hochschule mannheim

test First line
second line title

Daniel Koch

Bachelor-Thesis

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

22.07.2020

Betreuer

Prof. Dr. Thomas Ihme, Hochschule Mannheim

Koch, Daniel:
TEST / Daniel Koch. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2020. 21 Seiten.

Koch, Daniel:
TEST / Daniel Koch. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2020. 21 pages.

Erklärung

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

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

Mannheim, 22.07.2020

Daniel Koch

Abstract

TEST

TEST.

TEST

TEST.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

1.2 Ziel der Arbeit

1.3 Aufbau der Arbeit

Kapitel 2

Grundlagen

2.1 Aufbau des Laufroboters

2.2 Robotik

2.2.1 Koordinatensysteme

2.2.2 Direkte Kinematik

2.2.3 Inverse Kinematik

2.2.4 Laufplanung

statische Laufalgorithmen, reaktive, planende Laufalgorithmen like RandomSampling

2.3 Frameworks

2.3.1 Robot Operating System

2.3.2 Gazebo

2.3.3 MeshLab

Kapitel 3

Random Sampling

Dieses Kapitel geht auf die konzeptionellen Aspekte des Random Samplings ein. Zunächst betrachtet die Arbeit, weshalb der Algorithmus die beste Wahl für den Laufplaner ist. Darauf folgt die Darstellung der Funktionsweise des Random Samplings, welches große Mengen an zufälligen und gültigen Lösungen generiert. Dabei sind zwei Fragen von großer Bedeutung. Zum einen ist das: Wie können zufällige gültige Lösungen generiert werden. Wenn diese Lösungen vorliegen ist die nächste wichtige Frage: Wie können diese Lösungen bewertet werden?

Das Kapitel basiert auf der Arbeit von André Herms [1], der verschiedene planende Verfahren vorstellt und mit geeigneten Kriterien bewertet. Da seine Arbeit zusammenfassend davon ausgeht, dass Random Sampling die nach seinen Kriterien beste Wahl darstellt, nutzt auch diese Arbeit den Algorithmus.

3.1 Kriterien zur Auswahl des Algorithmus

Herms stellt in seiner Arbeit neben dem Random Sampling die folgenden Algorithmen vor:

- *Greedy Verfahren:* Es werden nacheinander Teillösungen durch ein lokales Kriterium generiert. Teillösungen werden nicht mehr verworfen. Daher kommt auch der Begriff „greedy“, da der Algorithmus gierig ist und somit Lösungen nicht wieder verwirft.

- *Branch and Bound:* Das Verfahren liefert durch einen modifzierten Backtracking-Ansatz immer die exakte Lösung. Anders als beim Backtracking werden die Suchbäume gestutzt, so dass die Laufzeit dadurch verkürzt wird.
- *Lokale Suche:* Das Verfahren ist eine modifzierte Variante des Random Samplings. Neben der zufälligen Generierung von Lösungen wird auch iterativ in der Nachbarschaft nach einem besseren Punkt geschaut.
- *Tabu-Suche:* Das Verfahren ist eine Erweiterung der lokalen Suche. Der Nachbar muss nicht zwingend besser bewertet sein, um als Nachfolgepunkt akzeptiert zu werden. Damit bleibt der Algorithmus nicht in lokalen Maxima hängen.
- *Simulated Annealing:* Das Verfahren ist eine erneute Verbesserung der lokalen Suche und der Tabu-Suche, das sich hinsichtlich des Optimierungsproblems an dem physikalischen Prozess des langsamen Abkühlen fester Stoffe orientiert.
- *Genetische Algorithmen:* Das Verfahren nutzt die Mechanismen der natürlichen Evolution und wendet diese auf das Problem der Laufplanung an.

Die genannten Algorithmen vergleicht Herms nun unter Berücksichtigung der folgenden Kriterien:

- Parallelisierbarkeit
- Speicherbedarf
- Anytime-Fähigkeit
- Allgemeine Anwendbarkeit

Dabei stellt sich heraus, dass nur die Algorithmen *Random Sampling*, *Lokale Suche* und *Simulated Annealing* alle Kriterien mindestens erfüllen. Nach der Implementierung und erster Tests stellt sich heraus, dass das Random Sampling die Kriterien am besten erfüllt, da die Lokale Suche sowie das Simulated Annealing die Lösungen zu langsam erstellen. Tabelle 3.1 zeigt die weiteren Kriterien, die für die Bewertung des Algorithmus wichtig sind. Auch hier schneidet das Random Sampling positiv ab.

Zusammenfassend ist nun festzuhalten, dass das Random Sampling für den Laufplaner genutzt werden soll, da es zwischen allen Algorithmen am besten abschneidet.

Tabelle 3.1: Bewertung des Random Samplings

Kriterium	Erfüllung
Parallelisierbarkeit	Lösungen können ohne großen Aufwand unabhängig voneinander generiert werden. Am Ende müssen diese nur noch synchronisiert werden, so dass die Lösung mit der besten Bewertung übernommen wird.
Speicherbedarf	Der Algorithmus benötigt kaum Speicher, da immer nur eine Lösung generiert wird und diese die vorherige beste Lösung überschreibt.
Anytime	Das Kriterium ist erfüllt, sobald eine Lösung generiert ist. Allerdings kann es passieren, dass in einer bestimmten Zeit noch keine gute Lösung vorhanden ist.
Anwendbarkeit	Der Algorithmus lässt sich auf jedes Problem der Laufplanung anwenden. Die Qualität hängt allerdings vom Anteil guter Lösungen, die im Lösungsraum vorhanden sind. Außerdem spielt die konkrete Implementierung des Algorithmus eine große Rolle. Insgesamt gilt, dass je länger der Algorithmus läuft, desto besser sind potentiell die Lösungen.

3.2 Erzeugen gültiger Lösungen

Der Algorithmus zur Erzeugung gültiger Lösungen für die Gesamtstrecke wurde initial von André Herms [1] entwickelt. Uli Ruffler [2] hat diesen in seiner Arbeit unter anderem dahingehend weiterentwickelt, dass dieser auf einer inkrementellen Arbeitsweise läuft. Der folgende Abschnitt geht nun auf das grundlegende Konzept der Generierung von gültigen Lösungen ein.

Wichtig ist, dass alle möglichen Lösungen generiert werden können und nur Lösungen ausgeschlossen werden, die außerhalb des gültigen Bereichs der Fußwinkel und Fußgeschwindigkeiten liegen, die auf instabilem Untergrund wie einem zu starkem Gefälle liegen und die den Roboter zum Umkippen bringen würden.

Das Generieren von Lösungen erfolgt nun in mehreren Schritten. Zu Beginn müssen einige Berechnungen für die Festlegung des Pfades zum Ziel ablaufen, bevor der Hauptteil des Algorithmus abläuft, welcher sich schrittweise an das Ziel nähert, bis der Abstand zum Ziel annähernd null beträgt.

3.2.1 Festlegung des Pfades zum Ziel

Eine erste Methode ist es, den Weg vom Start zum Ziel auf direktem Weg zu erreichen. Da allerdings nicht immer Lösungen auf direktem Weg möglich sind, müssen auch andere Pfade generiert werden. Beispielsweise könnte ein unüberwindbarer Graben oder eine hohe Mauer dafür sorgen, dass kein direkter Weg möglich ist. In diesem Fall müsste der Laufroboter den Weg um das Hindernis planen, um eine

gültige Lösung zu finden. Um nun alle Fälle abzudecken, benötigt man eine zufällige Streckenplanung. Diese erfolgt, indem zuerst die Anzahl der Pfadsegmente festgelegt wird und dann genau so viele Wegpunkte gesetzt werden.

Festlegung der Anzahl der Pfadsegmente

Die Festlegung der Anzahl der Pfadsegmente erfolgt über eine *geometrische Verteilung*. Diese Verteilung wird genutzt, da geringe Streckenanzahlen häufig sein sollen, da davon ausgangen wird, dass der direkte Weg der schnellste Weg ist. Höhere Streckenanzahlen sollen weniger oft auftreten. Durch die Veränderung des Parameters p lassen sich die Wahrscheinlichkeiten dahingehend variieren, dass entweder höhere oder niedrigere Streckenanzahlen bevorzugt werden. Standardmäßig geht man von $p = 0.5$ aus, was bedeutet, dass die Wahrscheinlichkeit genau einen Streckenabschnitt zu erhalten 50% beträgt.

Festlegung der Wegpunkte

Nun ist definiert, wie viele Segmente der Pfad haben soll. Jetzt müssen die Wegpunkte auf dem Pfad zufällig generiert werden. Für jeden Pfadpunkt P wird einzeln eine Lösung generiert. Zunächst wird der Mittelpunkt $m(x, y)$ sowie die Länge l zwischen Start und Ende berechnet. Auf den Mittelpunkt wird nun ein zufälliger Vektor $r(x, y)$ addiert, dessen generierter Wert vom negativen bis zum positiven Betrag der Länge l reicht. Gleichung 3.1 und Gleichung 3.2 zeigen mögliche Ergebnisse für drei Pfadsegmente P_1, P_2 und P_3 für den Startpunkt $S(0, 0)$ und dem Zielpunkt $Z(10, 0)$. Der letzte Pfadpunkt P_3 entspricht immer genau der Zielposition, damit der Roboter dieses Ziel in jedem Fall erreicht. Abbildung 3.1 zeigt die generierten Punkte und damit den Pfad des Roboters grafisch dargestellt.

$$P_1 = \begin{pmatrix} x_m + x_r \\ y_m + y_r \end{pmatrix} = \begin{pmatrix} 5 - 2.31 \\ 0 - 4.1 \end{pmatrix} = \begin{pmatrix} 3.31 \\ -4.1 \end{pmatrix} \tag{3.1}$$

$$P_2 = \begin{pmatrix} x_m + x_r \\ y_m + y_r \end{pmatrix} = \begin{pmatrix} 5 + 3.1 \\ 0 + 0.5 \end{pmatrix} = \begin{pmatrix} 8.1 \\ 0.5 \end{pmatrix} \tag{3.2}$$

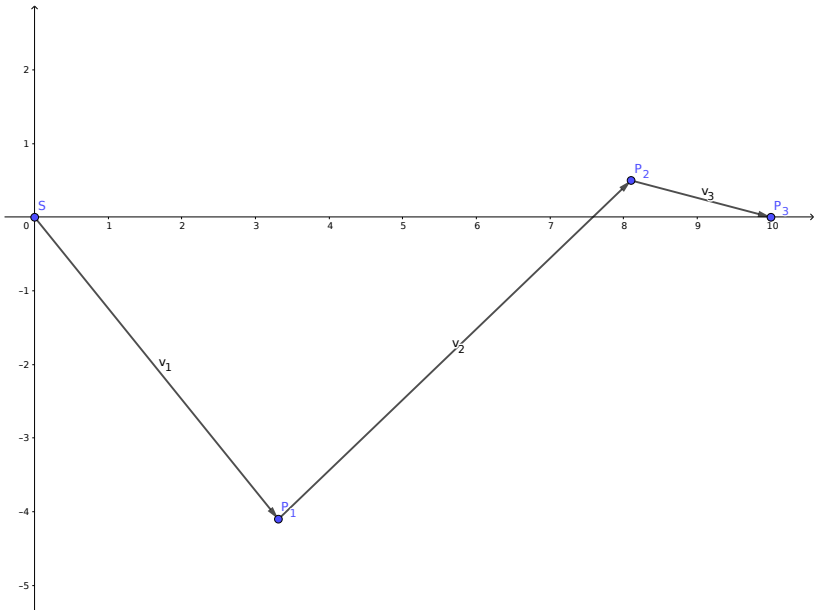

Abbildung 3.1: Zufällig generierte Wegpunkte

Mit dieser Berechnung sind nun für den Mittelpunkt des Körpers des Laufroboters ausschließlich Bewegungen auf diesem Pfad möglich. Bewusst wird hier noch nicht mit der z-Koordinate gearbeitet, da diese je nach Fußstellung und Terrain variiert.

3.2.2 Schrittweise Näherung an das Ziel

Die schrittweise Näherung an das Ziel erfolgt solange, bis der Abstand zum Ziel minimal ist. Ein exakter Vergleich mit dem Abstand null ist nicht effektiv, da der Roboter sonst fortwährend versuchen würde auf seine Zielposition zu gelangen, aber ununterbrochen minimale Abweichungen zum Ziel hätte, was den Prozess erneut anstoßen würde. Der Ablauf, um die Füße anzuheben oder abzusetzen sowie den Körper zu bewegen, läuft folgendermaßen ab:

1. Berechnung des zulässigen Bereichs
2. Auswahl der nächsten Fuß-Konfiguration

Neben den zwei Schritten nennt André Herms [1] noch zwei weitere Schritt zur Berechnung der Mittelpunktpositionen und der Festlegung der Dauer der Bewegungen. Dieser werden erst nach der schrittweisen Näherung an das Ziel durchgeführt.

Berechnung des zulässigen Bereichs

Der zulässige Bereich des Mittelpunkts ist auf Grund von zwei Kriterien eingeschränkt. Zum einen durch der maximalen Reichweite der Fußpunkte vom Mittelpunkt aus. Zum anderen dadurch, dass der minimale *Stability Margin* nicht unterschritten werden darf. Der Stability Margin ist der kleinste Abstand zum Mittelpunkt der konvexen Hülle der Fußpunkte vom Mittelpunkt aus. Dieser ist ein Maß dafür, wie stabil der Roboter steht. Ist der Stability Margin kleiner als null, so ist der Mittelpunkt außerhalb der konvexen Hülle, was ein Kippen des Roboters verursacht. Je größer der Stability Margin, desto stabiler steht der Roboter.

Der zulässige Bereich kann nur an zwei Punkten verlassen werden. Damit wird ein Start- und ein Endpunkt der aktuellen Fußstellung gebildet. Diese sind analytisch schwer zu berechnen. Daher muss mit einem numerischen Verfahren, der *binären Suche*, mit einem inneren zulässigen Punkt und einem äußerem unzulässigen Punkt der Schnittpunkt mit dem zulässigen Bereich gesucht werden. Dazu genügt eine Funktion welche angibt, ob der Mittelpunkt zulässig oder unzulässig ist.

Auswahl der nächsten Fuß-Konfiguration

Der Mittelpunkt des Roboterkörpers kann sich in diesem aktuellen Zustand nur noch vom zuletzt berechneten Start- und Endpunkt des zulässigen Bereichs bewegen. Dies geschieht durch das Verschieben der Körpermitte. Dazu müssen die Füße gehoben und umgesetzt werden. Auch hier müssen die zu Beginn genannten Kriterien beachtet werden. Diese sind unter anderem, dass sich mindestens drei Füße auf dem Boden befinden und dass mindestens ein Fuß auf jeder Seite den Körper stützt, damit der Stability Margin positiv ist. Mit diesen Regeln lassen sich 40 mögliche Fußkonfigurationen, auch Stützzustände genannt, darstellen, welche in Tabelle 3.2 abgebildet sind.

Mit jedem Übergang eines Stützzustands in einen anderen Stützzustands wird exakt ein Fuß entweder angehoben oder abgesetzt. Es können auch mehrere Füße gleichzeitig angehoben oder abgesetzt werden, indem in dem aktuellen Übergang eine Zeitdauer von $t = 0s$ angenommen wird. Damit wird der nächste Stützzustand mit dem aktuellen Stützzustand ausgeführt. Tabelle 3.3 gibt alle möglichen Stützzustände an.

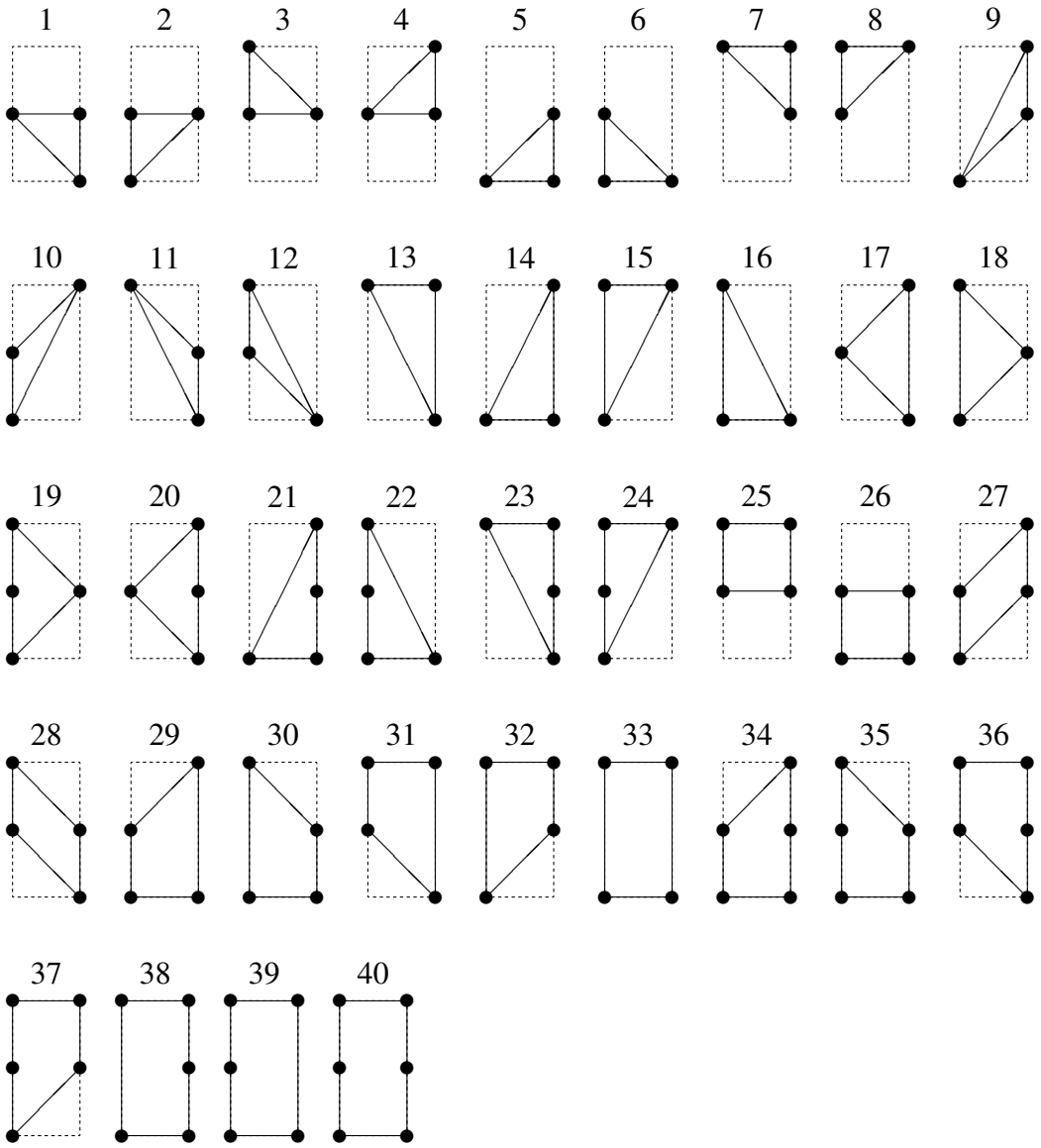

Tabelle 3.2: 40 zulässige Stützzustände [1]

Von dem aktuellen Stützzustand wird per Zufall bestimmt, wie der Nachfolgezustand sein soll. Dazu werden alle möglichen nächsten Zustände bewertet. Wird ein Fuß angehoben und verletzt dabei die Stabilitätskriterien, wird dieser Übergang verworfen. Aus den verbleibenden zulässigen Übergängen wird nun zufällig ein Übergang ausgewählt.

Zustandswechsel können theoretisch sofort wieder aufgehoben werden, was dazu führt, dass der Laufroboter sich nicht nach vorne bewegt. Daher müssen vorher veränderte Füße weniger gewichtet werden und Füße, die länger nicht verändert wur-

Stützzustand	*mögliche Nachfolger*
1	28 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 3.3: Transitionstabelle der Stützzustände [1]

den, höher gewichtet werden. Dies wird über einen Bonus geregelt. Die Berechnng läuft wie folgt ab:

- Ist ein Fuß verändert worden, wird sein Bonus auf null gesetzt.
- Ist ein Fuß nicht verändert worden, wird sein Bonus um eins erhöht.

Die Auswahl des nächsten Übergangs erfolgt über eine *Glücksradauswahl*. Dabei wird über eine Bewertungsfunktion $f(b_i) = (b_i)^2$ für jeden Fuß die Summe aller Boni gebildet. Die Wahrscheinlichkeit, dass ein Übergang, sprich eine Fußänderung, ausgewählt wird, hängt von der eigenen Bewertung des Fußes verglichen mit der Summe aller Bewertungen ab. Nach der zufälligen Auswahl des nächsten Zustands ergibt sich daraus entweder ein Anheben oder ein Absetzen des gewählten Fußes.

Beim Anheben eines Fußes kann die konvexe Hülle der Fußpositionen vom Mittelpunkt aus kleiner werden. Ist sie zu gering, kommt es zum Kippen des Roboters.

Das Absetzen eines Fußes ist in der Regel immer möglich, da die konvexe Hülle nur größer werden kann. Ausnahmen ergeben sich, wenn sich beispielsweise eine tiefe Klippe oder eine Mauer in der Nähe der Fußposition befindet. Beim Absetzen muss ein zufälliger Punkt in der Nähe des Fußes bestimmt werden. Als Erwartungswert wird hier ein in Richtung Ziel verschobener Mittelpunkt addiert. Sollte die Fußposition nicht gültig sein, läuft eine spiralförmige Suche rund um den Punkt ab. Laut André Herms passiert es nur sehr selten, dass dabei keine gültigen Lösungen gefunden wird.

Hat der Algorithmus nun schon die maximale Anzahl an Durchläufen erreicht, ist die Lösung ungültig. Ansonsten beginnt der Algorithmus nun wieder bei der Berechnung des zulässigen Bereichs für die neue Roboterposition, bis der Mittelpunkt des Roboters das Ziel erreicht hat.

3.2.3 Berechnungen des Mittelpunkts und der Bewegungsdauer

Da nun jeder Übergang, eingeschlossen der Fußposition, der Fußkonfiguration und des zulässigen Bereiches des Mittelpunkts definiert ist, können nun konkrete Werte für den zulässigen Bereich des Mittelpunkts definiert werden, damit auch Bewegungen mit konkreten Werten möglich sind. Da die zulässigen Bereiche der Übergänge sich überlappen, kann ein Mittelwert für die Bereiche des vorherigen und des nächsten Übergangs definiert werden. Dies gilt sowohl für Übergänge, bei denen ein Fuß angehoben als auch abgesetzt wird. Da eine Gleichverteilung ungünstige Ergebnisse erzielt, wird hier eine Dreiecksverteilung genutzt, die den vorherigen Mittelpunkt als Erwartungswert annimmt. Dadurch haben kürzere Bewegungen eine höhere Wahrscheinlichkeit.

Zum Abschluss wird noch die Zeit zwischen den einzelnen Übergängen benötigt. Die minimal mögliche Zeit hängt davon ab, wie lange der Mittelpunkt zur neuen Position benötigt und wie lange ein Fuß zum Absetzen benötigt. Um beide Kriterien einzuhalten ist das Maximum beider Werte die Zeit zwischen dem Übergang.

3.3 Bewertung gültiger Lösungen

Da das Random Sampling nach jedem Durchlauf nur die beste Lösung übernimmt, benötigt der Algorithmus Kriterien für die Bewertung. André Herms erstellt daraus

eine Bewertungsfunktion, welche für jede Lösung ausgeführt werden kann. Dabei nutzt er die folgenden Bewertungskriterien:

- Dauer der Bewegung
- Kippstabilität
- Untergrundstabilität
- Zulässigkeit der Lösung

Es sind noch weitere Kriterien denkbar wie der Energieverbrauch oder die Fehlertoleranz beim Ausfall eines Beins, welche allerdings nicht für die Umsetzung dieses Laufplaners eingesetzt wurden.

Kapitel 4

Portierung des Laufplaners nach ROS und Gazebo

4.1 Analyse bestehender Laufplaner

Herms, Ruffler, und AKrobat GitHub (gibt es dazu eine Arbeit?)

4.2 Allgemeiner Aufbau des Pakets

Die zuvor genannten Pakete müssen nun nach Robot Operating System (ROS) und Gazebo portiert werden. Ferner muss der Laufplanungsalgorithmus extrahiert und mit ROS kompatibel gemacht werden.

Ordnerstruktur oder vllt in Baustein, Laufzeit und Verteilungssicht?

4.3 Aufsetzen der Simulation

4.3.1 Aufsetzen des Roboter-Modells mittels urdf

Notes: urdf, xacro, Collissions, Inertia + Berechnung + STL-Dateien (Vereinfachung durch einfaches Geometry Object wenn möglich, sonst vereinfachtes STL, sonst das Original STL) / MeshLab

4.3.2 Definition der Gelenkmotoren mittels ros_control

Notes: URDF-File, config file, Controller

4.3.3 Aufsetzen der Umgebung mittels Gazebo

launch-files

4.3.4 Aufsetzen der Fußsteuerung des Laufroboters

4.4 Aufsetzen von Laufalgorithmen

4.4.1 Implementierung

4.4.2 Generierung von Bewegungen als xml-Datei

4.4.3 Einlesen und Abspielen der Bewegungen

Kapitel 5

Testen der Ergebnisse

Kapitel 6

Zusammenfassung

Kapitel 7

Ausblick

Abkürzungsverzeichnis

ROS Robot Operating System

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literatur

[1] A. Herms, „Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren“, Magisterarb., Otto-von-Guericke-Universität Magdeburg, Jan. 2004.

[2] U. Ruffler, „Laufplanung basierend auf realitätsnahen Umgebungsdaten für einen sechsbeinigen Laufroboter“, Magisterarb., Institut für Robotik der Fakultät für Informatik an der Hochschule Mannheim, Aug. 2006.