existiert.

Anteil der	*Anzahl der Iterationen*			
guten Lösungen	100	1000	10 000	10^5
1%	0,634	0,999 957	fast 1	fast 1
0,1%	0,095	0,632	0,999 95	fast 1
0,01%	0,001	0,095	0,632	0,999 54

Tabelle 4.1: Wahrscheinlichkeit einer guten Lösung bei gleichverteiltem Random Sampling

Da die Gleichung 4.6 nicht sehr anschaulich ist, sind einige Beispiele in Tabelle 4.1 dargestellt. Man sieht hier, dass für jede noch so kleine Anzahl von möglichen guten Lösungen die Wahrscheinlichkeit sehr hoch werden kann, tatsächlich eine dieser Lösungen zu finden. Nötig dafür ist lediglich eine ausreichende Anzahl an Iterationen. Außerdem wurde in diesen Berechnungen von einer gleichverteilten Erzeugung ausgegangen. Durch Anwendung von Vorwissen können aber auch *andere Verteilungen* gewählt werden. Werden Lösungen bevorzugt, die ein Merkmal einer guten Lösung besitzen, lässt sich die Wahrscheinlichkeit erhöhen. Ein solches Merkmal muss nicht einmal eindeutig sein. Der Lösungsraum wird weiterhin vollständig durchsucht, da potentiell noch jede Lösung möglich ist. Es kommt damit zu keiner Einschränkung.

Parallelisierbarkeit

Random Sampling ist ideal für eine parallele Verarbeitung geeignet. Auf den einzelnen Prozessoren werden unabhängig voneinander zufällige Lösungen berechnet. Dazu ist keinerlei Kommunikation nötig. Lediglich am Ende des Vorgangs müssen die besten Ergebnisse gesammelt und ausgewertet werden.

Speicherbedarf

Für das Random Sampling ist kaum Speicher nötig. Es wird immer nur eine Lösung berechnet und die beste gespeichert.

Anytime

Es werden solange Lösungen generiert, bis eine gute gefunden wurde. Es kann aber auch solange gesucht werden, bis eine vorgegebene Zeit abgelaufen ist. Anschließend wird das beste Ergebnis zurückgegeben. War die Zeit ausreichend, um genügend viele Lösungen zu testen, wird mit einer gewissen Wahrscheinlichkeit eine gute dabei sein.

Anwendbarkeit

Durch die Einfachheit von Random Sampling lässt sich das Verfahren sehr gut auf jedes Problem anwenden. Die Qualität der Lösung hängt zwar stark von dem Anteil guter Lösungen im Lösungsraum ab, aber auch die konkrete Implementierung ist entscheidend. Eine gleichverteilte Erzeugung ist ein einfacher Ansatz. Meist führen aber erst besser gewählte Verteilungen zu befriedigenden Resultaten. Zumindest gilt immer: Wird lange genug gesucht, so wird auch mit einer entsprechend hohen Wahrscheinlichkeit eine gute Lösung ermittelt.

4.4 Lokale Suche

Das vorgestellte Random-Sampling-Verfahren hat einen Nachteil: Es findet sehr selten die optimale Lösung. Es werden stets nur einzelne Punkte im Lösungsraum betrachtet. Ist die Anzahl der optimalen Lösungen zur Gesamtanzahl der möglichen Lösungen sehr gering, dann ist auch das Finden einer optimalen Lösung höchst unwahrscheinlich.

Die *Lokale Suche* beseitigt diesen Nachteil. Es wird nicht nur ein einzelner Punkt, sondern auch dessen Nachbarschaft betrachtet. Dazu wird ausgehend von einem zufällig gewählten Startpunkt iterativ zu einem jeweils besseren Nachbarn gewechselt. Wird kein besserer Nachbarpunkt gefunden, terminiert der Algorithmus. Die gefundene Lösung stellt ein lokales Maximum dar. Durch mehrfaches Ausführen mit unterschiedlichen Startwerten kann aber auch ein globales Optimum bestimmt werden.

Algorithmus 5 Lokal Search

```
a := (random) initial solution
loop
  select b ∈ N(a) s.th. φ(b) ≥ φ(n) : ∀n ∈ N(a)
  if φ(b) > φ(a) then
    a := b
  else
    STOP and return a
  end if
end loop
```

Für das Verfahren benötigt man die Nachbarschaftsfunktion

$$N : S \mapsto 2^S. \tag{4.7}$$

In der Regel ist eine benachbarte Lösung lediglich in einem kleinen Detail verändert. Damit ist ein Finden der Nachbarschaftsfunktion nicht allzu schwierig. Komplexe Nachbarschaften sind natürlich auch möglich. Eine allgemeine Regel ist nur, dass die Bewertung von benachbarten Lösungen ähnlich der des Ausgangspunktes sein sollte.

Die Lokale Suche terminiert meist sehr schnell. Je nach Startlösung wird ein anderes lokales Optimum gefunden. Um die Wahrscheinlichkeit einer guten Lösung zu erhöhen, wird der Algorithmus in der Regel mehrfach gestartet.

Parallelisierbarkeit

Ein einzelner Lauf der Lokalen Suche ist schlecht zu parallelisieren, da er sehr sequentiell abläuft. In der Regel wird der Algorithmus aber wiederholt gestartet, um gute Ergebnisse zu erzielen. Dies kann auch parallel auf mehreren Prozessoren erfolgen. Damit ist eine Parallelisierung möglich.

Speicherbedarf

Bei der lokalen Suche werden keine größeren Datenmengen benötigt. Es werden immer nur die aktuelle Lösung und ein Nachbar zu einem Zeitpunkt betrachtet. Somit ist der Speicherbedarf ebenfalls sehr gering.

Anytime

In dem Verfahren liegt zu jedem Zeitpunkt eine zulässige Lösung vor. Wird der Algorithmus abgebrochen, kann diese übernommen werden. Werden mehrere Läufe betrachtet, gibt es auch immer eine beste Lösung, verwendet werden kann.

Anwendbarkeit

Die Lokale Suche lässt sich gut auf unterschiedliche Probleme anwenden. Man benötigt lediglich eine Definition der Nachbarschaft. Oft erhält man Nachbarn einer Lösung, indem ein einzelnes Element geringfügig verändert wird. Dies sollte sich bei den meisten Problemen leicht anwenden lassen.

4.5 Tabu-Suche

Die Lokale Suche wechselt ausgehend von einer Lösung immer zu einer besseren benachbarten Lösung. Der Algorithmus terminiert, sobald kein besserer Nachbar gefunden wird, also ein lokales Optimum vorliegt. Die Hoffnung ist, dass dieses lokale Optimum auch ein globales ist, oder zumindest eine gute Lösung darstellt. Treten in dem Lösungsraum sehr

viele lokale Maxima auf, so terminiert der Algorithmus bereits in diesen. Der Lösungsraum wird damit nur unzureichend durchsucht, was die Erfolgschancen verringert.

Dieses Verhalten kann man durch die *Tabu-Suche* [43] umgehen. Dazu wird die Bedingung aufgehoben, dass der folgende Nachbar besser bewertet sein muss als die aktuelle Lösung. Hierdurch wird es möglich, aus einem lokalen Maximum zu „entkommen".

Würde man nur dieses Vorgehen allein wählen, bestünde das Risiko eines Zyklus: Angenommen, der Algorithmus wechselt von einer Lösung s zu seinem besten Nachbarn s'. Dieser stellt ein lokales Maximum dar. Durch die Aufhebung der Terminierungsbedingung wird nun von s' zu einem Nachbarn mit einer schlechteren Bewertung gewechselt. Dies könnte auch wieder s sein. Da der Algorithmus deterministisch abläuft, würde im nächsten Schritt wieder s' gewählt werden, dann wieder s und so weiter.

Zum Verhindern solcher Zyklen werden die bereits besuchten Lösungen in einer Tabu-Liste gespeichert. Es wird dann nicht mehr zwangsläufig zu dem Nachbarn mit der besten Bewertung gewechselt, sondern die Folgelösung darf zusätzlich auch nicht in der Tabu-Liste enthalten sein. Der Algorithmus terminiert dann, wenn alle Nachbarn der aktuellen Lösung in der Tabu-Liste enthalten sind oder wenn diese voll ist.

Da bei diesem Ansatz nicht nur lokale Maxima, sondern auch globale verlassen werden können, wird während der Suche immer die beste gefundene Lösung abgespeichert. Diese wird am Ende als Ergebnis geliefert.

Algorithmus 6 Tabu Search

a := (random) initial solution
$best := a$
loop
  $Tabu := Tabu \cup \{a\}$
  select $b \in N(a) \setminus Tabu$ s.th. $\phi(b) \geq \phi(n) : \forall n \in N(a)$
  if $N(a) \setminus Tabu \neq \emptyset \wedge |Tabu| \leq k$ **then**
    $a := b$
    if $\phi(best) < \phi(a)$ **then**
      $best := a$
    end if
  else
    STOP and return $best$
  end if
end loop

Entscheidend für die Güte der gefundenen Lösung ist die Kapazität der Tabu-Liste. Ist diese zu klein, terminiert der Algorithmus zu schnell. Eventuell wurde noch nicht einmal ein lokales Maximum erreicht. Die Wahrscheinlichkeit einer guten Lösung lässt sich zudem noch verbessern, indem mehrfach mit unterschiedlichen Startlösungen gerechnet wird.

Parallelisierbarkeit

Für die Parallelisierbarkeit der Tabu-Suche gilt das gleiche wie für die Lokale Suche. Durch den sequentiellen Ablauf ist ein einzelner Lauf nicht ohne weiteres zu parallelisieren. Es

ist aber auch hier möglich, auf mehreren Prozessoren Berechnungen mit unterschiedlichen Startwerten laufen zu lassen. Dies erhöht die Wahrscheinlichkeit einer guten Lösung.

Speicherbedarf

Das Verfahren speichert in jedem Schritt die besuchten Lösungen in der Tabu-Liste. Diese sollte recht groß sein, damit das Verfahren nicht zu früh terminiert. Da die Tabu-Liste entsprechend viel Speicherplatz belegt, ergibt sich für die Tabu-Suche auch ein verhältnismäßig hoher Speicherbedarf.

Anytime

Während eines Laufs der Tabu-Suche enthält die Variable $best$ immer die beste bisher gefundene Lösung. Wird der Algorithmus vorzeitig abgebrochen, kann diese Lösung als Ergebnis verwendet werden.

Anwendbarkeit

Für die Anwendbarkeit der Tabu-Suche gelten die gleichen Bedingungen wie für die Lokale Suche. Man benötigt die Definition der Nachbarschaft, welche sich meistens leicht finden lässt.

4.6 Simulated Annealing

Mit der Tabu-Suche wurde die Lokale Suche dahingehend verbessert, dass ein „Entkommen" aus lokalen Optima möglich ist. Um dabei Zyklen zu verhindern, wurden die bereits besuchten Lösungen in einer Liste gespeichert. Beim *Simulated Annealing* wird das gleiche Ergebnis erreicht, allerdings ohne dass diese Liste geführt wird.

Das Simulated Annealing (simuliertes Ausglühen) orientiert sich dabei an einem physikalischen Prozess - dem langsamen Abkühlen von festen Stoffen. Bei diesem Vorgang nehmen die Atome einen Zustand minimaler Energie an. Dies kann auf folgende Art und Weise als Optimierungsproblem betrachtet werden:

Zu Beginn befinden sich etliche Fehler in der Kristallstruktur. Ziel ist es, diese Fehler zu entfernen und die Atome in eine geordnete Struktur zu bringen, die dem minimalen energetischen Zustand entspricht. Dazu wird das Material bis auf die Schmelztemperatur erhitzt. Dadurch werden die Atome frei beweglich und können zufällige Positionen einnehmen. Anschließend wird die Temperatur langsam gesenkt, einem vorgegebenen Verlauf entsprechend. Dadurch ordnen sich die Atome zu einem energetisch günstigen Zustand.

Dieses Prinzip wird beim Simulated Annealing auf Optimierungsprobleme übertragen. Das Verfahren kann als eine Art Lokale Suche mit Zufallskomponente angesehen werden. Dazu

wird in jedem Schritt zufällig ein Nachbar der aktuellen Lösung bestimmt. Hat dieser eine bessere Bewertung, wird er als neue Lösung übernommen. Ist die Bewertung schlechter, wird er mit einer gewissen Wahrscheinlichkeit genommen. Entscheidend ist, wie hoch die Wahrscheinlichkeit für das Übernehmen ist. Beim Ausglühen entspricht die Temperatur eines Stoffes der Teilchenbeweglichkeit. Hier bestimmt sie die „Beweglichkeit" einer Lösung, indem die Wahrscheinlichkeit der Übernahme eine schlechteren Lösung daraus abgeleitet wird.

Algorithmus 7 Simulated Annealing

```
t := 0
a := random initial solution
T := initial temperature
while T > 0 do
  randomly select b ∈ N(a)
  if φ(b) ≥ φ(a) then
    a := b
  else
    generate uniformly a random number r ∈ (0, 1)
    if r < e^(−(φ(b)−φ(a))/T) then
      a := b
      t := t + 1
      T := f(T, t)
    end if
  end if
end while
```

Wie die Temperatur über der Zeit abgesenkt wird, bestimmt sich über die Funktion f. Diese muss geeignet gewählt werden. Einige Beispiele für geeignete Funktionen finden sich in [28, 1].

Parallelisierbarkeit

Das Simulated Annealing verläuft äußerst sequentiell. Daher kann ein einzelner Lauf nicht besonders gut auf mehrere Prozessoren verteilt werden. In der Regel wird man aber den Algorithmus mehrmals starten, um die Wahrscheinlichkeit einer guten Lösung zu erhöhen. Dies kann auch parallel auf mehreren Prozessoren geschehen.

Speicherbedarf

Im Gegensatz zur Tabu-Suche wird beim Simulated Annealing kein zusätzlicher Speicher benötigt. Da keine großen Datenmengen verwendet werden, ist auch der gesamte Speicherbedarf sehr gering.

Anytime

Beim normalen Simulated Annealing liegt die endgültige Lösung erst am Ende der Berechnung fest. Zwischenzeitlich enthält die Variable a zwar eine Lösung, diese kann aber sehr schlecht sein.

Das Verfahren lässt sich allerdings leicht modifizieren: Zusätzlich zur aktuellen Lösung a wird immer die beste im Verfahren betrachtete Lösung gespeichert. Diese kann dann anstatt des Endergebnisses zurückgeliefert werden. Diese Modifikation führt auf keinen Fall zu schlechteren Ergebnissen, kann aber in einigen Fällen zu besseren führen. Zusätzlich kann dieser Wert Verwendung finden, wenn der Algorithmus vorzeitig abgebrochen wird.

Anwendbarkeit

Das Simulated Annealing stellt die gleichen Anforderungen wie Lokale Suche und Tabu-Suche. Es wird lediglich die Definition einer Nachbarschaft benötigt. Für den Algorithmus ist zusätzlich noch eine Temperaturfunktion erforderlich. Diese kann aus der reichhaltigen Auswahl an bekannten Funktionen [1] gewählt werden.

4.7 Genetische Algorithmen

Genetische Algorithmen basieren auf der Idee von Evolutionsstrategien [40]. Das Verfahren wurde von John Holland entwickelt und ausführlich in [19] beschrieben. Grundlage ist die Darwinsche Evolutionstheorie. Die Mechanismen der natürlichen Evolution werden dabei auf ein mathematisches Problem angewendet. Nach dem Prinzip des Überlebens der stärksten Individuen werden die besten Lösungen ermittelt. Durch Anwendung der Bewertungsfunktion erhält man damit potentiell gute Lösungen.

Das Verfahren arbeitet auf einer Menge von Lösungen - der Population. Die einzelnen Individuen der Population werden einem künstlichen Evolutionsprozess unterzogen. Dazu dienen die Selektion, der Crossoverprozess und die Mutation.

Kodierung

Um ein Problem mit Genetischen Algorithmen zu behandeln, müssen die Lösungen über eine geeignete Codierung dargestellt werden. Natürliche Chromosomen sind Sequenzen aus vier organischen Basen. In Anlehnung daran werden Lösungen meistens als Sequenzen über einem Alphabet dargestellt. Im einfachsten Fall sind es Bitfolgen. Aber auch komplexere Darstellungen wie Bäume werden verwendet. Die Idee bei Genetischen Algorithmen ist, dass Teile der Codierung gewissen Merkmalen entsprechen. Durch Neukombination oder lokale Manipulation werden die Merkmale ausgetauscht beziehungsweise verändert.

Selektion

Die Selektion dient der Auswahl der besten Individuen. Möglichst nur diese sollen ihre Merkmale an die nächste Generation vererben. Um dies zu erreichen, werden Individuen zufällig, aber je nach Bewertung mit einer höheren Wahrscheinlichkeit ausgewählt. Sie dienen als Eltern zum Erzeugen von Elementen einer neuen Population.

Crossover

Crossover dient der Kombination von Merkmalen der Elternindividuen. Ähnlich dem biologischen Vorbild werden dabei Chromosomenstücke der Eltern ausgetauscht. Dazu werden die Chromosomen an einer oder mehreren zufälligen Positionen geteilt. Die entstandenen Teilstücke werden ausgetauscht und neu zusammengesetzt. Somit entstehen neue Lösungen aus Teilen der Elternchromosomen. Entsprechen die ausgetauschten Stücke gewissen Merkmalen, werden diese neu kombiniert.

Mutation

Die Mutation dient dem Erzeugen neuer Merkmale. Allein durch Crossover kann nicht der gesamte Lösungsraum durchsucht werden. Es kommen dabei keine neuen Teilstücke in die Population. Die Mutation verändert punktuell ein Chromosom und ermöglicht somit das Entstehen neuer Merkmale.

Der herkömmliche Algorithmus

Der herkömmliche Algorithmus verwendet die beschriebenen Methoden nun zur Bestimmung einer möglichst optimalen Lösung. Dazu wird anfangs eine zufällige Elternpopulation erzeugt. Ausgehend von dieser wird eine neue Generation gebildet. Individuen werden durch Selektion ausgewählt, mit einer gewissen Wahrscheinlichkeit ein Crossover angewendet und die Kindindividuen werden möglicherweise einer Mutation unterzogen. Ist die neue Population aufgefüllt, so dient sie im nächsten Schritt als Elternpopulation.

Als Terminierungsbedingung kann die Anzahl der Schritte, aber auch die Güte der besten Lösung verwendet werden. Anschließend wird die beste Lösung der neuen Generation als Ergebnis zurückgeliefert.

Es gibt einige Erweiterungen an diesem Verfahren, die für spezielle Probleme zu besseren Ergebnissen führen. Bei besonderen Kodierungen können auch Crossover oder Mutation durch andere Operationen ersetzt werden. Häufig wird auch ein als Elitismus bezeichnetes Prinzip angewendet. Nach dem Aufbau der neuen Generation werden normalerweise alle alten Lösungen verworfen. Dies kann dazu führen, dass sehr gute Lösungen verloren gehen. Beim Elitismus wird das verhindert, indem eine bestimmte Anzahl der besten Lösungen automatisch in die nächste Generation übernommen wird.

Algorithmus 8 Basic Genetic Algorithm

```
Generate random population P of n chromosomes
for all x ∈ P do
  Evaluate the fitness f(x)
end for
repeat
  N := ∅ { start with an empty new generation}
  repeat
    Select two parents x_1, x_2 ∈ P randomly according to f(x_i) {selection}
    choose random number r ∈ [0, 1]
    if r < crossover probability then {crossover}
      (x̃_1, x̃_1):= crossover(x_1, x_2)
    else
      (x̃_1, x̃_1):= (x_1, x_2)
    end if
    for all position x̃_1^i on x̃_1 do
      choose random number r ∈ [0, 1]
      if r < mutation probability then {mutation}
        mutate x̃_1^i
      end if
    end for
    for all position x̃_2^i on x̃_2 do
      choose random number r ∈ [0, 1]
      if r < mutation probability then {mutation}
        mutate x̃_2^i
      end if
    end for
    N := N ∪ {x̃_1, x̃_2}
  until |N| = population size
  P := N {Use new population for next turn }
until good enough
return best solution of N
```

Parallelisierbarkeit

Der grundlegende Genetische Algorithmus lässt sich gut parallelisieren. Die Individuen einer neuen Generation werden unabhängig voneinander erzeugt. Dies kann also auch parallel auf mehreren Maschinen passieren. Die neue Population muss im Anschluss zusammengefasst und an die einzelnen Prozessoren verteilt werden. Eine andere Möglichkeit ist das Parallelisieren durch unabhängiges Starten auf den einzelnen Prozessoren. Eine Kommunikation ist damit erst am Ende nötig. Diese beiden Ansätze lassen sich aber auch kombinieren. Dazu werden die Populationen getrennt voneinander berechnet. Zu gewissen Zeitpunkten werden sie zusammengefasst und einem Evolutionsschritt unterzogen. Anschließend erfolgt eine zufällige Trennung und unabhängige Berechnung. Dieses Verfahren hat den Vorteil, dass der Kommunikationsaufwand sich vorgeben lässt, je nach Häufigkeit der Vereinigung.

Speicherbedarf

Der Speicherbedarf bei Genetischen Algorithmen bestimmt sich hauptsächlich nach der Größe der Populationen. Diese umfassen in der Regel 100 bis 10 000 Individuen. Es wird daher der 100- bis 10 000-fache Speicherbedarf eines einzelnen Individuums benötigt. Je nach Kodierung ist ein Individuum einige Byte bis Kilobyte groß. Dieser Speicherbedarf ist also nicht unerheblich.

Anytime

Für Genetische Algorithmen ist kein festes Abbruchkriterium festgelegt. Prinzipiell kann jederzeit gestoppt und die beste Lösung der aktuellen Population übernommen werden.

Anwendbarkeit

Ein Genetischer Algorithmus kann immer dann zum Lösen eines Optimierungsproblems verwendet werden, wenn sich eine geeignete Kodierung für die Lösungen finden lässt. Ist das Problem mathematisch formuliert, liegt immer eine formale Beschreibung einer Lösung vor. Diese Lösung sollte sich zumindest immer als Bitfolge darstellen lassen. Auf diese Bitfolge kann dann das herkömmliche Verfahren angewendet werden. Die Qualität der erreichten Lösungen wird damit aber nicht garantiert.

4.8 Zusammenfassung und Bewertung

Im vorhergehenden Teil der Arbeit wurden prominente Heuristiken zum Lösen von Optimierungsproblemen betrachtet. Die Verfahren wurden auf ihre Eignung bezüglich der für die Laufplanung entscheidenden Kriterien untersucht. Dies ist noch einmal in der folgenden Tabelle zusammengefasst.

Heuristik	*Anytime*	*Parallelisierbarkeit*	*Speicherbedarf*	*allgemeine Anwendbarkeit*
Greedy	nein	nein	gering	schlecht
Branch and Bound	ja	ja	gering	schlecht
Lokale Suche	ja	multistart	gering	gut
Tabu-Suche	ja	multistart	hoch	gut
Random Sampling	ja	ja	gering	sehr gut
Simulated Annealing	ja	multistart	gering	gut
Genetische Alg.	ja	ja	hoch	gut

Alle vier geforderten Eigenschaften werden nur von Random Sampling, Lokaler Suche und Simulated Annealing erfüllt. Die anderen Verfahren ließen sich nur unter Einschränkungen verwenden. Aus diesem Grund werden hier lediglich die drei genannten Verfahren näher untersucht.

5 Random Sampling

Im vorhergehenden Kapitel hat sich Random Sampling als theoretisch geeignetes Verfahren herausgestellt. In diesem Kapitel soll ein Algorithmus nach diesem Ansatz entwickelt und bewertet werden. Dazu ist eine Funktion zum Erzeugen zufälliger gültiger Lösungen nötig. Sie wird im ersten Teil beschrieben. Es folgt eine kurze Beschreibung der Implementierung und eine erste Bewertung des entwickelten Algorithmus.

5.1 Erzeugen zufälliger gültiger Lösungen

Für Algorithmen entsprechend der Verfahren Random Sampling, Lokale Suche und Simulated Annealing wird eine Funktion zum Erzeugen zufälliger zulässiger Lösungen benötigt. Um nicht bereits im Voraus gute Lösungen auszuschließen, ist es erforderlich, dass *alle* möglichen Bewegungen las potentielle Lösungen erzeugt werden können. Ausgeschlossen sind dabei nur solche, die:

- ein Kippen des Roboters verursachen,
- zum Betreten von instabilem Untergrund führen,
- Fußpositionen außerhalb der physikalischen Reichweite der Beine annehmen,
- zum Zusammenstoßen von Füßen führen,
- eine Geschwindigkeit für die Fußbewegung annehmen, die der Regler nicht leisten kann.

Am Ende soll eine zufällige zulässige Bewegung geliefert werden, wie sie in Abschnitt 3.3 beschrieben ist.

Das Erzeugen einer Lösung erfolgt in mehreren Schritten:

1. Es wird ein Pfad zum Ziel festgelegt, auf dem der Roboter bewegt wird. Für den Mittelpunkt sind damit nur noch Positionen auf diesem Pfad zulässig.
2. Für die aktuelle Fußstellung wird der Bereich ermittelt, auf dem sich der Mittelpunkt befinden kann ohne die oben genannten Einschränkungen zu verletzen.
3. Es wird zufällig ausgewählt, welcher Fuß angehoben oder abgesetzt wird.

4. Im Fall eines Absetzens des Fußes wird zufällig eine neue gültige Position ermittelt.
5. Die Position des Mittelpunktes wird genau festgelegt. Die geschieht zufällig im Bereich, der in Punkt 2 ermittelt wurde.
6. Zu den Positionen werden minimale zulässige Werte für Dauer und Startzeit berechnet.

Die genaue Ausführung der Schritte ist im Folgenden beschrieben.

5.1.1 Festlegen eines Pfades für den Mittelpunkt

Bereits frühzeitig wird ein Pfad festgelegt, dem der Robotermittelpunkt im Laufe der Bewegung folgt. Darauf aufbauend erfolgt dann die Festlegung der Fußbewegung und der entsprechenden Bewegungszeiten. Häufig sollte es möglich sein, das Ziel auf direktem Wege über eine Gerade zu erreichen. Bei Hindernissen ist dies eventuell ausgeschlossen. Der Pfad muss dann entsprechend um das Hindernis herumgeführt werden. Er erhält dadurch einen „Knick".

Die Darstellung des Pfades erfolgt über einen oder mehrere Streckenabschnitte. Der Startpunkt des ersten Abschnittes ist der Startpunkt der Bewegung, der Endpunkt des letzten Abschnittes ist dementsprechend der Endpunkt der Bewegung. Die Anzahl der nötigen Streckenabschnitte hängt vom „Umweg" ab, der zum Umgehen von Hindernissen erforderlich ist. Je weniger Abschnitte benötigt werden, umso direkter erfolgt der Weg zum Ziel. Dementsprechend wurde auch die Verteilung zum Erzeugen der Anzahl gewählt. Geringe Streckenanzahlen sollen häufig sein. Je höher die Anzahl, desto geringer soll die Wahrscheinlichkeit sein.

Diese Eigenschaften hat die hier verwendete *Geometrische Verteilung*. Dies ist eine diskrete Verteilung mit einer Einzelwahrscheinlichkeit

$$P(x) = \begin{cases} p(1-p)^{x-1} & : \quad x \geq 1 \\ 0 & : \quad sonst \end{cases} . \tag{5.1}$$

Der Erwartungswert liegt bei

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

und die Varianz beträgt

$$Var(X) = \frac{1-p}{p^2}. \tag{5.3}$$

Die Anzahl der Abschnitte lässt sich über den Parameter p variieren. Mit kleiner werdenden p steigt der Erwartungswert, das heißt es steigt die Wahrscheinlichkeit für eine höhere Abschnittsanzahl.

Die Berechnung der Zufallsgröße erfolgt durch Ableitung aus einer gleichverteilten Zufallsgröße, wie in Algorithmus 9 beschrieben.

Nachdem die Anzahl der Abschnitte des Pfades festgelegt ist, werden noch Punkte benötigt, die als Start beziehungsweise Ende der Abschnitte dienen. Sie werden einfach zufällig gleichverteilt um Start- und Zielpunkt gewählt.

Algorithmus 9 Berechnen einer geometrisch verteilten Zufallszahl

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

5.1.2 Ermitteln des zulässigen Bereichs für den Mittelpunkt

Bei einer gegebenen Fußstellung ist nur ein gewisser Bereich für den Mittelpunkt zulässig, ohne dass die auf Seite 51 genannten unzulässigen Eigenschaften auftreten. Dieser Bereich wird beschränkt durch die Reichweite der Fußpunkte vom Mittelpunkt und den minimalen Stability Margin. Der Mittelpunkt darf also nur im Durchschnitt des stabilitätserhaltenden Bereiches (konvexe Hülle der Fußpunkte minus minimaler Stability Margin) und der Erreichbarkeitsbereiche der Füße liegen. Im weiter wird hierfür der Begriff *zulässiger Bereich* gebraucht.

An dieser Stelle sei angemerkt, dass sowohl der stabilitätserhaltende Bereich als auch die Erreichbarkeitsbereiche konvex sind. Daraus ergibt sich, dass auch der Durchschnitt (der zulässige Bereich) die Konvexitätseigenschaft erfüllt. Diese Eigenschaft wird für die weiteren Betrachtungen benötigt.

Da sich der Mittelpunkt nur auf dem vorher festgelegten Pfad bewegen kann, ist der zulässige Bereich weiter eingeschränkt. Aufgrund der Konvexität und der Tatsache, dass der Pfad aus geraden Streckenstücken zusammengesetzt ist, existieren genau zwei Punkte, in denen der Mittelpunkt den zulässigen Bereich verlassen kann. Diese *begrenzenden Pfadpunkte* dienen im weiteren Verlauf der Beschreibung der Mittelpunktpositionen. Sie spannen ein Intervall auf dem Pfad auf, das alle für den Mittelpunkt erlaubten Positionen enthält. In Abbildung 5.1 ist dies vereinfacht dargestellt. Der zulässige Bereich ist in diesem Beispiel identisch mit der konvexen Hülle der Fußpunkte (minimaler Stability Margin ist 0).

Berechnung der begrenzenden Pfadpunkte

Die begrenzenden Pfadpunkte sind definiert als Schnittpunkte des Pfades mit dem Rand des zulässigen Bereichs. Sie beschreiben auf dem Pfad das zulässige Intervall für den Mittelpunkt. Dieses kann aber, wie in Abbildung 5.2, nicht eindeutig sein. Zur Identifikation wird daher noch eine zusätzliche Information benötigt. Dies geschieht hier durch Angabe eines Punktes als Repräsentant für den umgebenden Bereich. Es sind also die begrenzenden Pfadpunkte gesucht, deren Intervall den gegebenen *inneren Punkt* enthält.

Der zulässige Bereich ist äußerst schwierig analytisch zu beschreiben. Die Schnittpunkte lassen sich deshalb auch nicht ohne weiteres analytisch berechnen. Aus diesem Grund wird ein numerisches Verfahren verwendet. Dieses benötigt nur eine Funktion, die testet, ob ein gegebener Punkt im zulässigen Bereich liegt. Über Binäre Suche lässt sich dann der Schnittpunkt ermitteln.

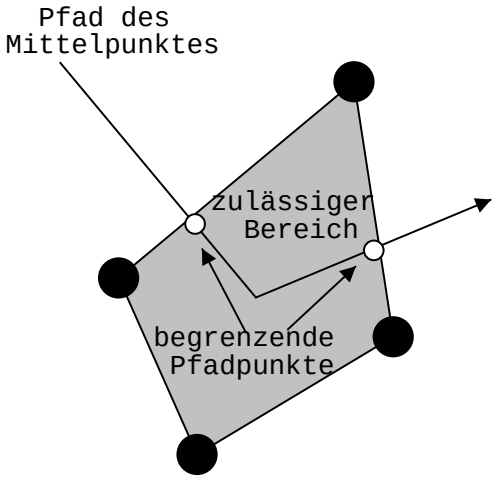


Abbildung 5.1: begrenzende Pfadpunkte

Hierzu werden aber zwei Punkte benötigt, die sich auf einem Streckenabschnitt befinden, wobei einer innerhalb und einer außerhalb des zulässigen Bereichs liegt. Der Streckenabschnitt wird durch Suche, ausgehend vom vorgegebenen inneren Punkt, ermittelt. Gesucht wird ein Streckenabschnitt in jeweils einer Richtung auf dem Pfad. Sobald sich der Endpunkt außerhalb des zulässigen Bereichs befindet, stehen die benötigten Punkte für die Binäre Suche zur Verfügung. Der Endpunkt befindet sich dann außerhalb und kann verwendet werden. Als innerer Punkt kann der vorgegebene dienen, falls er auf dem herausführenden Streckenabschnitt liegt. Sonst wird der Startpunkt des Steckenabschnitts verwendet.

Mit Hilfe der gefundenen Punkte wird auf dem Streckenabschnitt die Binäre Suche angewendet. Dadurch wird jeweils ein Schnittpunkt des Pfades mit dem zulässigen Bereich ermittelt.

5.1.3 Auswahl des zu wechselnden Fußes

Das Vorwärtsbewegen des Roboters erfolgt durch Verschieben des Mittelpunktes. Dies ist allerdings nur innerhalb des zulässigen Bereiches möglich, der sich wie in Abschnitt 5.1.2 beschrieben berechnen lässt. Um nun eine Bewegung zum Ziel zu ermöglichen, muss dieser Bereich verschoben werden, was durch Heben und Umsetzen der Fußpunkte geschieht.

Für das Wechseln der Füße sind drei Kriterien zu beachten, die für die Gültigkeit der Lösung entscheidend sind:

- Die Stabilität muss gewahrt bleiben.
- Ein Fuß darf nur auf zulässige Bereiche gesetzt werden.
- Die Füße dürfen nicht zusammenstoßen.

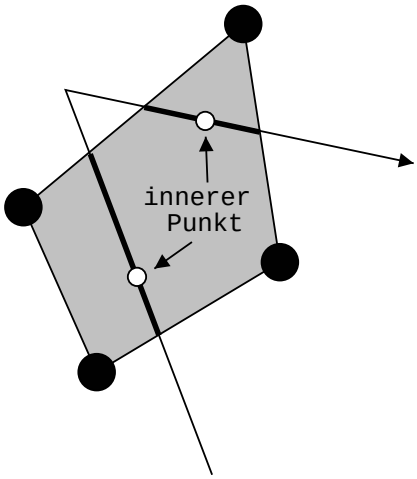


Abbildung 5.2: Festlegen des Pfadbereichs über innere Punkte

Für die Stabilität ist es wichtig, welche Füße angehoben sind, und welche den Körper stützen. Grundsätzlich müssen sich mindestens drei Füße auf dem Boden befinden. Außerdem muss mindestens ein Fuß auf jeder Seite den Körper stützen.

Für einen sechsbeinigen Laufroboter ergeben sich damit 40 Möglichkeiten. Diese hier *Stützzustände* genannten Möglichkeiten sind in Tabelle 5.1 dargestellt. Nur wenn ein solcher Stützzustand vorliegt, kann überhaupt ein stabiler Stand für den Roboter gefunden werden.

Der Zustandsgraph

Für das Heben und Setzen der Füße gilt, dass immer in einen dieser Zustände gewechselt werden muss. Nur dann ist es möglich, den stabilen Stand beizubehalten, und der Vorgang ist somit zulässig.

In dieser Arbeit wird die Zulässigkeit mit Hilfe eines gerichteten Graphen

$$G = (C, T) \tag{5.4}$$

modelliert, wobei C die Kontenmenge und T die Kantenmenge darstellt. Die Knoten entsprechen den zulässigen Stützzuständen. Ein einzelner Zustand wird über eine Teilmenge der Fußnummern

$$F \subseteq \mathbb{N} \cap [1, 6] = \{1, 2, 3, 4, 5, 6\} \tag{5.5}$$

dargestellt. Die Knotenmenge C ist dementsprechend:

$$C = \{c \in 2^{\mathbb{N} \cap [1,6]}, c \text{ ist in Tabelle 5.1 enthalten}\}. \tag{5.6}$$

Der Graph enthält immer dann eine Kante zwischen zwei Knoten, wenn sich die Zustände um genau einen Fuß unterscheiden.

$$T = \{(c_1, c_2) \in C \times C, |(c_1 \setminus c_2) \cup (c_2 \setminus c_1)| = 1\} \tag{5.7}$$

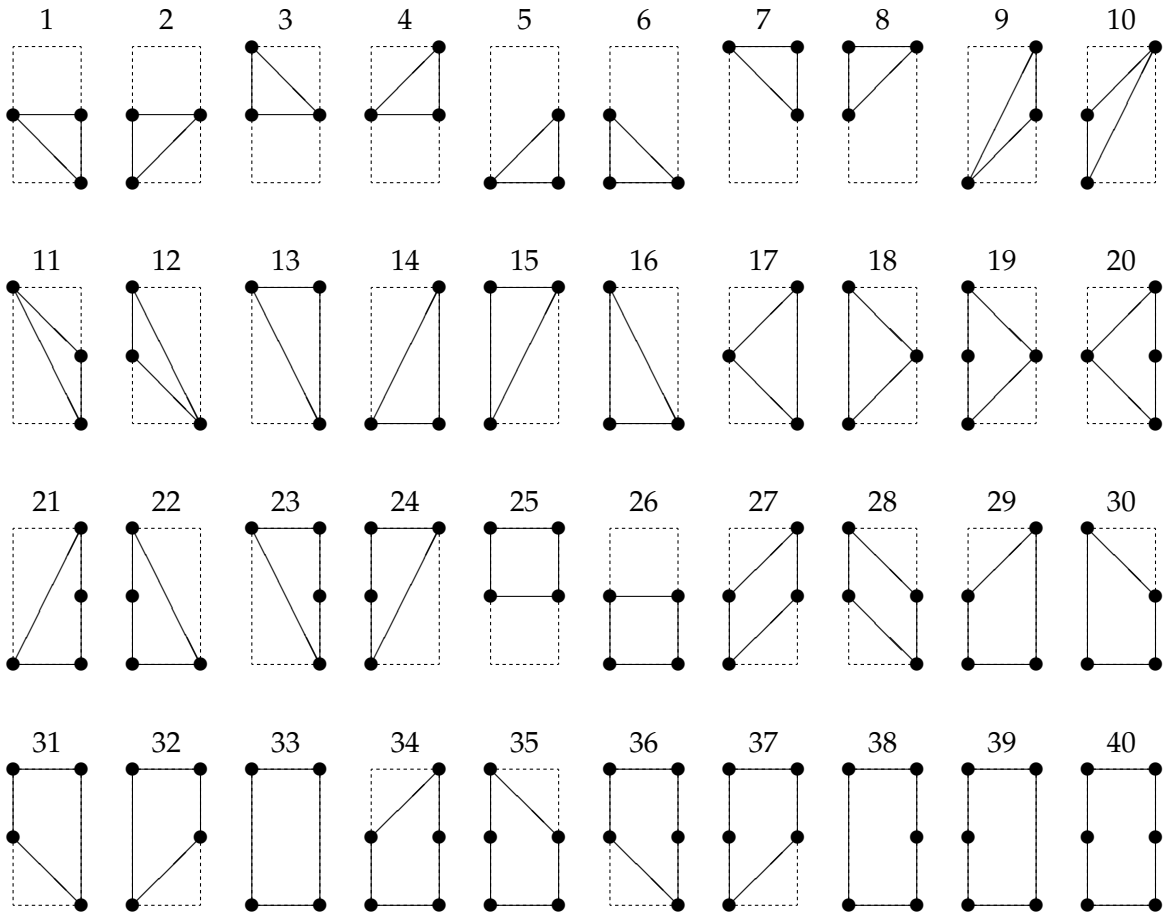


Tabelle 5.1: 40 zulässige Stützzustände

Dieser Graph enthält also alle Nachfolger eines Stützzustandes, die sich durch Anheben oder Absetzen genau eines Fußes ergeben. Sie sind in Tabelle 5.2 aufgelistet.

Für eine Laufbewegung kann es erforderlich sein, zwei Füße *gleichzeitig* zu heben oder aufzusetzen. Scheinbar wird dies nicht in dem Graphen erfasst. Dennoch ist es möglich, da durch die Übergänge nicht die Dauer beschrieben ist, für die ein Zustand angenommen wird. Sollen beispielsweise die Füße 3 und 4 gleichzeitig gehoben werden, ist dies durch die Übergänge $\{1, 2, 3, 4, 5, 6\} \longrightarrow \{1, 2, 4, 5, 6\} \longrightarrow \{1, 2, 5, 6\}$ erfasst. Der Zwischenzustand $\{1, 2, 4, 5, 6\}$ wird dabei für eine Dauer von $t = 0$ angenommen, wodurch sofort in den Nachfolgezustand gewechselt wird. Die Füße 3 und 4 werden gleichzeitig angehoben. Auf diese Weise sind *alle* möglichen Übergänge durch den Graphen abgedeckt.

Absetzen eines Fußes

Das Absetzen eines Fußes sollte immer möglich sein, sofern sich ein sicherer Punkt auf dem Untergrund findet. Die Stabilität bleibt erhalten, da die konvexe Hülle der Fußpunkte

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

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

Tabelle 5.2: Transitionstabelle der Stützzustände

sich nur vergrößern kann. Lediglich eine Verkleinerung des zulässigen Bereichs ist möglich, wenn dieser nun durch die Reichweite des Beins zusätzlich eingeschränkt wird. Anschließend muss also eine Neuberechnung des zulässigen Bereichs erfolgen, wie in Abschnitt 5.1.2 beschrieben.

Die Fußposition für das Absetzen wird zufällig ermittelt. Dazu wird zufällig ein Punkt in der Reichweite des Fußes gewählt. Die Position wird über eine Dreiecksverteilung bestimmt. Die Parameter werden so gewählt, dass der leicht in Richtung Ziel verschobene Mittelpunkt als Erwartungswert angenommen wird.

Ist der zufällig gewählte Punkt zulässig, wird er als neue Fußposition übernommen. Falls nicht, wird nach einem zulässigen Punkt in der Umgebung des gewählten gesucht. Die Suche geschieht dabei spiralförmig, wie in Abbildung 5.3 dargestellt.

Dadurch wird ein Punkt gesucht der nicht, allzu weit vom vorher bestimmten entfernt ist. Der Abstand der einzelnen Punkte der Spirale muss kleiner gewählt werden, als die Auflösung der Terrain Map, damit alle Felder mindestens einmal getestet werden. Der Umfang der Spirale muss so festgelegt werden, dass der gesamte erreichbare Bereich des Fußes durchsucht wird.

Wird kein zulässiger Punkt gefunden, kann keine Lösung generiert werden. Da kein Backtracking oder ähnliche Mechanismen vorgesehen sind, bleibt nur die Möglichkeit des Abbruchs. Die Lösung muss verworfen und eine neue generiert werden. Dieser Fall tritt sehr selten auf. Es ist daher sehr wahrscheinlich, dass beim nächsten Versuch eine zulässige Lösung erzeugt wird.

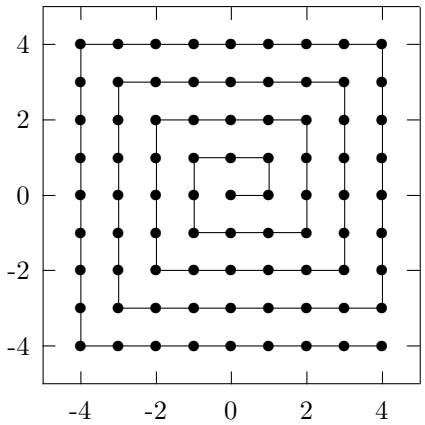


Abbildung 5.3: Suchspirale

Anheben eines Fußes

Beim Anheben eines Fußes besteht die Gefahr, dass die Stabilität nicht mehr gewährleistet ist. Dies ist der Fall, wenn sich nach dem Anheben kein Punkt des Mittelpunktpfades mehr innerhalb des zulässigen Bereichs befindet (vgl. Abbildung 5.4). Es ist also notwendig, diesen Fall zu testen und das Anheben gegebenenfalls zu unterbinden.

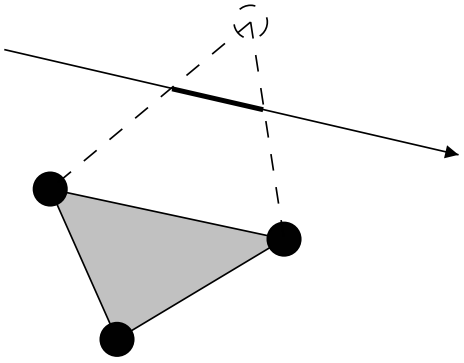


Abbildung 5.4: Nicht zulässiges Anheben eines Fußes

Die Suche nach einem geeigneten Punkt beginnt mit einem begrenzenden Pfadpunkt. Ausgehend von diesem wird der Pfad bis zu dem anderen begrenzenden Pfadpunkt durchsucht. Hat sich der zulässige Bereich verkleinert, muss ein gültiger Punkt innerhalb dieses Bereichs liegen. Hat er sich vergrößert, liegt immer noch der alte zulässige Punkt in dem Bereich und kann verwendet werden.

Durch die kontinuierliche Darstellung der Fußpositionen gibt es entweder keine oder unendlich viele Punkte im zulässigen Bereich. Es ist daher nicht möglich, alle Punkte zu untersuchen. An dieser Stelle lässt sich die Konvexität des zulässigen Bereichs ausnutzen: Liegt keiner der beiden Endpunkte eines Streckenabschnittes innerhalb des Bereichs, sind auch al-

le Punkte dazwischen zulässig. Es genügt daher, nur die Endpunkte der Streckenabschnitte zu betrachten.

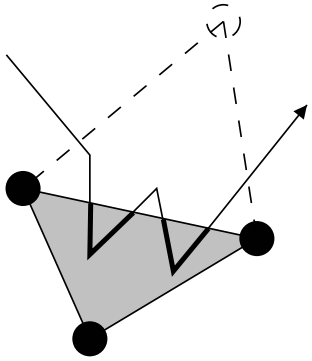


Abbildung 5.5: Entstehung nicht zusammenhängender zulässiger Bereiche

Eine seltene aber mögliche Konstellation ist, dass nach dem Anheben mehr als ein zusammenhängender zulässiger Bereich entsteht. Dies ist der Fall, wenn der Pfad wie in Abbildung 5.5 mehrfach den neuen zulässigen Bereich verlässt. Der Algorithmus gibt *alle* Bereiche zurück. Anschließend wird zufällig (gleichverteilt) ein Bereich gewählt. Da dieser Fall sehr selten auftritt, ist die Wahl der Verteilung auch nicht entscheidend für die Qualität des Ergebnisses. Wird kein Bereich gefunden, so ist das Anheben des Fußes nicht zulässig.

Wahl des Nachfolgezustands

Die Entscheidung, welcher Fuß gehoben oder abgesetzt wird, bestimmt sich aus dem Zustandsgraphen. Der resultierende Folgezustand muss als Nachfolger des aktuellen Zustands im Graphen enthalten sein. Außerdem muss im Fall eines Anhebens die Stabilität gewahrt bleiben. Daher werden alle möglichen Nachfolgezustände entsprechend dieser Kriterien bestimmt. Aus diesen muss zufällig eine Wahl getroffen werden.

Diese Auswahl hat sich als entscheidend für die Qualität der resultierenden Bewegung herausgestellt. Eine einfache gleichverteilte Wahl führt zu sehr schlechten Ergebnissen. Grund dafür ist die Struktur des Zustandsgraphen. Für jede Kante $a \rightarrow b$ existiert auch eine $b \rightarrow a$. Es ist möglich, einen Zustandswechsel sofort wieder rückgängig zu machen, indem die entgegengesetzte Kante anschließend gewählt wird. Ein Fuß wird dann nach dem Aufsetzen sofort wieder angehoben, ohne eine Stützfunktion zu übernehmen. Bei einer Gleichverteilung ergibt sich für die Nachfolgezustände eine Wahrscheinlichkeit von mindestens $\frac{1}{6}$ (bei maximal 6 Nachfolgezuständen). Entsprechend hoch ist auch die Wahrscheinlichkeit der Wahl des vorherigen Zustands, was einer solchen Revision entspricht.

Eine einfache Möglichkeit wäre, den betroffenen Nachfolgezustand auszuschließen. Allerdings wird das Problem dadurch nicht gelöst. Nachdem ein anderer Fuß bewegt wurde, steht der vorherige wieder zur Auswahl. Es wird dann zwar nicht in den exakt gleichen Zustand gewechselt, aber der Fuß kann wieder zurückgesetzt werden. Eigentlich sollte er aber für eine gewisse Zeit stützen beziehungsweise angehoben bleiben.

Es erscheint daher sinnvoll, Füße, die kurz vorher gewählt wurden, weniger wahrscheinlich zu wählen. Füße, die längere Zeit angehoben oder abgesetzt waren, sollten mit einer höheren Wahrscheinlichkeit gewählt werden. Aus diesen Überlegungen entstand das folgende Auswahlverfahren:

Jeder Fuß erhält einen Bonus. Dieser gibt an, wie lange ein Fuß nicht verändert wurde. In jedem Auswahlschritt werden die Boni um eins erhöht, während der Bonus des geänderten Fußes auf Null gesetzt wird. Der Nachfolgezustand wird über eine *Glücksradauswahl* bestimmt, wobei die Einzelwahrscheinlichkeiten gemäß dem Bonus des zu wechselnden Fußes berechnet werden.

Jeder Fuß hat einen Bonus b_i. Er erhält über eine Bewertungsfunktion f einen Wert $f(b_i)$. Die Wahrscheinlichkeit, dass ein Fuß bewegt wird ergibt sich aus seiner Bewertung im Verhältnis zur Summe aller Bewertungen.

$$P(s) = \frac{f(b_s)}{\sum_i f(b_i)} \tag{5.8}$$

Algorithmus 10 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) / sum { get relative value }
end for
choose uniformly random number r ∈ [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
```

Der Nachfolgezustand wird entsprechend Algorithmus 10 berechnet. Als Bewertungsfunktion hat sich

$$f(b_i) = {b_i}^2 \tag{5.9}$$

als günstig herausgestellt.

Nach der Auswahl liegt ein neuer Stützzustand vor. Im Falle eines Absetzens wird auch eine neue Fußposition, wie beschrieben, festgelegt. Hierdurch ändert sich gegebenenfalls der zulässige Bereich auf dem Mittelpunktpfad. Er wird daher anschließend neu berechnet.

Es werden solange neue Stützzustände ermittelt, bis das Ende des Mittelpunktpfades erreicht wird. Dies ist eventuell nicht möglich, wenn keine Lösung gefunden werden kann. Ist der Roboter zum Beispiel von hohen Wänden umgeben, so kann er keinen Weg heraus

finden. Daher wird eine maximale Anzahl von Schritten festgelegt. Hat der Roboter bis dahin keine Lösung gefunden, wird die Suche abgebrochen und die aktuelle Bewegung als ungültig markiert.

Die Abfolge der Fußpositionen, Stützzustände und begrenzenden Pfadpunkte wird in einer Liste gespeichert. Diese Datenstruktur beschreibt nun eine Möglichkeit die Füße zu setzen, um den vorgegebenen Zielpunkt zu erreichen.

5.1.4 Festlegen der Mittelpunktpositionen

Bei den bisherigen Berechnungen wurden für den Mittelpunkt nur Intervalle auf dem Pfad angenommen. Für die Beschreibung der Bewegung sind aber konkrete Werte nötig. Nachdem für jeden Stützzustand die Fußpositionen festliegen, erfolgt nun die Festlegung der Mittelpunktposition.

Die Intervalle sind in der Liste aus der vorherigen Berechnung enthalten. Dabei ist durch die vorherige Berechnung sichergestellt, dass sich zwei aufeinander folgende Intervalle überlappen. Nur so ist eine Bewegung des Mittelpunktes möglich.

Gesucht wird nun ein geeigneter Mittelpunkt zu jedem Stützzustand. Dieser muss im Intervall des aktuellen Stützzustands liegen. Wird beim Wechsel in den Zustand ein Fuß aufgesetzt, geschieht dies wie in Abbildung 5.6. Der zulässige Bereich vergrößert sich. Vor dem Erreichen des neuen Zustands gilt noch der alte Bereich, welcher nicht vom Mittelpunkt verlassen werden darf. Dies gilt auch noch zum Zeitpunkt des Aufsetzens. Damit muss der Mittelpunkt immer noch im alten Intervall liegen. Zusätzlich muss er aber auch im neuen zulässigen Bereich liegen. Damit ergibt sich nur der Durchschnitt beider Bereiche (dicke schwarze Linie) als erlaubt für die neue Mittelpunktposition.

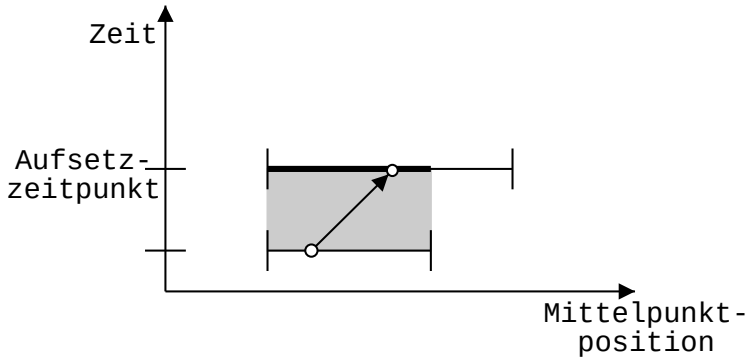


Abbildung 5.6: zulässiger Bereich während Aufsetzens

Wird beim Wechsel in den neuen Zustand ein Fuß angehoben, ergibt sich eine ähnliche Bedingung (Abbildung 5.7). Der Fuß kann bis zum Erreichen des neuen Zustands außerhalb des Zielintervalls (graue Fläche) liegen. Am Ende muss er sich aber auch innerhalb des Zielintervalls befinden. Da dieses beim Anheben kleiner als das Startintervall ist, kann auch hier einfach der Durchschnitt beider verwendet werden.

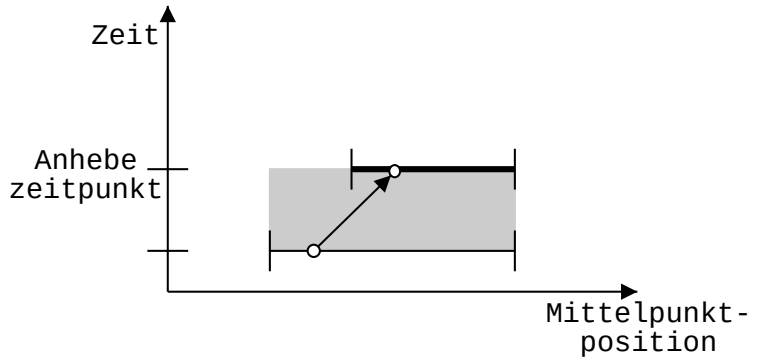


Abbildung 5.7: zulässiger Bereich während Anhebens

Der Mittelpunkt für einen Zustand kann also aus dem Durchschnitt der Intervalle des aktuellen und vorherigen Zustands gewählt werden. Würde dies einfach zufällig gleichverteilt erfolgen, ergäben sich eventuell unerwünschte Ergebnisse.

Der Mittelpunkt würde beliebig innerhalb der Bereiche platziert sein, was wie in Abbildung 5.8 zu unnötigen Hin-und-Her-Bewegungen führen kann. Entsprechend viel Zeit benötigt der Roboter dann für die Bewegung. Besser für die Dauer wäre hier eine lineare Bewegung, wie durch die gestrichelte Linie angedeutet. Da aber auch der Stability Margin von der Wahl der Mittelpunktposition beeinflusst wird, ist eine deterministische Auswahl nicht zulässig. Einen Kompromiss stellt hier die Auswahl über eine Dreiecksverteilung dar, wobei die alte Mittelpunktposition als Erwartungswert dient. Dadurch werden kurze Bewegungen wahrscheinlicher, und damit auch die Gesamtdauer.

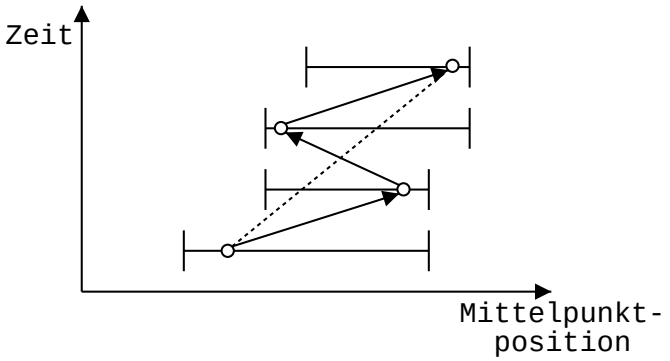


Abbildung 5.8: mögliche Mittelpunktbewegung bei reiner Zufallswahl

Durch die Wahl des Mittelpunktes wird nicht nur die Dauer beeinflusst, sondern auch der Stability Margin. Er bestimmt unter anderem die Weite des zulässigen Bereichs. Wird ein Punkt am Rand des Intervalls gewählt, führt dies in der Regel zu einem geringeren Stability Margin als ein Wert aus dem mittleren Bereich. Da dies für jeden Zustand geschieht, ist die Wahrscheinlichkeit, dass einmal ein Punkt nahe dem Rand gewählt wird, sehr hoch. Daraus resultiert eine geringe Wahrscheinlichkeit für einen großen minimalen Stability Margin.

Eine Möglichkeit, solches Verhalten zu verhindern, ist das *Erzwingen* einer Wahl nahe der

Mitte der Intervalle bzw. ein minimaler Abstand der Mittelpunkte vom Rand. Natürlich muss auch dies wieder zufällig geschehen. Der minimale Randabstand wird zufällig gleichverteilt zwischen Null und der Hälfte des kleinsten Intervalls gewählt. Anschließend wird von jedem Intervall der ermittelte Wert abgezogen. Hierdurch gelingt es, die Wahrscheinlichkeit eines großen minimalen Stability Margin drastisch zu erhöhen.

5.1.5 Berechnung der Dauer der Bewegungen

Nachdem die Positionen des Mittelpunktes festgelegt wurden, wird zur Beschreibung der Bewegung nur noch die Zeit zwischen den einzelnen Zuständen benötigt. An dieser Stelle kann bis auf die Dauer der Bewegung keines der Bewertungskriterien mehr beeinflusst werden. Es ist daher möglich, die Dauer einzeln zu optimieren. Ein deterministisches, exaktes Verfahren kann angewendet werden.

Der Zeitpunkt, zu dem ein neuer Zustand frühestens angenommen werden kann, hängt von zwei Größen ab:

- die Zeit, die der Mittelpunkt mindestens von der alten Position zur neuen benötigt,
- im Fall des Aufsetzens eines Fußes, die Mindestdauer der Bewegung von der alten Fußposition zur neuen.

Die Zeiten sollen minimiert werden, müssen aber beide Bedingungen einhalten. Deshalb wird von beiden einzuhaltenden Zeiten das Maximum gebildet. Die sich ergebende Zeit wird als Dauer zwischen den Zuständen gewählt.

Somit sind alle Werte zur Beschreibung einer Bewegung bestimmt. Die Datenstruktur für die Bewegung wird einfach aus den Werten aufgebaut. Somit liegt eine zufällige zulässige Bewegung in der gewünschten Form vor.

5.2 Implementierung und erste Bewertung

Dieser Abschnitt beschreibt die Implementierung und erste Bewertung, die für den Random-Sampling-Algorithmus vorgenommen wurden. Hierdurch wird es möglich, den Algorithmus praktisch zu testen und Aussagen über die Leistungsfähigkeit des Verfahrens zu treffen.

5.2.1 Implementierung

Es wurde der Algorithmus, wie er in Abschnitt 4.3 beschrieben ist, implementiert. Zum Erzeugen zufälliger Lösungen dient das in Abschnitt 5.1 beschriebene Verfahren. Die Implementierung erfolgt in C++ unter Linux. Da sowohl für den PC als auch für die Mikrocontroller ein C++-Compiler zur Verfügung steht, muss der Code nur geringfügig angepasst werden, um auch auf letzterem lauffähig zu sein.

5.2.2 Bewertung des Algorithmus

Beim Random Sampling handelt es sich um einen randomisierten Algorithmus. Das heißt, der Ablauf ist nicht deterministisch, sondern wird zum Teil durch eine Zufallsgröße bestimmt. Als Konsequenz ist auch das Ergebnis der Berechnung nicht deterministisch. Für die gleiche Eingabe können unterschiedliche Ausgaben entstehen. Eine Bewertung kann deshalb nicht auf Basis eines einzelnen Laufs getroffen werden. Vielmehr muss eine statistische Auswertung erfolgen.

Für die Bewertung wurde vorerst nur ein Szenario gewählt. Dies sollte für einen ersten Eindruck ausreichend sein. Die Untersuchung anderer Szenarien wird in einem späteren Kapitel durchgeführt. Es wird an dieser Stelle das Laufen auf flachem Untergrund betrachtet. Das Gelände im Weltmodell ist eben. Die Aufgabe des Roboters ist es, ein 5 m in Blickrichtung entferntes Ziel zu erreichen. Dieses einfache Szenario wurde gewählt, um möglichst wenig ungültige Lösungen zu erhalten. Dadurch stehen mehr gültige für die Auswertung zur Verfügung.

Die Beurteilung erfolgt nach zwei Gesichtspunkten:

1. Wie schnell wird eine Lösung gefunden?
2. Wie gut sind die gefundenen Lösungen?

Die Qualität des Algorithmus hängt lediglich von den zufällig generierten Lösungen ab. Deshalb wird die entsprechende Funktion hier untersucht. Es werden 10 000 Lösungen erzeugt und statistisch ausgewertet. Aus den Ergebnissen lassen sich dann Rückschlüsse auf den gesamten Algorithmus ziehen.

Finden einer Lösung

Wenn eine Lösung für ein Szenario existiert, wird diese vom Algorithmus nach einer bestimmten Zeit gefunden. Die Anzahl der dafür benötigten Schritte kann aber beliebig hoch werden. Damit würden aber auch die Ereignislisten entsprechend lang werden, was bedingt durch den endlichen Speicher des Rechners nicht zulässig ist. Deshalb wird die Berechnung abgebrochen, wenn die Schrittanzahl einen bestimmten Wert überschreitet. Hier wurde 400 als Limit gewählt. Die Bewegung ist im Falle eines Abbruchs nicht vollständig und wird für ungültig erklärt.

Im durchgeführten Test waren 2259 Bewegungen von 10 000 gültig. Diese Zahl mag sehr gering erscheinen, ist aber auf die vorgegebene maximale Schrittanzahl zurückzuführen. Durch Heraufsetzen des Limits könnte sie gesteigert werden. Andererseits kommt eine Laufbewegung, die mehr als 400 Schritte für 5 m benötigt, kaum als gutes Ergebnis in Frage.

Die Wahrscheinlichkeit, eine gültige Lösung zu finden, wird nicht nur durch die Anzahl der gültigen generierten Bewegungen bestimmt. Durch mehrfache Ausführung steigt die Wahrscheinlichkeit erheblich. Wird eine einzelne Bewegung erzeugt, liegt die Wahrscheinlichkeit einer gültigen Lösung bei $p = \frac{2259}{10\,000} = 0,225\,9$. Bei n-maliger Ausführung kann sie über

$$P(\text{n-mal}) = 1 - (1 - p)^n \tag{5.10}$$

bestimmt werden. In der folgenden Tabelle sind einige Werte für unterschiedliche n dargestellt.

n	*P(n-mal)*
1	0,225 9
2	0,400 769 19
4	0,640 922 4
10	0,922 737
50	0,999 997 246 7
100	0,999 999 999 992 419

Man sieht deutlich, wie die Wahrscheinlichkeit mit größeren n steigt. Schon ab 50 erzeugten Bewegungen ist sie fast 1. Da sich eine solche Anzahl sehr schnell generiert lässt, wird mit fast vollständiger Sicherheit eine Lösung gefunden. Der Algorithmus kann also sehr gut eine gültige Lösung für dieses Szenario ermitteln.

Güte der Lösungen

Die Güte der am Ende gefundenen Lösung hängt von zwei Faktoren ab – der Wahrscheinlichkeit für eine entsprechend gute Lösung und der Anzahl der erzeugten Bewegungen. Für die hier vorgenommene Messung ist die Verteilung der Bewertungen in Abbildung 5.9 dargestellt. Man erkennt, dass mittelmäßig bewertete Bewegungen am häufigsten generiert werden. Besonders gute oder besonders schlechte treten seltener auf.

Hieraus darf aber nicht geschlussfolgert werden, dass der Algorithmus am Ende meist mittelmäßige Ergebnisse liefert. Entscheidend ist die Anzahl der generierten Lösungen. Die Wahrscheinlichkeit, unter einem bestimmten Anteil der besten Bewegungen zu liegen, kann wie in Abschnitt 4.3 berechnet werden. Die Werte für die hier betrachtete Messung sind in Tabelle 5.3 zu sehen. Hiermit ist es möglich, die Wahrscheinlichkeit bei einer gegebenen Anzahl an Iterationen zu bestimmen. Beispielsweise liegt die Wahrscheinlichkeit, bei 100 Iterationen eine Lösung aus den besten 1% zu finden, bei etwa 0,2. Die konkreten Bewertungen zu den Anteilen lassen sich über die grauen Linien in Abbildung 5.9 identifizieren

Anteil an allen gültigen Bewegungen	*Anzahl der Iterationen*				
	1	10	100	1000	10 000
100%	0,226	0,923	≈ 1	≈ 1	≈ 1
50%	0,113	0,698	≈ 1	≈ 1	≈ 1
10%	0,022 6	0,204	0,898	≈ 1	≈ 1
1%	0,002 26	0,022 4	0,202	0,895	≈ 1
0,1%	0,000 226	0,002 26	0,022 3	0,202	0,895

Tabelle 5.3: Wahrscheinlichkeit guter Bewegungen (flaches Gelände)

Bei einer üblichen Iterationsanzahl von 1000 liegt die beste gefundene Lösung höchstwahrscheinlich unter den besten 10% aller gültigen Lösungen. Der Algorithmus liefert in diesem Fall nicht nur gültige Lösungen, sondern auch solche mit einer guten Bewertung. Damit genügt er den Anforderungen aus der Aufgabenstellung.

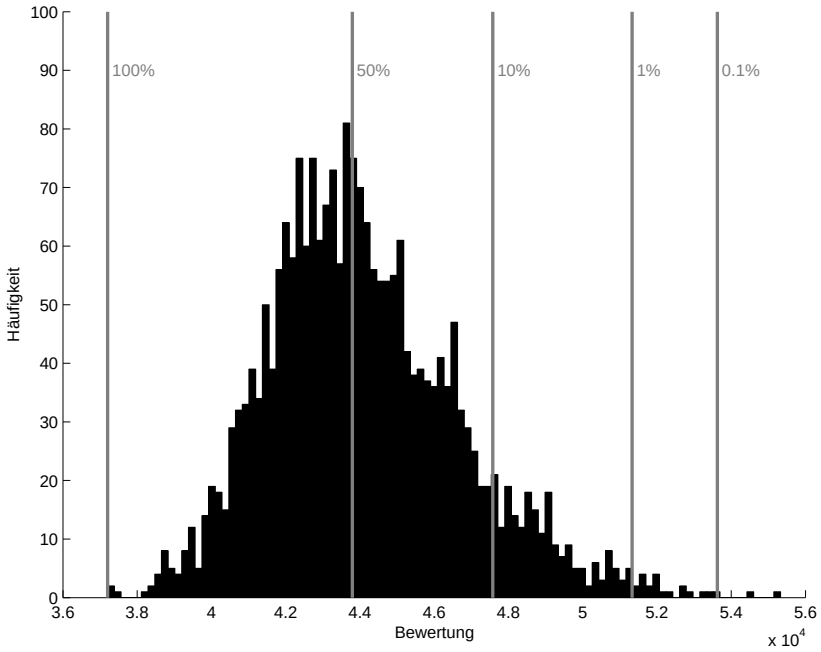


Abbildung 5.9: Histogramm Szenario „flaches Gelände"

6 Lokale Suche und Simulated Annealing

Dieses Kapitel behandelt die Verfahren Simulated Annealing und Lokale Suche, die bereits in den Abschnitten 4.6 und 4.4 vorgestellt wurden. Die zwei Verfahren sind sehr ähnlich, weshalb sie auch in einem gemeinsamen Kapitel behandelt werden. So ist für beide eine Funktion zum Erzeugen zufälliger Lösungen und eine Nachbarschaftsdefinition nötig. Das Erzeugen zufälliger Lösungen wurde bereits beim Random Sampling in Kapitel 5 behandelt. Die Definition einer Nachbarschaft soll im Folgenden geschehen. Außerdem wird noch eine Methode zur Enumeration der Nachbarschaft beschrieben, die bei der Implementierung des Simulated Annealing sehr nützlich ist. Zuletzt erfolgt eine kurze Beschreibung der Implementierung und Bewertung der zwei Verfahren.

6.1 Definition einer Nachbarschaft

Für die Lokale Suche und Simulated Annealing wird die *Definition einer Nachbarschaft* benötigt. Sie dient dem Durchsuchen des Lösungsraums. Von einer gültigen Lösung ausgehend werden weitere erzeugt. Ja nach Güte der neuen Lösung wird diese dann als Ausgangspunkt der nächsten Iteration verwendet.

Normalerweise ist die Definition einer Nachbarschaft recht einfach. Ein Nachbar kann erzeugt werden, indem ein Teil einer Lösung ein wenig verändert wird. Die Menge aller so erzeugbaren Lösungen wird dann als Nachbarschaft verwendet. Durch mehrmaliges Anwenden der Erzeugungsregel soll es möglich sein, ein beliebiges Element des Lösungsraums zu konstruieren.

6.1.1 Diskretisierung der Nachbarschaft

Aus der kontinuierlichen Beschreibung in Teilen einer Lösung (z.B. Fußposition $\in \mathbb{R}^2$) resultiert, dass es unendlich viele benachbarte Lösungen gibt. Die Nachbarschaft müsste damit unendlich viele Elemente enthalten. Da bei der Lokalen Suche aber *alle* Nachbarn betrachtet werden, ist eine derartige Definition nicht möglich. Um eine geeignete *endliche* Menge zu finden, ist eine Diskretisierung der Größen notwendig. Unter Betrachtung der späteren Verwendung kann man solch ein Vorgehen auch vertreten. Beispielsweise lassen sich Fußpositionen in diskreten Abständen von 5 mm beschreiben. Dies ist für die spätere Verwendung noch genau genug. Auch Zeitausdrücke lassen sich mit einer Granularität von 0,5 Sekunden beschreiben, ohne damit praktisch relevante Lösungen auszuschließen.

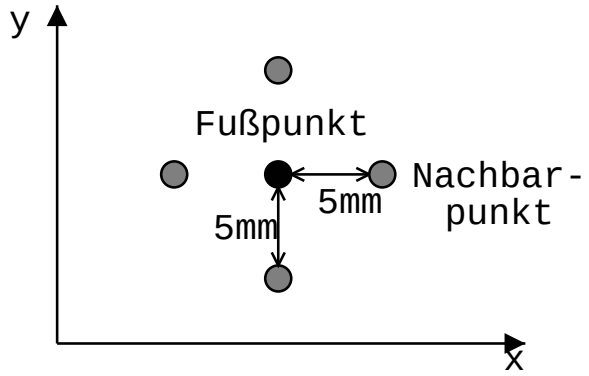


Abbildung 6.1: Diskrete Nachbarschaft der Fußposition

Für die Fußpunkte wird eine Vierer-Nachbarschaft verwendet. Wie in Abbildung 6.1 wird dabei die Position um einen Wert von hier 5 mm in jeweils positiver oder negativer Richtung verschoben. In den zwei Dimensionen ergeben sich damit die dargestellten vier Möglichkeiten.

Die Zeitangaben sind der Startzeitpunkt und die Dauer der Ereignisse. Eine benachbarte Bewegung wird erzeugt, indem die Zeiten um einen geringen Wert (hier 0,5 Sekunden) erhöht oder verringert werden. Für die zwei Zeitgrößen ergeben sich damit jeweils zwei Möglichkeiten, also insgesamt vier pro Ereignis.

6.1.2 Aufspalten und Verschmelzen von Ereignissen

Aus einer Lösung sollte man über die Nachbarschaftsrelation durch mehrmaliges Anwenden zu jedem beliebigen Element des Lösungsraums gelangen können. Dies ist allein durch Ändern der Fußpositionen oder der Zeiten in den Ereignislisten nicht möglich. Die Anzahl der Ereignisse wird durch die bisher erwähnten Operationen nicht beeinflusst. Beispielsweise kann damit aus einer Bewegung mit zehn Ereignisse nie eine mit zwölf erzeugt werden.

Es soll aber gewährleistet werden, dass in der Nachbarschaft einer Lösung auch solche möglich sind, die eine andere Ereignisanzahl besitzen. Dazu müssen Ereignisse *eingefügt* beziehungsweise *gelöscht* werden. Zu beachten ist aber, dass ein wahlloses Einfügen oder Löschen innerhalb der Ereignislisten nicht unbedingt die gewünschten Ergebnisse bringt. Ein Ereignis beansprucht aufgrund der Bewegungsdauer eine relevante Zeitspanne. Da die Zeiten in den nachfolgend Ereignissen relativ zum Ende des aktuellen interpretiert werden, würden sie durch die geänderte Zeit verschoben werden. Von *kleinen Änderungen* kann man hier nicht mehr ausgehen, da sich das gesamte nachfolgende Verhalten ändert. Oft wird die Bewegung dadurch ungültig, da die Zeiten in den anderen Ereignislisten unverändert bleiben. Es werden also andere Operationen benötigt.

Aufspalten von Ereignissen

Die Anzahl der Ereignisse lässt sich erhöhen, indem ein einzelnes Ereignis entfernt, und stattdessen zwei neue eingefügt werden. Haben die neuen Ereignisse zusammen eine Gesamtdauer, die der des alten Ereignisses entspricht, findet keine Verschiebung der Nachfolger statt. Ein solches Aufspalten bedeutet bezogen auf ein Fußpunktereignis, dass die gleiche Bewegung wie vorher ausgeführt wird. Der Fuß wird lediglich zwischendurch noch einmal aufgesetzt. Für ein Mittelpunktereignis bewirkt es kein anderes Verhalten. Dieses Vorgehen entspricht einer kleinen Änderung im Bewegungsablauf, weshalb es hier verwendet wird.

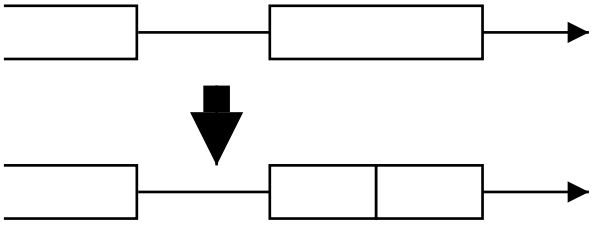


Abbildung 6.2: Aufspalten von Ereignissen

Ein Ereignis ist definiert über Startzeitpunkt, Zielposition und Dauer.

$$E = (t_{\text{start}}, p_{\text{dest}}, t_{\text{dur}}) \tag{6.1}$$

Beim Aufspalten müssen diese Werte für die neuen Ereignisse angegeben werden. Wie dies geschieht, ist nicht zwingend festgelegt. Hier wurde eine recht einfache Variante gewählt:

Der Startzeitpunkt des ersten neuen Ereignisses entspricht dem Startzeitpunkt des ursprünglichen. Somit ist sichergestellt, dass der Fuß für die gleiche Dauer den Körper stützt. Für die Dauer wird die Hälfte des ursprünglichen Wertes gewählt. Als Zielposition wird die Mitte zwischen der vorherigen Fußposition und der alten Zielposition festgelegt. Damit setzt der Fuß in der Mitte der Strecke noch einmal auf.

$$E_1 = \left(t_{\text{start}}, \frac{p_{\text{dest}} + p_{\text{dest}}^{-1}}{2}, \frac{t_{\text{dur}}}{2} \right) \tag{6.2}$$

Das zweite Ereignis beginnt direkt nach dem ersten. Dementsprechend wird die Startzeit auf 0 gesetzt. Die Zielposition entspricht der alten Zielposition. Damit die Dauer beider Ereignisse gleich der des ursprünglichen ist, muss die Ereignisdauer auch hier auf die Hälfte der ursprünglichen gesetzt werden.

$$E_2 = \left(0, p_{\text{dest}}, \frac{t_{\text{dur}}}{2} \right) \tag{6.3}$$

Das ursprüngliche Ereignis wird durch die beiden beschriebenen Ereignisse ersetzt.

Verschmelzen von Ereignissen

Eine Reduktion der Ereignisanzahl lässt sich erzielen, indem aufeinander folgende Ereignisse zusammengefasst werden. Wenn die Dauer des resultierenden Ereignisses der Gesamtdauer der ursprünglichen entspricht, entsteht ebenfalls keine Verschiebung der Nachfolger.

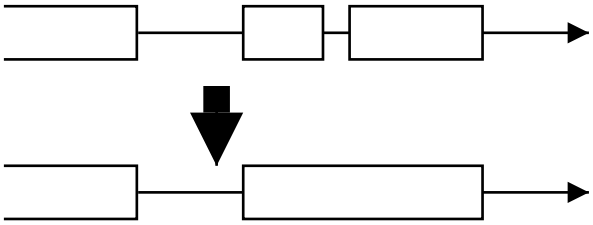


Abbildung 6.3: Verschmelzen von Ereignissen

Es sind unterschiedliche Varianten möglich. Für zwei aufeinander folgende Ereignisse E_1 und E_2 mit

$$E_1 = (t^1_{\text{start}}, p^1_{\text{dest}}, t^1_{\text{dur}}) \tag{6.4}$$

$$E_2 = (t^2_{\text{start}}, p^2_{\text{dest}}, t^2_{\text{dur}}) \tag{6.5}$$

wird ein neues Ereignis

$$E = (t^1_{\text{start}}, p^2_{\text{dest}}, t^1_{\text{dur}} + t^2_{\text{start}} + t^2_{\text{dur}}) \tag{6.6}$$

eingefügt.

Für ein Fußereignis bedeutet dies, dass der Fuß zwischendurch nicht aufgesetzt wird, sondern angehoben bleibt. Für ein Mittelpunktereignis gilt, dass der Zielpunkt des ersten Ereignisses nicht mehr erreicht wird, sondern sofort der des zweiten angesteuert wird. Die neue Ausführungszeit entspricht der Summe der zwei Ausgangsereignisse, was eine Verschiebung der Nachfolger verhindert.

Operationen zum Erzeugen von Nachbarn

Durch diskrete Änderungen der Größen in Ereignissen und den Möglichkeiten zum Verändern der Ereigniszahl ergeben sich nun die folgenden Operationen zum Erzeugen von benachbarten Lösungen:

- Verschieben der Zielposition eines Ereignisses um 5 mm in x-Richtung,
- Verschieben der Zielposition eines Ereignisses um 5 mm in y-Richtung,
- Verschieben der Zielposition eines Ereignisses um -5 mm in x-Richtung,
- Verschieben der Zielposition eines Ereignisses um -5 mm in y-Richtung,
- Erhöhen der Dauer eines Ereignisses um 0,5 Sekunden,
- Verringern der Dauer eines Ereignisses um 0,5 Sekunden,

- Erhöhen der Startzeit eines Ereignisses um 0,5 Sekunden,
- Verringern der Startzeit eines Ereignisses um 0,5 Sekunden,
- Aufspalten eines Ereignisses,
- Verschmelzen zweier aufeinander folgender Ereignisse.

Diese Operationen können auf jedes Ereignis der Bewegung angewendet werden. Eine Ausnahme stellt nur die letzte Operation dar, da sie einen Nachfolger benötigt. Die jeweils letzten Ereignisse einer Liste besitzen jedoch keinen. Je nach Anzahl der Ereignisse ergibt sich damit eine entsprechend große Nachbarschaft. Für eine Bewegung mit insgesamt n Ereignissen in den sieben Ereignislisten erhält man $10n - 7$ Nachbarlösungen. Für $n = 100$ wären das beispielsweise 993.

6.2 Enumeration der Nachbarschaft

In einem Schritt des Simulated Annealing wird zufällig ein Element der Nachbarschaft ausgewählt. Da die Menge aber sehr groß ist, erscheint es nicht sinnvoll, alle Nachbarn vorab zu erzeugen. Vielmehr sollte nur der erzeugt werden, der später auch Verwendung findet. Ein solches Vorgehen lässt sich durch *Enumeration der Nachbarschaft* realisieren. Jedes Element der Nachbarschaft erhält eine Nummer. Anschließend wird zufällig eine Zahl gewählt, die einem der Nachbarn entspricht. Diese kann anschließend aus der Nummer abgeleitet und erzeugt werden.

Die Enumeration stellt also eine eineindeutige Abbildung der Nachbarschaft N auf die natürlichen Zahlen zwischen 1 und der Anzahl der Nachbarn dar.

$$\sharp : N \mapsto [1, |N|] \cap \mathbb{N} \tag{6.7}$$

$$\sharp^{-1} : [1, |N|] \cap \mathbb{N} \mapsto N \tag{6.8}$$

Diese Abbildung wird wie folgt realisiert: Es werden sieben Listen verwaltet, sechs für die Fußpunkt-Ereignisse und eine für den Mittelpunkt. Auf diesen Listen können die gleichen Operationen angewendet werden, weshalb sie gleich behandelt werden. Zuerst wird die Enumeration der Nachbarn betrachtet, die durch Manipulation einer der Listen entsteht.

Durch Verändern einer Liste der Länge l können bei den zehn möglichen Operationen genau $10l - 1$ Nachbarn erzeugt werden. Dabei wird eine Operation auf eines der Ereignisse der Liste angewendet. Das Verschmelzen mit dem Nachfolger ist für das letzte Ereignis nicht möglich. Die Operation für eine Nummer n wird bestimmt, indem n durch die Länge der Liste dividiert wird:

$$op = \frac{n}{l}. \tag{6.9}$$

Man erhält dabei eine Zahl zwischen 0 und 9, die der Bestimmung der Operation dient. Die Auswahl des Ereignisses e erfolgt über $e = l \bmod n$. Man erhält hiermit einen Wert zwischen 0 und $l - 1$. Da $n < 10l - 1$ gilt, kann für die Operation 9 nie das letzte Ereignis $l - 1$ gewählt werden. Somit ist auch die Korrektheit in diesem Fall garantiert.

Die Enumeration nach dem obigen Schema bildet auf die Zahlen 0 bis $10l-2$ die entsprechende Nachbarschaft ab. Die Enumeration der gesamten Nachbarschaft geschieht nun, indem die jeweiligen Nachbarschaften aneinander gehangen werden. Hierbei entspricht die Liste der Mittelpunkt-Ereignisse der Liste 0. Die Listen 1 bis 6 sind die der jeweiligen Fußpunkt-Ereignisse. Die Abgrenzung erfolgt über

$$s_i = \begin{cases} 0 & : \quad i = -1 \\ s_{i-1} + 10l_i - 1 & : \quad 0 \leq i \leq 6 \\ \textit{nicht definiert} & : \quad \text{sonst} \end{cases} \tag{6.10}$$

Die Nachbarn, die über eine Liste i erzeugt werden, haben immer Zahlenwerte zwischen s_{i-1} und $s_i - 1$. Auf diese Weise wird die gesamte Nachbarschaft enumeriert.

6.3 Implementierung und Bewertung

Dieser Abschnitt beschreibt die Implementierung und Bewertung, die für die beiden Verfahren vorgenommen wurde. Die Algorithmen wurden praktisch getestet und können somit hinsichtlich ihrer Eignung beurteilt werden.

6.3.1 Implementierung

Die Algorithmen wurden implementiert, wie in den Abschnitten 4.6 und 4.4 beschrieben. Zum Erzeugen der zufälligen Startlösungen dient das in Abschnitt 5.1 erläuterte Verfahren, welches auch beim Random Sampling Anwendung findet. Die in Abschnitt 6.2 beschriebene Enumeration wird zum Erzeugen von Nachbarlösungen verwendet. Dafür steht eine Funktion zur Verfügung, die aus einer gegebenen Nummer die dazugehörige Lösung erzeugt. Bei der Lokalen Suche werden alle Nummern durchlaufen und somit die komplette Nachbarschaft betrachtet. Beim Simulated Annealing wird, dem Algorithmus entsprechend, eine zufällige Nummer gewählt.

Die Implementierung erfolgte wie schon beim Random Sampling in C++ unter Linux.

6.3.2 Bewertung der Lokalen Suche

Die Bewertung der Lokalen Suche und des Simulated Annealing soll im Vergleich zum bereits beschriebenen Random Sampling erfolgen. Daher wird hier das gleiche Szenario (Laufen auf flachem Gelände) verwendet. Da bereits durch das Random Sampling ein geeignetes Verfahren vorliegt, ist die Fragestellung, inwieweit die anderen beiden Verfahren Vorteile bieten.

Die Lokale Suche verfolgt einen ähnlichen Ansatz wie das Random Sampling. Es werden zufällige Bewegungen generiert und die beste als Ergebnis geliefert. Der Unterschied besteht darin, dass die Lokale Suche eine generierte Bewegung zusätzlich noch durch lokale Änderungen zu verbessern versucht. Dazu wird die Nachbarschaft betrachtet.

Eine Bewegung kann je nach Länge der Ereignislisten eine sehr hohe Anzahl an Nachbarn besitzen. Durch deren Betrachtung entsteht ein zusätzlicher Rechenaufwand. Zu klären ist, ob die Ergebnisse diesen rechtfertigen. Dazu werden für das gewählte Szenario Berechnungen mit dem Algorithmus durchgeführt. Von Interesse sind vor allem die Anzahl der zu betrachtenden Nachbarlösungen und der gewonnene Zuwachs bei der Bewertung der Lösungen.

Bei dem Test wurde der Algorithmus 2177-mal gestartet. Dabei waren 609 Startlösungen gültig und wurden weiter verfolgt. Die Bewegung wurde iterativ verfeinert durch Betrachtung der Nachbarschaft. Insgesamt geschah dies 9389-mal mit 52 127 767 Nachbarlösungen. Damit liegt die durchschnittliche Größe der Nachbarschaft bei 5552.

Dem Algorithmus gelingt es tatsächlich, die Lösungen zu verbessern. Im Durchschnitt wurde die Startlösung um 1,8% verbessert. Der maximale Wert lag bei 8,4%. Dazu waren im Schnitt 15 Verbesserungsschritte nötig.

Für die Beurteilung muss aber auch der Aufwand der Verbesserung betrachtet werden. In einem Iterationsschritt werden etwa 5500 Lösungen betrachtet. Bei 15 Schritten müssen pro Startlösung etwa 82 500 Nachbarlösungen betrachtet werden. Damit wird bei gleicher Anzahl an Iterationen 82 500-mal soviel Rechenzeit benötigt wie beim Random Sampling. Auch wenn eine Verbesserung der Lösung gelingt, ist doch die aufzuwendende Rechenzeit unverhältnismäßig hoch. Damit kann der Algorithmus auch nicht praktisch eingesetzt werden. Er ist also nicht geeignet.

6.3.3 Bewertung des Simulated Annealing

Das Simulated Annealing arbeitet ähnlich der Lokalen Suche. Es werden Startlösungen erzeugt, die anschließend verbessert werden. Auch hier stellt sich die Frage, inwieweit der Aufwand für die Verbesserung durch die Ergebnisse gerechtfertigt ist. Der Algorithmus wurde auf das bekannte Szenario angewendet. Als Temperatur-Funktion diente ein einfaches Dekrementieren, also Verringern um eins. Der Startwert lag bei 1000.

Es wurden 1550 Iterationen vorgenommen. Bei 461 lag eine gültige Startlösung vor, so dass sie weiter betrachtet wurden. Die durchschnittliche Bewertung der untersuchten Startlösungen lag bei 44 169. Durch das Simulated Annealing fand eine Verbesserung statt, so dass die durchschnittliche Bewertung der ermittelten Bewegungen bei 44 196 liegt. Die maximale Verbesserung beträgt 491,3. Bei 325 der 461 Bewegungen fand überhaupt keine Erhöhung des Wertes statt.

Bedenkt man den viel höheren Rechenaufwand, der beim Simulated Annealing entsteht, ergeben sich kaum Vorteile gegenüber dem einfachen Random Sampling. Die nachträgliche Verbesserung der Startlösungen bringt fast keinen Gewinn. Dafür wird hier pro Iteration um den Faktor 1000 mehr Rechenzeit benötigt. Das Verfahren scheint daher nicht geeignet.

Von den drei untersuchten Ansätzen hat sich in der Praxis nur das Random Sampling als geeignet herausgestellt. Sowohl Lokale Suche als auch Simulated Annealing benötigen zum Berechnen guter Lösungen unverhältnismäßig viel Rechenzeit. Für die Online-Planung sind

sie damit nicht verwendbar. Als Laufplaner kommt deshalb der auf Random Sampling basierende Algorithmus zum Einsatz.

7 Bewertung des Laufplaners

Die Ergebnisse des Bewegungsplaners sollen auf dem Roboter getestet werden. So kann sichergestellt werden, dass sie wirklich korrekt arbeiten. Zum Zeitpunkt des Entstehens dieser Arbeit war aber ein Teil der Systemsoftware des LAURON III noch nicht verfügbar. Deshalb konnten auch noch keine Tests am realen Roboter durchgeführt werden. Um dennoch einen Eindruck von den berechneten Bewegungen zu erhalten, wurde eine Visualisierung entwickelt. Mit Hilfe dieser Software können die Bewegungen anhand eines dreidimensionalen Modells dargestellt werden. Hierdurch war es möglich, Bewegungen zu berechnen und anschließend visuell auszuwerten.

7.1 Aufbau der Testumgebung

Anhand der Testumgebung wurden die drei in dieser Arbeit beschriebenen Verfahren untersucht. Die grobe Struktur des Systems ist in Abbildung 7.1 dargestellt. Ein Test besteht aus zwei Schritten – der Berechnung und der anschließenden Auswertung der Bewegung. Dementsprechend ist auch die Testumgebung aufgeteilt. Sie besteht aus der berechnenden Komponente mit dem eingebetteten Laufplaner und der Visualisierung zum Darstellen der Ergebnisse. Die Funktionsweise wird im Folgenden beschrieben.

7.1.1 Einbettung des Laufplaners

Der Laufplaner dient dem Erzeugen von Bewegungen, die anschließend dargestellt werden. Normalerweise wird er in der in Abschnitt 2.2.3 beschriebenen Umgebung eingesetzt. Diese steht aber nicht zur Verfügung und muss deshalb simuliert werden. Dazu werden die Gegenstellen für die Schnittstellen der Ein- und Ausgabe bereitgestellt.

Eingabe

Die vom Laufplaner erwartete Eingabe besteht aus Start- und Zielposition sowie dem Weltmodell in Form von Höhendaten. Die Start- und Zielposition wird dem Laufplaner einfach als Parameter übergeben. Das Weltmodell wird in Form von Rasterbildern geliefert. Dabei entsprechen die Pixel einzelnen Höhenpositionen. Der Farbwert wird auf einen Höhenwert abgebildet. Je heller ein Pixel, umso höher das Gelände an der Stelle. Auf diese Weise lassen sich sehr einfach Szenarien entwerfen. Die Rasterbilder können mit einer herkömmlichen Bildverarbeitungssoftware erzeugt werden.

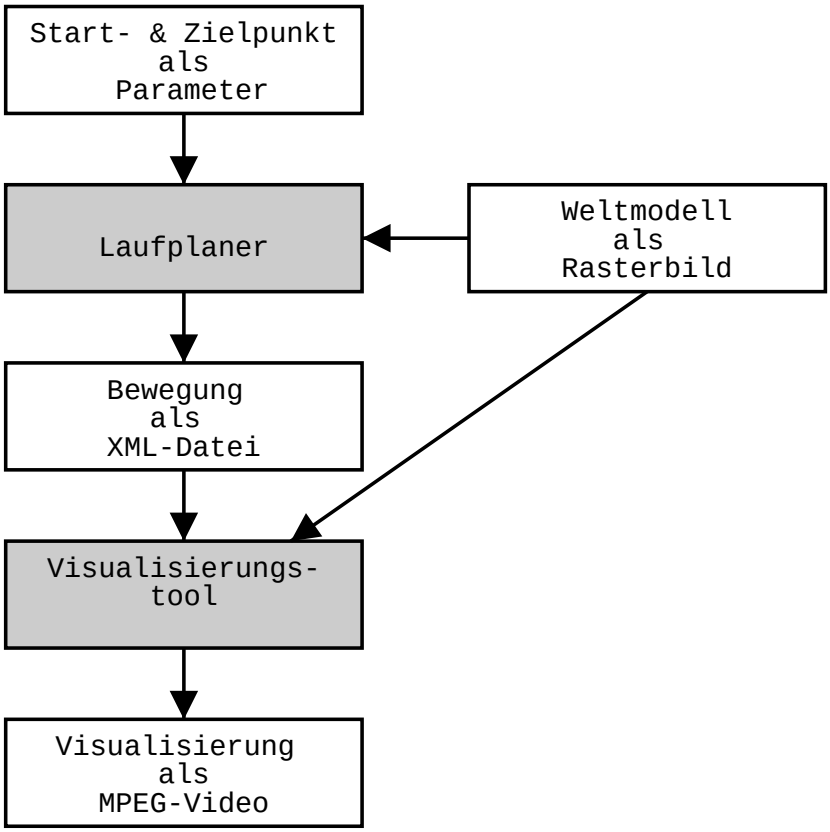


Abbildung 7.1: Aufbau der Testumgebung

Ausgabe

Die Laufplaner erzeugt als Ausgabe die in Abschnitt 3.3 beschriebene Datenstruktur. Sie wird mittels LibXml2 [45] als XML-Datei ausgegeben. Die Ausgabe in Dateien erlaubt das dauerhafte Speichern einmal berechneter Bewegungen. Außerdem können die Daten auf diese Weise einfach an andere Programme übergeben werden. Hier dient sie der Übergabe der Daten an die Visualisierung.

7.1.2 Visualisierung

Die Visualisierung (vgl. Abbildung 7.2) stellt das Ausführen einer Bewegung grafisch dar. Dies erfolgt anhand eines originalgetreuen 3D-Modells des LAURON III. Zusätzlich zum Roboter wird noch das Gelände eingeblendet. Die Darstellung geschieht mit Hilfe der 3D-Bibliothek OpenInventor [33]. Sie ermöglicht den Aufbau komplexer Strukturen aus einfachen geometrischen Formen, was den Entwicklungsaufwand erheblich reduziert.

Abbildung 7.2: Screenshot der Visualisierung

Eingabe

Für die Visualisierung werden die Bewegungsbeschreibung und das dazugehörige Weltmodell benötigt. Die Bewegungen werden in dem vom Laufplaner verwendeten XML-Format eingelesen. Das Weltmodell wird wie schon beim Laufplaner aus dem jeweiligen Rasterbild aufgebaut.

Ausgabe

Die Visualisierung liefert eine dreidimensionale Darstellung. Dabei wird die Bewegung in Echtzeit wiedergegeben. Die Szene kann beliebig gedreht, verschoben, vergrößert oder verkleinert werden. Somit ist es möglich, die Bewegung aus verschiedenen Perspektiven zu betrachten. Sie kann damit ausgiebig studiert und auf Fehler im Laufverhalten untersucht werden. Ein besonderer Fall ist die Platzierung eines Fußpunktes außerhalb der Reichweite des Beins. Derartige Fehler werden durch das Modell selbständig korrigiert. Um sie trotzdem zu bemerken, wird deshalb eine Fehlermeldung ausgegeben.

Zum Exportieren und Darstellen in anderen Anwendungen lassen sich einzelne Bilder, aber auch ganze Bewegungen als Animation in Dateien abspeichern. Dies ermöglicht das Betrachten der Bewegungen, auch ohne die entsprechende Software einzusetzen. Die Animationen können sogar in Internetseiten oder Präsentationen eingebettet werden.

7.1.3 Parallelisierung des Laufplaners

Die Laufplaner benötigen für die Lokale Suche und Simulated Annealing recht viel Rechenzeit. Ein Ergebnis liegt mitunter erst nach einigen Stunden vor. Für die Verwendung auf dem Roboter sind sie damit nicht geeignet. Allerdings müssen zur Bewertung der Verfahren Berechnungen durchgeführt werden. Um trotzdem in vertretbarer Zeit Ergebnisse zu erhalten, wurde eine Eigenschaft der Algorithmen ausgenutzt – die Parallelisierbarkeit. Durch Verwendung mehrerer Rechner wird die verfügbare Rechenleistung vervielfacht. Da die Algorithmen gut parallelisierbar sind, skaliert das Verfahren auch sehr gut mit der Anzahl der Prozessoren.

Die Verteilung auf mehrere Prozessoren erfolgte hier durch Verwendung eines OpenMosix-Clusters [34, 4]. Dabei handelt es sich um eine Erweiterung des normalen Linux-Kerns. Prozesse können für den Nutzer transparent auf andere Knoten des Clusters verschoben werden. Die Ein- und Ausgabe wird über das Netzwerk zum ursprünglichen Rechner geleitet. Somit ist die Verteilung auch für die Prozesse völlig transparent. Die Software muss nicht angepasst werden, um im Rechencluster zu laufen. Allerdings erfolgt die Parallelisierung nur auf Basis von Prozessen. Eine Anwendung, die nur aus einem Prozess besteht, erhält damit keine zusätzliche Rechenleistung. Sie muss also mehrere parallel laufende Prozesse beinhalten, um eine Lastverteilung zu ermöglichen.

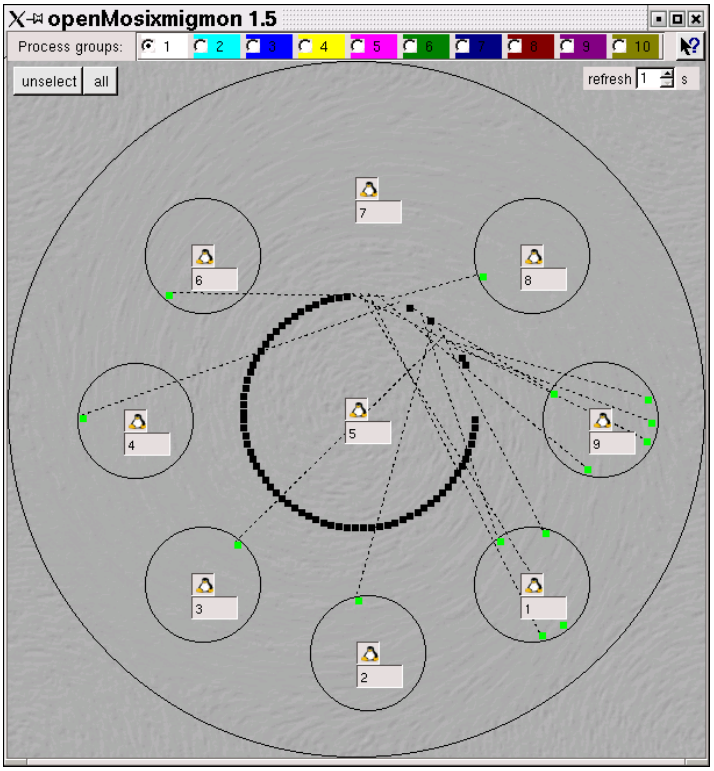


Abbildung 7.3: Migration von OpenMosix-Prozessen

Alle in dieser Arbeit betrachteten Algorithmen lassen sich durch mehrfaches Starten auf Prozesse aufteilen. Da es sich um randomisierte Verfahren handelt, müssen zu Beginn die Zu-

fallszahlengeneratoren unterschiedlich initialisiert werden. Ansonsten berechnen alle Prozesse identische Lösungen, was aber dem Ziel der Parallelisierung widersprechen würde.

Beim Start erzeugt das Programm über den Unix-Systemaufruf `fork()` eine geeignete Anzahl an identischen Prozessen. Diese werden, wie in Abbildung 7.3 zu sehen, automatisch auf die verfügbaren Rechner verteilt. Die Ergebnisse werden in individuelle Dateien geschrieben und können anschließend ausgewertet werden.

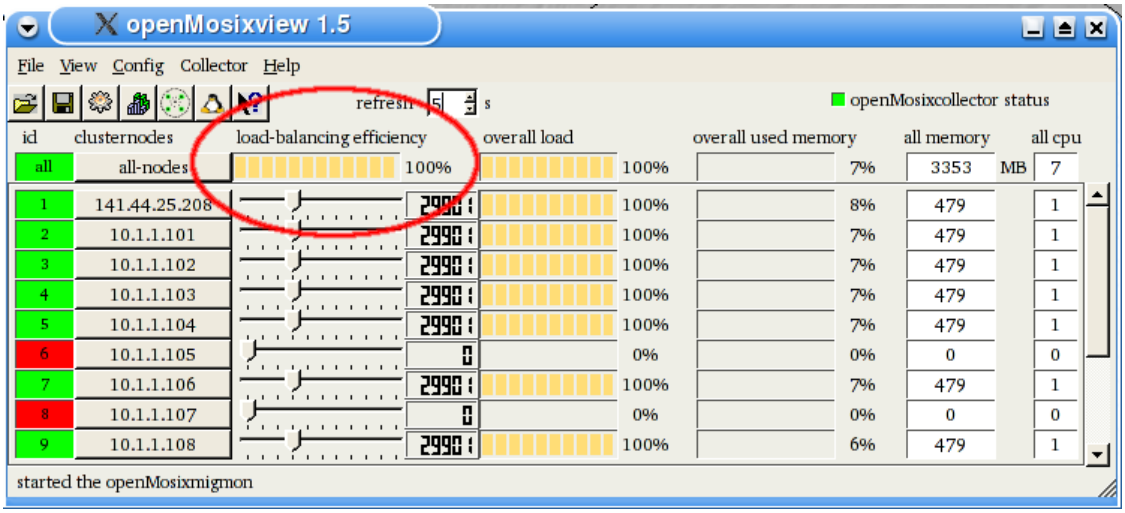


Abbildung 7.4: Lastverteilung im OpenMosix-Cluster

Wie in Abbildung 7.4 dargestellt, kann die Lastverteilung die Rechenleistung aller verfügbaren Rechner maximal ausnutzen. Auf diese Weise gelang es, auch für die langsamen Verfahren Ergebnisse zu berechnen. Außerdem zeigt sich hier, dass die geforderte gute Parallelisierbarkeit tatsächlich von den Algorithmen erbracht wird.

7.2 Test einiger Szenarien

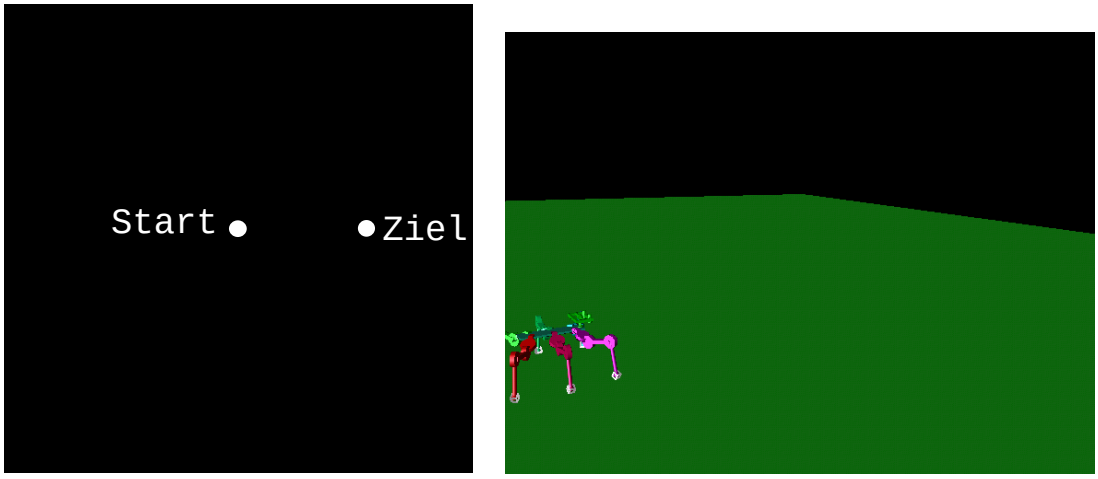
Um eine sinnvolle Aussage über die Qualität des auf Random Sampling basierenden Laufplaners treffen zu können, ist es notwendig, mehr als ein Szenario zu betrachtet. Das bisher verwendete Laufen auf flachem Grund ist einfach zu lösen und eigentlich nicht der Fall, der interessant ist. Der Laufplaner soll gerade für unebenes Gelände geeignet sein. Deshalb werden in diesem Abschnitt einige zusätzliche Szenarien untersucht.

7.2.1 Laufen auf flachem Grund

Das einfachste Problem ist das bereits bei den Algorithmen betrachtete Laufen auf flachem Grund. Hier bieten die eingangs beschriebenen Laufmuster bereits eine gute Lösung. Der Laufplaner muss auch in der Lage sein, diesen Fall abzudecken. Deshalb wird das Szenario „flaches Gelände" hier noch einmal kurz betrachtet.

Beschreibung des Weltmodells

Im Weltmodell haben alle Höhen den gleichen Wert. Damit ergibt sich ein flaches Gelände. Das Rasterbild besteht, wie in Abbildung 7.5 zu sehen, nur aus einem Farbwert.



Rasterbild *3D-Ansicht*

Abbildung 7.5: Szenario „flaches Gelände"

Ergebnis der Berechnung

Der Laufplaner findet sehr schnell eine Lösung für dieses Szenario. Die genauen Daten wurden bereits in Abschnitt 5.2.2 dargestellt. Dank der Visualisierung kann hier auch der optische Eindruck beschrieben werden. Das Laufen erfolgt auf direktem Weg zum Ziel und wirkt insgesamt flüssig. Es werden kaum unnötig erscheinende Beinbewegungen vorgenommen.

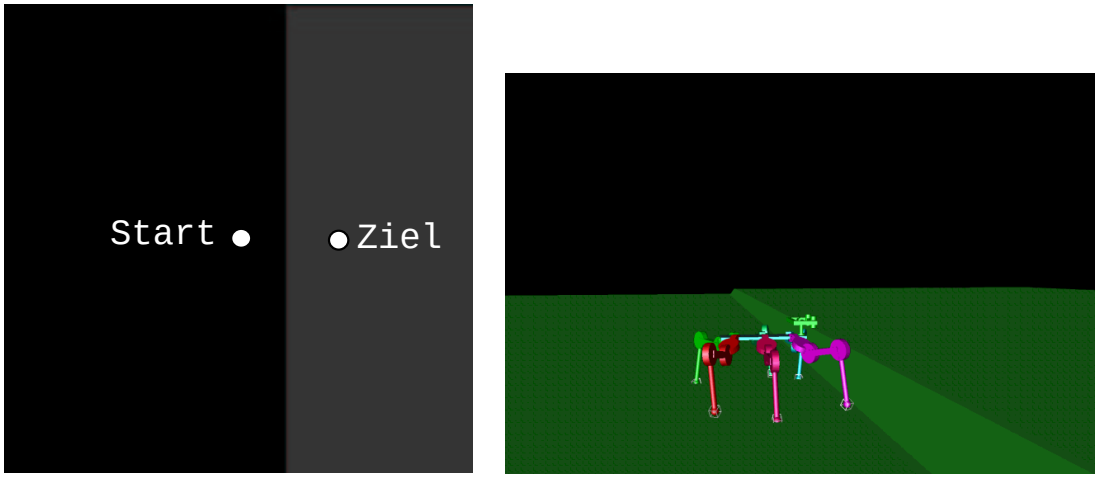
7.2.2 Laufen über eine Stufe

Das Laufen über eine Stufe entspricht dem Überwinden kleiner Hindernisse. Steuerungen basierend auf Laufmustern sind dazu in der Lage, wenn sie zusätzlich den Elevator-Reflex zum Anheben des Beins verwenden. Das Hindernis wird aber erst bei der Kollision mit dem Bein erkannt. Der Reflex bewegt dann den Fuß durch Tasten über das Hindernis. Der Laufplaner soll dagegen in der Lage sein, dem Hindernis vorausschauend auszuweichen. Der Fuß wird von Anfang an auf die entsprechende Höhe gebracht, wodurch das Hindernis ohne weitere Störungen überwunden wird. Ein derartiges Verhalten ist bereits durch Kenntnis der nötigen Höhe möglich. Diese lässt sich aus dem Weltmodell gewinnen.

Als besondere Schwierigkeit wurde hier die Höhe der Stufe so gewählt, dass der Roboter sie gerade noch überwinden kann. Dazu muss sich der Körper in einer geeigneten Entfernung befinden, um die Füße auf die Stufe setzen zu können.

Beschreibung des Weltmodells

Das Gelände im Weltmodell besteht aus zwei in der Höhe versetzten Ebenen. An der Kante entsteht die Stufe. Das Rasterbild und eine entsprechende 3D-Grafik sind in Abbildung 7.6 zu sehen. Der Roboter muss von der unteren auf die obere Ebene gelangen. Da sich die Stufe über den gesamten Bereich erstreckt, muss sie unweigerlich überwunden werden.



Rasterbild *3D-Ansicht*

Abbildung 7.6: Szenario „Stufe"

Ergebnis der Berechnung

Der Laufplaner findet nicht so schnell eine Lösung, wie beim einfachen flachen Gelände. Aufgrund der erhöhten Schwierigkeit war dies auch nicht anzunehmen. Die gefundenen Lösungen entsprechen den Erwartungen. Der Roboter läuft bis zur Stufe. Anschließend werden zuerst die vorderen, gefolgt von den hinteren Beine heraufgesetzt. Danach wird die Bewegung normal fortgeführt.

Anteil an allen gültigen Bewegungen	*Anzahl der Iterationen*				
	1	10	100	1000	10 000
100%	0,012 6	0,119	0,719	≈ 1	≈ 1
50%	0,0063	0.061 2	0,468	0,998	≈ 1
10%	0,001 26	0,012 5	0,118	0,717	≈ 1
1%	0,000 126	0,001 26	0,0125	0,118	0,716
0,1%	≈ 0	0,000 126	0,001 26	0,012 5	0,118

Tabelle 7.1: Wahrscheinlichkeit guter Bewegungen (Stufe)

Unter 10 000 berechneten Bewegungen waren 126 gültige. Die Verteilung der dazugehörigen Bewertung ist im Histogramm in Abbildung 7.7 dargestellt. Damit ergeben sich die Wahrscheinlichkeiten für gute Lösungen, wie in Tabelle 7.1. Bei 1000 erzeugten Bewegungen ist

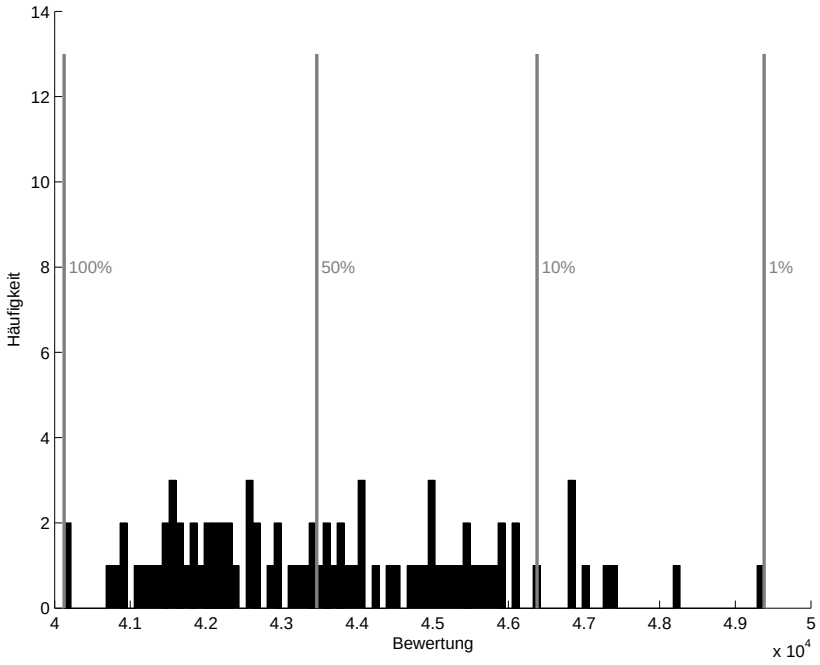


Abbildung 7.7: Histogramm „Stufe"

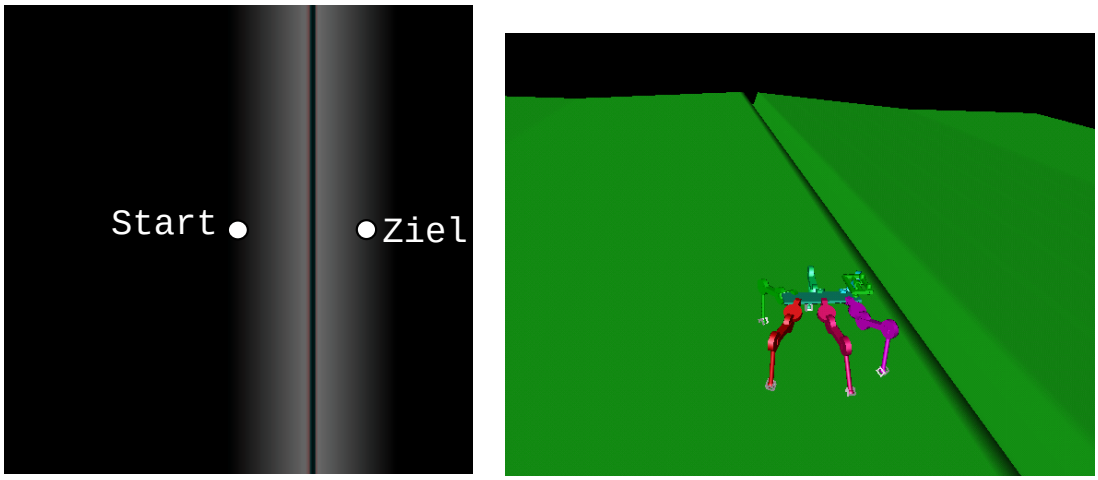
ziemlich sicher, dass eine gültige Lösung (100%-Zeile) gefunden wird. Dabei wird mit einer Wahrscheinlichkeit von 73,8% sogar eine unter den besten 10% aller gültigen Lösungen gefunden. Durch mehr Iterationen steigt die Wahrscheinlichkeit dafür. Der Laufplaner kann also dieses Szenario bewältigen und gute Ergebnisse liefern.

7.2.3 Laufen über einen Graben

Szenarien, die mit normalen Laufmustern nicht mehr bewältigt werden können, beinhalten das Überschreiten größerer Hindernisse. Ein in der Literatur (beispielsweise in [8]) oft verwendeter Repräsentant dieser Gruppe ist das Überwinden eines tiefen Grabens. Die Beine finden innerhalb des Grabens keinen Halt. Dieser Bereich darf also nicht betreten werden. Ist der verbotene Bereich klein genug, kann er durch Anpassung der Schrittweite einfach überwunden werden. Ist er aber fast genauso breit wie die Reichweite eines Beins, gelingt dies nicht mehr so einfach. Die gesamte Laufbewegung muss angepasst werden.

Beschreibung des Weltmodells

Das Gelände besteht aus zwei Rampen, zwischen denen sich der Graben befindet. Der Roboter muss die erste Rampe hinauf laufen, den Graben überwinden und anschließend die zweite Rampe wieder hinab laufen. Das in Abbildung 7.8 dargestellte Rasterbild besitzt dementsprechend zwei Hell-Dunkel-Verläufe, die den Höhenverlauf repräsentieren. Dazwischen ist ein dunkler Streifen erkennbar, der dem Graben entspricht.



Rasterbild *3D-Ansicht*

Abbildung 7.8: Szenario „Graben"

Ergebnis der Berechnung

Dem Bewegungsplaner gelingt es, eine geeignete Bewegung zum Überwinden des Grabens zu finden. Dies wäre mit anderen Ansätzen wie Laufmustern nicht möglich gewesen. Auch reaktive Ansätze könnten die hier gefundenen Ergebnisse nicht erzielen. Man erkennt das bei Betrachtung der gefundenen Lösungen. Der Körper muss erst dicht an den Graben heran bewegt werden. Dann werden die Beine nacheinander übergesetzt, beginnend mit den vorderen. Solch ein Laufverhalten ist in den bekannten Ansätzen nicht vorgesehen. Man könnte zwar die Bewegung fest vorgeben und falls nötig zum Einsatz bringen. Solch ein Umschalten der Laufbewegung erfordert allerdings ein Erkennen der Situation. Das Problem wird dann auf das Erkennen des Graben-Falls verschoben. Einen allgemeinen Lösungsansatz stellt diese Methode darüber hinaus nicht dar, weil nur solche Fälle behandelt werden können, die bekannt sind. An diesem Szenario zeigt sich deutlich der Vorteil eines Laufplaners. Es konnte ein Hindernis bewältigt werden, an dem nichtplanende Laufsteuerungen gescheitert wären.

Anteil an allen gültigen Bewegungen	*Anzahl der Iterationen*				
	1	10	100	1000	10 000
100%	0,013 4	0,126	0,741	≈ 1	≈ 1
50%	0,0067	0.065	0,489	0,999	≈ 1
10%	0,001 34	0,013 3	0,125	0,738	≈ 1
1%	0,000 134	0,001 33	0,0133	0,125	0,738
0,1%	≈ 0	0,000 133	0,001 33	0,013 3	0,125

Tabelle 7.2: Wahrscheinlichkeit guter Bewegungen (Graben)

Von den 10 000 berechneten Bewegungen waren 134 gültig. Damit liegt nach 100 bis 1000 Berechnungen mit sehr hoher Wahrscheinlichkeit eine gültige Lösung vor. Aus Tabelle 7.2 wird ersichtlich, dass für eine gute Lösung eine höhere Anzahl an Iterationen nötig ist. Für dieses Szenario ist aber entscheidender, dass überhaupt eine gültige Bewegung gefunden

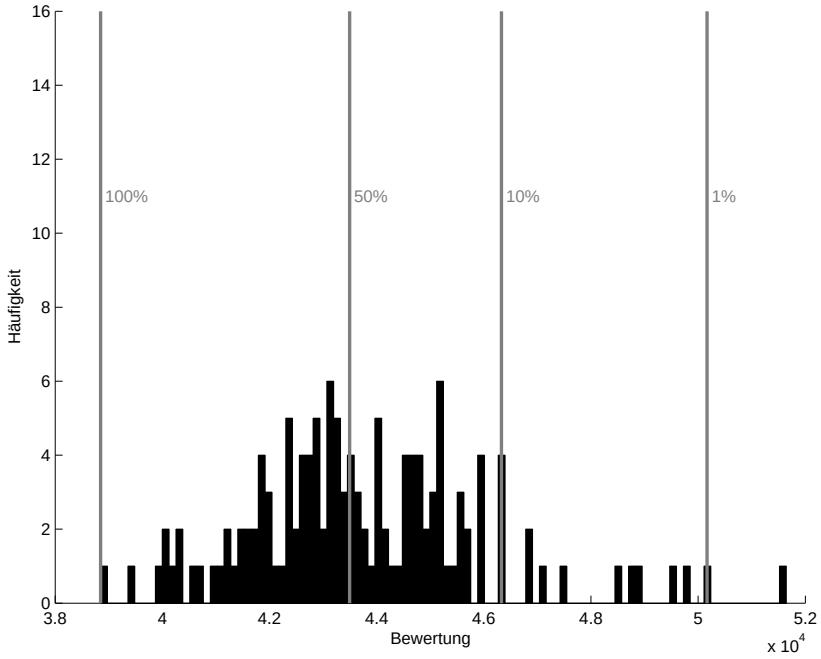


Abbildung 7.9: Histogramm „Graben"

wird. Der optische Eindruck der besten gefundenen Lösung ist dennoch recht gut. Die Beinbewegungen sind fast immer angemessen. Der Graben wird relativ zügig überquert.

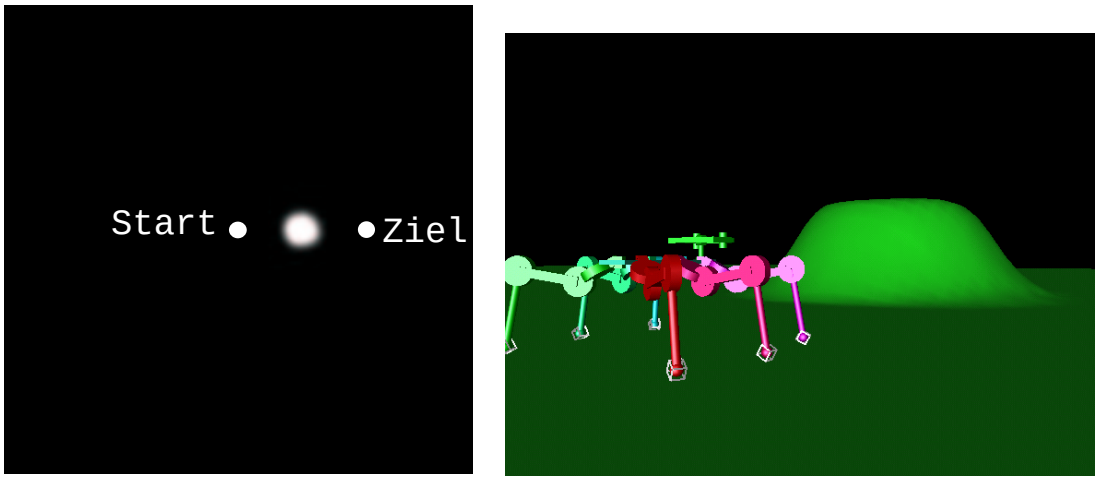
7.2.4 Laufen um ein unüberwindbares Hindernis

Viele Laufsteuerungen ermöglichen die Bewegung des Roboters auf einer vorgegebenen Bahn. Dabei wird die Strecke von der Anwendung genau vorgegeben. Die Aufgabe der Steuerung beschränkt sich auf das Platzieren der Fußpunkte. Dies hat jedoch einen entscheidenden Nachteil: Existiert keine zulässige Bewegung bei Einhaltung der gegebenen Bahn, wird keine Lösung ermittelt. Eventuell könnte aber durch eine leichte Abweichung von der Vorgabe eine Bewegung gefunden werden.

Dieser Fall wird im Szenario „unüberwindbares Hindernis" betrachtet. Der Laufplaner soll eine Bewegung zwischen zwei Punkten ermitteln. Auf der geraden Strecke befindet sich aber ein Hindernis, das nicht überwunden werden kann. Eine Bewegung um das Hindernis herum ist dagegen möglich. Sie kann nur nicht auf einer Geraden zum Ziel verlaufen.

Beschreibung des Weltmodells

Das Gelände in diesem Szenario besteht fast nur aus flachem Untergrund. Auf der geraden Strecke zwischen Start- und Zielpunkt befindet sich allerdings das Hindernis. Dabei handelt es sich um einen Hügel, der so steile und hohe Seiten hat, dass er nicht überwunden werden kann. Im Rasterbild (Abbildung 7.10) wird er durch einen hellen Fleck auf sonst dunklem Grund dargestellt.



Rasterbild *3D-Ansicht*

Abbildung 7.10: Szenario „unüberwindbares Hindernis"

Ergebnis der Berechnung

In den berechneten Bewegungen läuft der Roboter wie erwartet um das Hindernis herum. Es braucht etliche Versuche, bis eine gültige Lösung gefunden wird. Auch hier gilt, dass das Finden einer Bewegung allein schon ein Fortschritt ist. Es zeigt sich daran, wie die vorgegebene Strecke flexibel interpretiert wird. Würde das Ziel direkt angesteuert, wäre kein Erreichen möglich. Normalerweise sollten große Hindernisse von der Anwendung beachtet und vermieden werden. Sonst könnten auch Zielpunkte vorgegeben werden, die überhaupt nicht erreichbar sind. In solchen Fällen findet der Laufplaner auch keine Bewegung, was ein Fortsetzen des Laufens verhindert. Ist die Zielvorgabe kritisch, aber noch möglich, wird bei diesem Verfahren trotzdem eine gültige Lösung gefunden. Der höhere Aufwand und die Suboptimalität der Lösung sind gerechtfertigt, wenn zumindest eine gefunden wird.

Anteil an allen gültigen Bewegungen	*Anzahl der Iterationen*				
	1	10	100	1000	10 000
100%	0,005 4	0,052 7	0,418	0,996	≈ 1
50%	0,002 7	0.026 7	0,237	0,933	≈ 1
10%	0,000 54	0,005 39	0,052 6	0,417	0,995
1%	≈ 0	0,000 539	0,005 39	0,052 6	0,417
0,1%	≈ 0	≈ 0	0,000 539	0,005 39	0,052 6

Tabelle 7.3: Wahrscheinlichkeit guter Bewegungen (Hindernis)

Von 10 000 erzeugten Bewegungen waren lediglich 54 gültig. Nach Tabelle 7.3 müssen etwa 1000 Möglichkeiten untersucht werden, bis ein positives Resultat erzielt wird. Dementsprechend ist auch nicht mit einer optimalen Bewegung zu rechnen. Es treten teilweise überflüssige Beinbewegungen auf. Der Roboterkörper bewegt sich stellenweise sehr ruckartig.

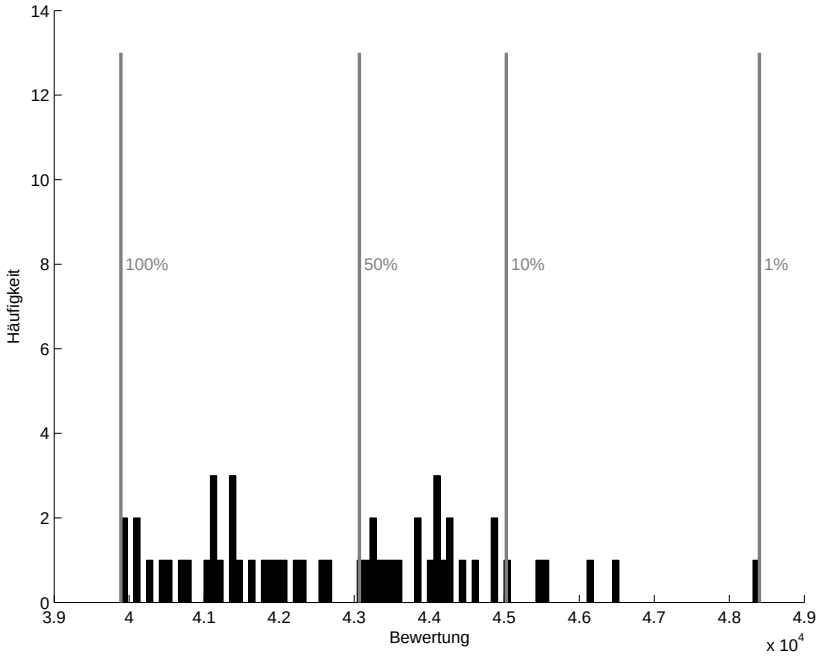


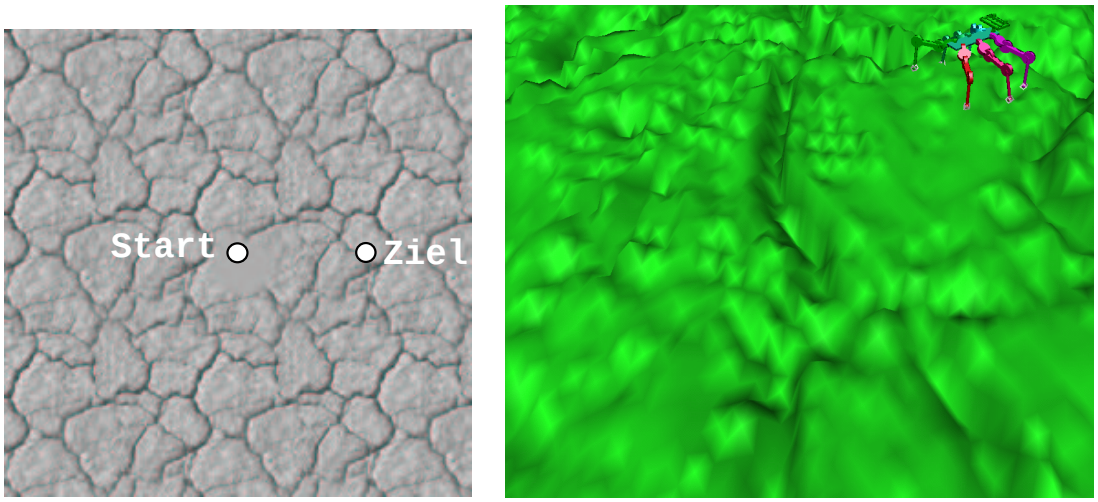
Abbildung 7.11: Histogramm „unüberwindbares Hindernis"

7.2.5 Laufen durch zerklüftetes Gelände

Die bisher betrachteten Szenarien stellten besondere Fälle dar, die jeweils eine spezielle Eigenschaft des Laufplaners vorführen. Sie entsprechen jedoch nicht einer natürlichen Landschaft. An dieser Stelle soll deshalb ein realistischeres Beispiel für unebenes Gelände betrachtet werden.

Beschreibung des Weltmodells

Die Landschaft weist kaum ebenes Gelände auf. Viele Bereiche besitzen einen starken Anstieg. Sie dürfen nicht betreten werden, weil sie keinen hinreichenden Halt bieten. Zusätzlich gibt es einige tiefere Furchen. Der Untergrund ist dort nicht erreichbar, weshalb auch kein Fuß aufgesetzt werden darf. In Abbildung 7.12 ist das dazugehörige Rasterbild dargestellt.



Rasterbild *3D-Ansicht*

Abbildung 7.12: Szenario „zerklüftetes Gelände"

Ergebnis der Berechnung

Die vom Laufplaner berechnete Bewegung weist kein fehlerhaftes Verhalten auf. Die Fußpunkte werden stets auf geeignetem Untergrund platziert. Da die Landschaft hierfür sehr wenige Bereiche bereitstellt, erscheint die Wahl der Positionen recht ungeordnet. Dies ist aber den Bedingungen des Szenarios geschuldet. Ein tieferer Graben, der den Weg durchquert, wird überwunden.

Unter den 10 000 berechneten Bewegungen waren 122 gültige. Die Wahrscheinlichkeiten einer guten Lösung ergeben sich damit wie in Tabelle 7.4. Bei 100 Berechnungen ist die Wahrscheinlichkeit einer zulässigen Lösung recht hoch. Ab 1000 ist sie eigentlich sicher. Für gute Lösungen müssen noch mehr Berechnungen durchgeführt werden. Die Anzahl ist jedoch nicht so hoch wie bei den anderen Szenarien.

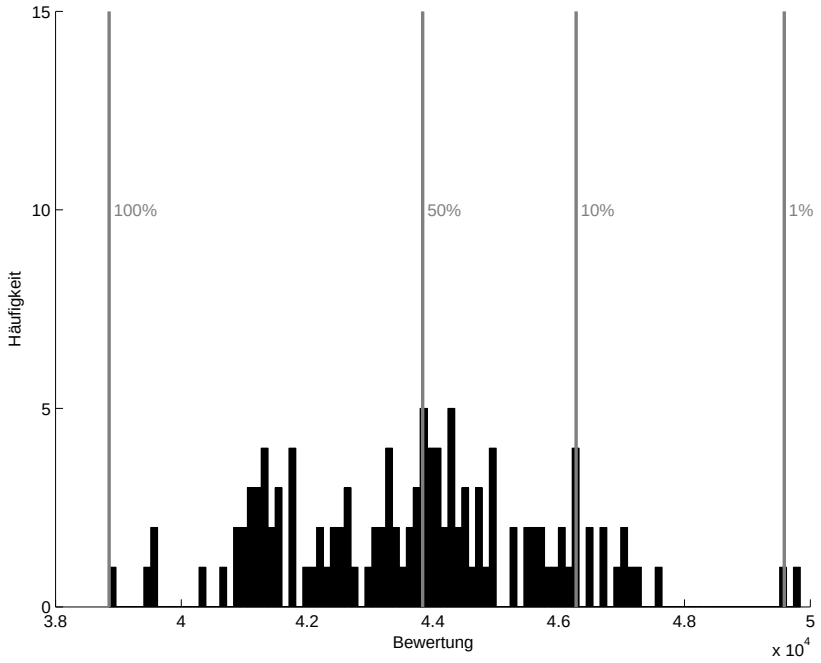


Abbildung 7.13: Histogramm „zerklüftetes Gelände"

Anteil an allen gültigen Bewegungen	*Anzahl der Iterationen*				
	1	10	100	1000	10 000
100%	0,012 2	0,115	0,707	≈ 1	≈ 1
50%	0,006 1	0.059 4	0,458	0,998	≈ 1
10%	0,001 22	0,012 1	0,115	0,705	≈ 1
1%	0,000 122	0,001 22	0,0121	0,115	0,705
0,1%	≈ 0	0,000 122	0,001 21	0,012 1	0,115

Tabelle 7.4: Wahrscheinlichkeit guter Bewegungen (zerklüftetes Gelände)

Es zeigt sich hier, dass der Laufplaner nicht nur für einige konstruierte Fälle geeignet ist. Es kann beliebiges Gelände behandelt werden, wie es die Aufgabenstellung fordert.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die Arbeit beschreibt die Entwicklung eines verteilten Laufplaners basierend auf dem heuristischen Optimierungsverfahren Random Sampling. Ziel war die Planung von Bewegungen für den Laufroboter LAURON III. Diese sollen zur Laufzeit auf Basis eines Weltmodells berechnet werden. Bewegungen mit einer hohen Güte sind zu bevorzugen. Da ein exaktes Lösungsverfahren aufgrund der Komplexität des Problems ausschied, ist die Verwendung einer geeigneten Heuristik nötig. Zur Verbesserung der Ergebnisse war die Verteilung des Algorithmus auf mehrere Prozessoren zu erwägen.

Aus der Forderung nach guten Lösungen stellt sich das Laufplanungsproblem als Optimierungsproblem dar. Dieses galt es zu lösen. Als ersten Schritt wurde das Problem daher formal definiert. Es war eine Beschreibung der Eingabe, des Lösungsraums und der Bewertungsfunktion nötig. Wird das resultierende Optimierungsproblem exakt gelöst, so liegt eine optimale Bewegung zum Ziel vor. Wird es hinreichend gut gelöst, so erhält man zumindest gute Bewegungen. Dies entspräche den Anforderungen.

Da kein exaktes Lösungsverfahren bekannt ist, wurde eine geeignete Heuristik gesucht. Es wurde eine Auswahl von allgemeinen Heuristiken auf ihre Eignung untersucht. Dies geschah zunächst theoretisch anhand der festgelegten Anforderungen. Als mögliche Kandidaten stellten sich Random Sampling, Lokale Suche und Simulated Annealing heraus. Sie wurden im Weiteren praktisch getestet.

Der Random Sampling Algorithmus ist sehr einfach. Er benötigt jedoch eine Funktion, die zufällige gültige Lösungen erzeugt. Es wurde detailliert beschrieben, wie dies für die Laufbewegungen des Roboters möglich ist. Aufbauend darauf wurde der Algorithmus implementiert und getestet. Er stellte sich als geeignet heraus.

Im Anschluss wurden noch die beiden Verfahren Lokale Suche und Simulated Annealing untersucht. Sie benötigen die Definition einer Nachbarschaft auf der Menge aller Bewegungen. Diese wurde beschrieben, was die Entwicklung der zugehörigen Algorithmen ermöglichte. Die Algorithmen wurden implementiert und getestet. Es zeigte sich jedoch, dass beide Ansätze aufgrund unverhältnismäßig langer Rechenzeiten für das vorliegende Problem nicht geeignet sind. Deshalb kam nur der Algorithmus basierend auf Random Sampling für die Laufplanung in Frage.

Der Laufplaner sollte anhand unterschiedlicher Szenarien getestet werden. Da ein Einsatz auf dem realen Roboter noch nicht möglich war, wurde eine Simulationsumgebung entwickelt. Die Bewegungen konnten hiermit berechnet und dargestellt werden. Für alle getesteten Szenarien ließ sich eine korrekte Lösung ermitteln. Die Güte der Bewegung hängt dabei

stark von der Schwierigkeit des Geländes ab. In einer flachen Umgebung sind die Ergebnisse gut. Bei komplizierten Hindernissen wird eine besonders gute Bewegung dagegen nur mit viel Rechenzeit gefunden. Hier ist das Finden einer gültigen Laufbewegung allerdings schon ausreichend.

Es wurde in dieser Arbeit gezeigt, dass die Laufplanung basierend auf einem heuristischen Optimierungsverfahren möglich ist. Damit kann eine geeignete Bewegung für Laufroboter in einer vorgegebenen Zeit berechnet werden. Sensorinformationen können somit verwendet werden, um eine Laufbewegung für ein Gelände zu berechnen, das mit anderen Verfahren wie reaktiven Steuerungen nicht zu bewältigen wäre.

8.2 Ausblick

Nachdem der Laufplaner in der Simulation getestet wurde, sollte er auf dem realen System zum Einsatz kommen. Bisher erfolgte die Laufplanung unter idealen Bedingungen. Der reale Roboter unterliegt aber Störungen wie Rauschen in den Kamerabildern. Daraus resultiert ein ungenaues Weltmodell, was sich auf die Exaktheit der Planung niederschlägt. Zu untersuchen wäre hier die Toleranz der Laufplanung.

Ein weiterer Aspekt wäre die Anwendung des Laufplaners auf andere Roboter. Grundsätzlich ist der Algorithmus sehr allgemein gehalten. Lediglich das verwendete kinematische Modell (Berechnung der Beinreichweite) und das Beinreglermodell (Berechnung der minimalen Dauer einer Bewegung) sind spezifisch für den LAURON III. Durch Austauschen dieser Komponenten könnte auch eine Laufplanung für andere Roboter möglich sein. Selbst die Anzahl der Beine sollte sich variieren lassen. Dies erfordert nur eine Aktualisierung der Tabelle der Stützzustände (Tabelle 5.1). Hierdurch ließe eine Aussage über die allgemeine Anwendbarkeit des vorgestellten Verfahrens treffen.

Index

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] AARTS, Emile H. L. ; KORST, Jan:
Simulated Annealing and Boltzmann Mashines.
Wiley - Interscience series in discrete mathmatics, 1989

[2] AUSIELLO, G. ; CRESCENZI, P. ; GAMBOSI, G. ; KANN, V. ; MARCHETTI-SPACCAMELA, A. ; PROTASI, M.:
Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.
Springer Verlag, 2000

[3] BADE, Richard:
Modifikation einer geeigneten Stereobildverarbeitungsmethode für die Anwendung in einer Echtzeitumgebung, Universität Magdeburg, Diplomarbeit, Juni 2003

[4] BARAK, A. ; GUDAY, S. ; WHEELER, R.:
The MOSIX Distributed Operating System, Load Balancing for UNIX.
Springer-Verlag, 1993

[5] BERGER, Christian:
Auslegung und Konstruktion einer sechsbeinigen Gehmaschine, Universität Karlsruhe, Institut für technische Mechanik, Studienarbeit, 1995

[6] BERNS, K.:
Die Anwendung rekurrenter neuronaler Netze für die Beinsteuerung der Laufmaschine LAURON.
In: *KI/94 Workshops, Saarbrücken*, 1994

[7] CANNY, John F.:
The complexity of robot motion planning.
The MIT Press, London, 1988

[8] CHEN, Chun-Hung ; KUMAR, Vijar ; LUO, Yuh-Chyun:
Motion Planning of Walking Robots in Environments with Uncertainty.
In: *Journal for Robotic Systems* Bd. 16.
1999, S. 527–545

[9] CRUSE, Holk:
The Control of Body Position in the Stick Insect (Carausius morosus) when Walking over Uneven Surfaces.
In: *Biological Cybernetics Vol. 24*
(1976), S. 25–33

[10] CRUSE, Holk [u. a.]:
Walking: A complex Behaviour Controlled by Simple Networks.
In: *Adaptive Behaviour*
3 (1995), Nr. 4, S. 385–418

[11] DEAN, Jeffrey:
A model of leg coordination in the stick insect Carausius morosus.
In: *Biological Cybernetics*
64 (1991), S. 403–411

[12] DEAN, T. L.:
Intrancibility and Time-Dependent Planning.
In: *Workshop on Reasoning about Actions and Plans*, 1987

[13] EARON, Ernest J. P. ; BARFOOT, Tim D. ; D'ELEUTERIO, Gabriele M. T.:
From Sea to the Sidewalk: The Evolution of Hexapod Walking Gaits by a Genetic Algorithm.
In: *International Conference on Evolvable Systems*, 2000

[14] ELDERSHAW, Craig:
Heuristic algorithms for motion planing, University of Oxford, D.Phil. Thesis, 2001

[15] FERRELL, Cynthia:
A comparison of three insect-inspired locomotion controllers.
In: *Robotics and Autonomous Systems*
16 (1995), S. 135–159

[16] Forschungszentrum Informatik:
Walking Robot Catalog.
Dezember 2003. –
`http://www.fzi.de/ids/Galerie.html`

[17] GASSMANN, Bernd:
Erweiterung einer modularen Laufmaschinensteuerung für unebenes Gelände, Forschungszentrum Informatik an der Universität Karlsruhe, Diplomarbeit, September 2000

[18] GUDDAT, Martin:
Fortschritt-Betrichte VDI. Bd. 995: *Autonome, adaptive Bewegungskoordination von Gehmaschinen in komplexer Umgebung*.
VDI Verlag GmbH, 2003

[19] HOLLAND, John:
Adaption in Natural and Artificial Systems.
The University of Michigan Press, 1975

[20] HROMKOVIČ, Juraj:
Algorithmics for Hard Problems: introduction to combinatorial optimization, ranomization, approximation and heuristics.
Springer Verlag, 2001

[21] IHME, Thomas:
Steuerung sechsbeiniger Laufroboter unter dem Aspekt techischer Anwendungen, Universität Magdeburg, Dissertation, März 2002

[22] IKEUCHI, Katsushi:

Determining a Depth Map Using Dual Photometric Stereo.
In: *The International Journal of Robotics Research*
6 (1987), Nr. 1, S. 15–31

[23] ILG, W. ; BERNS, K.:
Eine adaptive Steuerungsarchitektur zum Erlernen höherer Verhaltensweisen für autonome mobile Systeme.
In: *Informatik aktuell*
(1995)

[24] Infineon Technologies AG:
C167CR User's Manual, 16-Bit Single-Chip Microcontroller.
2000

[25] JIMÉNEZ, Maria A. ; SANTO, P. G.:
Terrain-Adaptive Gait for Walking Machines.
In: *The International Journal of Robotics Research*
16 (1997), June, Nr. 3, S. 320–339

[26] KALISIAK, Maciej ; PANNE, Michiel von d.:
A Graps-based Motion Planning Algorithm for Character Animation
/ Department of Computer Science, University of Toronto.
1998.
– Forschungsbericht

[27] KUBOW, T. M. ; FULL, R. J.:
The role of the mechanical system in control: a hypothesis of self-stabilization in hexapedal runners.
In: *Royal Society London*
354 (1999), S. 849–861

[28] LAARHOVEN, Peter J. M. ; AARTS, Emile H. L.:
Simulated Annealing: Theory and Applications.
Kluwer Academic Publishers, 1987

[29] LEE, D. T.:
Computational geometry.
In: *ACM Computing Surveys*
28 (1996), Nr. 1, S. 27–31

[30] LEWIS, M. A. ; FAGG, Andrew H. ; BEKEY, George A.:
Genetic Algorithms for Gait Synthesis in a Hexapod Robot.
In: *World Sientific*
(1994)

[31] LOZANO-PÉREZ, Thomás ; WESLEY, M. A.:
An algorithm for planning collision free paths among polyhedral obstacles.
In: *Communication of the ACM*
22 (1979), Oktober, Nr. 10, S. 21–29

[32] NISHII, Jun:
Legged insects select the optimal locomotor pattern based on enegetic costs.
In: *Biologic Cybernetics*
83 (2000), S. 435–442

[33] OpenInventor:
Homepage.
Dezember 2003. –
`http://oss.sgi.com/projects/inventor/`

[34] The OpenMosix Project:
Homepage.
Dezember 2003. –
`http://openmosix.sourceforge.net/`

[35] PAL, Prabir K. ; KAR, Dayal C.:
Gait Optimization though Search.
In: *The International Journal of Robotic Research*
19 (2000), April, Nr. 4, S. 394–408

[36] PARKER, Garry B. ; BRAUN, David W. ; CYLIAX, Ingo:
Learning Gaits for the Stiquito
/ Department of Computer Science, Indiana University.
1997.
– Forschungsbericht

[37] PARKER, Gary B. ; BRAUN, David W. ; CYLIAX, Ingo.
Evolving Hexapod Gaits using a cyclic Genetic Algorithm.
2000

[38] PFEIFFER, Friedrich ; ELTZE, Jürgen ; WEIDEMANN, Hans-Jürgen:
Six-legged technical walking considering biological principles.
In: *Robotics and Autonomous Systems*
14 (1995), S. 223–232

[39] PLUTOWSKY, Axel:
Konstruktionsoptimierung einer sechsbeinigen Gehmaschine, Universität Karlsruhe, Institut für technische Mechanik, Studienarbeit, 1995

[40] RECHENBERG, I.:
Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution.
Stuttgart: Fromman-Holzboog Verlag, 1973

[41] Robert Bosch GmbH, Stuttgart:
CAN Specification Version 2.0

[42] SONG, Shin-Min ; WALDRON, Kenneth J.:
An Anaytical Approach for Gait Study and Its Applications on Wave Gaits.
In: *The International Journal Of Robotics Research*
6 (1987), Summer, Nr. 2, S. 60–71

[43] VOSS, Stefan (Hrsg.) ; MARTELLO, Silvano (Hrsg.) ; OSMAN, Ibrahim H. (Hrsg.) ; ROUCAIROL, Catherine (Hrsg.):
Advances and Trends in Local Search Paradigms for Optimization.
Kluwer Academic Publishers, 1999

[44] WETTERGREEN, David:
Robot walking in natural terrain, Carnegie Mellon University, Diss., 1995

[45] The XML C parser and toolkit of Gnome:
Homepage.
Dezember 2003. –
`http://www.xmlsoft.org`

SELBSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, die vorliegende Arbeit selbst verfasst, Zitate gekennzeichnet und keine anderen als die offen gelegten Quellen und Hilfsmittel benutzt zu haben.

André Herms
Magdeburg, den 23. Januar 2004