Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Master-Thesis

Jan Ruhnke

Ein vierbeiniger Roboter für unebenes Gelände

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Jan Ruhnke

# Ein vierbeiniger Roboter für unebenes Gelände

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. Ing. Andreas Meisel

Abgegeben am 23.07.2014

**Jan Ruhnke**

**Thema der Master-Thesis**

Ein vierbeiniger Roboter für unebenes Gelände

**Stichworte**

Laufende Roboter, Quadruped Rough Terrain Robot, USAR, hybrides System, Systementwicklung, Schrittplanung.

**Kurzzusammenfassung**

Diese Thesis befasst sich mit einem Gesamtkonzept für autonomes vierbeiniges Laufen von Robotern auf unebenem Untergrund und abseits von Wegen (*Quadruped Rough Terrain Robots*). Es werden zunächst vergleichbare Arbeiten vorgestellt und die Grundlagen für die betreffenden Forschungsfelder eingeführt. Danach wird der realisierte Roboter AMEE-XW2 vorgestellt. Unter Berücksichtigung der mechanischen Gegebenheiten wird ein hybrides Kontrollerkonzept mit den benötigen Verfahren erläutert. Die entwickelten Verfahren werden kritisch hinterfragt. Im Schluss wird das Erreichte zusammengefasst und einige Verbesserungen und Erweiterungen werden vorgestellt.

**Jan Ruhnke**

**Title of the paper**

A Quadruped Rough Terrain Robot

**Keywords**

Legged Robots, Quadruped Rough Terrain Robot, USAR, deliberative System, Robot Design, Step Planning

**Abstract**

This thesis describes an overall concept for autonomous four-legged walks of robots in a rough terrain environment (quadruped rough terrain robots). Starting with an overview about related elaborations, an introduction to the related research fields is made. After that the developed robot AMEE XW2 is presented. The hybrid controller concept and the required methods are explained, taking the mechanical circumstances into account. This thesis ends with a summary and future prospects are pointed out.

# Inhaltsverzeichnis

# Vorwort

Ich möchte mich besonders bei einigen Personen und Sponsoren bedanken.
Bei meinem Vater, der meine ungewöhnlichen 3D-Modelle und mechanischen Ideen in herstellbare Bauteile umgewandelt hat.
Bei Björn Bettzüche meinen Teamkollegen im Projekt AMEE und Kommilitonen, der immer mit Rat und Tat bei AMEE-XW2 dabei war.
Bei Prof. Dr. Gunter Klemke unserem Teamkollegen im Projekt AMEE und Mentor, der tatkräftig bei der Realisierung mitgeholfen hat und uns immer auf neue Ideen brachte.
Bei Prof. Dr. Kai von Luck, der den Fokus unserer Arbeit immer weiter getrieben hat und uns immer den „Rücken frei gehalten“ hat.
Bei Prof. Dr. Andreas Meisel, der mich immer wieder dazu angehalten hat, das Gesamtproblem zu betrachten. „36-dimensionaler Zustandsraum“
Bei Prof. Dr. Birgit Wendholt, die auch die chaotischste Präsentation mit uns durchgestanden hat und uns bei so manchem Problem geholfen hat.
Bei Prof. Dr. Bettina Buth, die uns immer bei den großen Dingen im Hintergrund geholfen hat.
Bei Prof. Dr. Thomas Flower, der immer an unser Projekt geglaubt hat.
Bei den Mittarbeitern der Zentralwerkstadt-HAW-Hamburg, die uns alles geduldig erklärt haben.
Und an meine Frau, die alle Dinge im Hintergrund erledigt hat.
Zudem an alle Helfer, die uns in vielen Situationen unermüdlich geholfen haben.

Ohne Euch hätte ich das nicht geschafft. Unglaublich wie weit wir Drei mit Eurer Hilfe gekommen sind.

Ein besonderer Dank geht auch an die Sponsoren:
Department TI der HAW-Hamburg
Microsoft Deutschland GmbH und Microsoft USA Ltd. (Danke Barbara, Peter, Gunter und Ben)
Ott Antriebe GmbH
Hoffmann + Krippner GmbH

# 1 Einleitung

**Die Vision:** *„Sie bekommen einen Anruf. Der Anrufer bitte Sie als Operator der Robotics Task Force um Ihre Hilfe, da es in einem Chemiewerk im 1.500km entferntem Mailand einen Unfall gegeben hat. Durch das Feuer kann keiner der Feuerwehrmänner das halb eingestürzte Gebäude betreten. In diesem Gebäude befinden sich die Gashauptleitung und mehrere Tanks mit flüssigem Wasserstoff. Sie begeben sich in Ihren Virtual-Reality-Raum und aktivieren das System. Plötzlich sehen Sie plastisch die Eingangstür zur Werkshalle in Mailand, ohne dunkle Ecken, unnatürlich scharf. Sie rufen mit einer Handbewegung den Bauplan der Halle auf und treten durch die Tür. Auf dem Bauplan der Werkshalle markieren Sie den Gashauptanschluss und stellen fest, dass die Zwischendecke schon teilweise eingestürzt ist. Der Bauplan ist nutzlos, alles sieht anderes aus. Dann sehen Sie den 12 Meter hohen Schutthaufen und fragen sich noch, ob man dort überhaupt raufklettern kann. Es erscheint ein grünes Symbol in Ihrem Sichtfeld. Der Roboter, dessen hochauflösenden Sinneseindrücke Sie wahrnehmen, bietet Ihnen an, selbständig auf die Kuppe des Schutthaufens zu steigen. Natürlich stimmen Sie zu. Als der Roboter noch steigt, sehen Sie eine Anzeige die Ihnen mitteilt, dass die Gashauptleitung 27m vor Ihnen liegt. Die Temperatur der Außenhaut Ihres Roboteravatars steigt auf 148°C und Sie haben die Kuppe des Schutthaufens erreicht. Plötzlich sehen Sie nur noch Feuer und Betonstücke fliegen. Sie geraten leicht in Panik und verlieren die Orientierung. Mit einer Ruderbewegung springen Sie aus dem Roboterkörper und sehen Ihren Roboter von oben rechts. Durch eine Drehbewegung Ihrer Hand, sehen Sie fast die komplette Werkshalle live in 3D. Sie springen zurück in den Avatar und schlagartig haben Sie das Gefühl zu fallen! Schwarze und absolute Stille… Sie schütteln den leichten Schock ab als Sie realisieren, dass Sie völlig unverletzt sind. Der Roboter wurde von einer Deckenplatte zerstört. Sie nehmen den zweiten Roboter und können das Gasleck mit einem Dichtmittel abdichten. Die unmittelbare Gefahr wurde beseitigt und die Feuerwehr vor Ort kann mit den erfassten 3D Daten langsam und vorsichtig mit den Aufräumarbeiten beginnen.“*
Diese Vision umfasst sehr viele Einzelthemen. Das Szenario ist hier die Telepräsenz. Der Operator trägt einen VR-Helm, der ihm die optischen und akustischen Sinneseindrücke des Roboters wiedergibt und durch seine Kopfbewegungen wird auch das Bild geschwenkt. Zudem dient der Helm als Display für den Bauplan und die Statusmeldungen des Roboters. Der eigentliche Schwerpunkt in dieser Vision ist aber der Roboter. Er liefert mit Stereokameras die live Bilder und ergänzt diese optischen Bilder mit 3D Tiefendaten aus einem Laserscanner. Er baut sukzessive die reale Umwelt für den Operator auf. Damit der Roboter überhaupt auf den Schutthaufen klettern konnte, um das Gasleck zu finden, braucht er wahrscheinlich Beine. Dieser Roboter muss steigen und extremen Umweltbedingungen standhalten können. Diese Vision ist die Motivation für das Projekt AMEE und dieser Master-Thesis.

In den nächsten zwei Dekade werden mobile und autonome Roboter zum Alltag der Bevölkerung gehörten und in das tägliche Leben integriert sein. Dabei werden wahrscheinlich diese Roboter als *Companions* empfunden werden. Dies ist eine weitreichende Vision vieler Wissenschaftler [Mic14], die noch enorme Anstrengungen erfordert aber mittlerweile erreichbar scheint. Eine abstrakte

übergeordnete Schlüsselfrage ist, ob diese Roboter eine universell einsetzbare Kopie des menschlichen Körpers darstellen oder ob es für jede Aufgabe spezielle Roboter geben wird. Die Idee eines humanoiden Roboters ist verlockend. Diese Roboter können Maschinen und Werkzeuge benutzen die für Menschen konzipiert wurden. Die nötige Entwicklung für einen humanoiden Roboter schient mechanisch noch lösbar zu sein [Hon071], scheitert zurzeit in seiner Gesamtheit aber an einer vollständigen Implementierung für generische Aufgaben. Roboter, die speziell für einen Aufgabenkomplex konzipiert wurden, müssen meist auch softwareseitig nur spezialisierte Fähigkeiten abdecken. Der größte Teil dieser autonomen Hilfsroboter bewegt sich in Gebäuden oder in einem erschlossenen Gebiet. Für die zivile Sicherheit, extraterrestrische Erkundungen und für militärische Aufgabe werden sogenannte *Rough Terrain Robots* verwendet, die an extreme Umweltbedingungen [BA-JR] angepasst sind und größtenteils Such- und Erkundungsaufgaben erledigen. In diesem Aufgabenbereich ist nicht nur eine angepasste Mechanik nötig, sondern auch eine tiefgreifend andere Herangehensweise an die Software. Fast jeder mobile Roboter muss bei Ungenauigkeiten und uneindeutigen Situationen Entscheidungen treffen können. Dies gilt besonders für Rough Terrain Robots, da diese Roboter meist in Umgebungen und Situationen eingesetzt werden, die für Menschen gefährlich sind. Bedingt dadurch muss der Roboter sich gegen die Umwelt durchsetzen können und schon fast rücksichtslos agieren, damit er sich beispielsweise aus einer ungünstigen Position befreien kann.
Für diese Thesis wurde ein vierbeiniger Rough Terrain Robot (*Quadruped Rough Terrain Robot* - **QRTR**) realisiert, ausgehend von einem Simulationsmodell, über die mechanische und elektronische Realisierung bis hin zur Teilimplementierung und einem Kontrollerkonzept. In dieser Thesis werden einige interdisziplinäre Designentscheidungen aufgezeigt. Dies erfolgt aber immer aus der Betrachtungsweise der Informatik. Einige benötigte physikalische und maschinenbauliche Hintergründe werden kurz an gegebener Stelle erläutert.

## 1.1 Hintergrund - Themenvielfalt

Das Laufen auf unebenem Untergrund ist eine große Herausforderung. Bei vierbeinigen Robotern ergibt sich diese Herausforderung durch die komplexen Zusammenhänge zwischen physikalischen Grenzen der Konstruktion des Roboters und den unterschiedlichsten Strategien zum Umgang mit Problemen. Einfach ausgedrückt besteht die Frage: „Wohin setze ich vier Füße, um in einer chaotischen nicht vorhersagbaren Umwelt voranzukommen und dabei nicht umzukippen“.
Wird beispielwiese der kleine vierbeinige Roboter LittleDog (2.7.1) mit seinen 12 Gelenken betrachtet, entsteht ein 36 dimensionaler Zustandsraum [Ale11]. Dabei hat jede Lösung ein begrenztes Intervall mit Abhängigkeiten untereinander, wie beispielweise die Schwerpunktstabilität und die möglichen Winkel der Glieder. Daneben sind sie auch abhängig von einer chaotischen und nicht vorhersagbaren Umwelt. Diese Vorstellung vernachlässigt noch viele Parameter. Ein derartiges Gleichungssystem ist schwer lösbar. Zudem muss eine Lösung für jeden Schritt errechnet werden. Damit ein flüssiger Bewegungsablauf entsteht, muss eine Lösung relativ zeitnah vorliegen. Bedingt durch diesen

Hintergrund wird erkennbar, wie schwierig es ist, einen vierbeinigen Roboter auf unebenem Boden laufen zu lassen.
Aus diesem Grund versuchen Wissenschaftler, den Zustandsraum zu verkleinern und das Problem zu zerlegen. Eine Lösung ist es, die Einzelprobleme pragmatisch anzugehen und für jedes Teilproblem ein eigenes Modell zu entwickeln. Dadurch entstehen zwar viele simple und handhabbare Lösungen. Diese Lösungen sind aber voneinander abhängig. Das Zusammenspiel dieser vielen Lösungen führt dazu, dass eine ganze Sammlung von Strategien und Verfahren betrachtet werden muss, um zu verstehen, wie moderne QRTRs in unebenem Gelände laufen.
Historisch betrachtet ist das Problem auf unebenem Boden zu laufen schon seit langem ein Forschungsgegenstand und begann, laut Alexander Shkolnik [Ale11], mit den von ihm als *bahnbrechend* bezeichneten Arbeiten von Mark Raibert: „M. H. Raibert, Legged Robots That Balance. The MIT Press, 1986.“ und „Raibert, M. H., Chepponis, M., Brown, and H. B., Running on four legs as though they were one, IEEE Journal of Robotics 1986 “.
Es folgten viele weitere Ansätze mit innovativem mechanischen Design (Dante II [Wet95]) bis hin zu reaktiven Verfahren, bei denen das System nur auf den aktuellen Zustand reagierte. Diese Systeme konnten aber nicht vorausschauend planen, wie eine Geländeformation bewältigt werden kann.
Heutige semiprofessionelle Roboter wie der BigDog oder der LS3 (3.2.3) von Boston Dynamics® verwenden ein reaktives und ein planendes System. Diese Roboter bewegen sich vorsichtig und planen hunderte Aktionen im Voraus, bevor sie einen Schritt wirklich ausführen.
Auch diese historische Entwicklung führt dazu, dass auf jede Teillösung ein weiteres Verfahren gesetzt wurde. Diese vielen Teillösungen müssen aber betrachtet werden, um ein Konzept für einen QRTR aufzustellen.
*Private Anmerkung des Autors: Als Student habe ich viele Teillösungen untersucht und auch mit ihnen experimentiert. Es fiel mir aber schwer, die Hintergründe einer Teillösung nachzuvollziehen. Es war schwer, aus den bekannten Teillösungen ein komplettes Konzept für einen QRTR zu realisieren, da oft in den betrachteten Arbeiten die Übergänge zwischen den Teillösungen offen blieben. Erst die vorgestellten Arbeiten in Kapitel 2.6 vermittelten das Wissen wie diese Teillösungen zusammenhängen und sich beeinflussen. Diese Erfahrung möchte ich gern mit dieser Thesis weitergeben und ich habe versucht, die einzelnen Hintergründe* ***kurz*** *zu erläutern. Für das Verständnis war es besonders wichtig, die Probleme interdisziplinär zu beleuchten. Aus einer reinen Informatiksicht war dies nicht möglich und ich hoffe der Leser verzeiht mir die vielen textuellen „Abstecher“ in die physikalische Welt. Der von mir sehr geschätzte Mark Raibert schrieb 1986 ein 250 seitiges Buch und viele Paper, um seine Ideen zu vermitteln. Aber genau dies macht den Reiz des Themas QRTRs aus.*

## 1.2 Aufbau dieser Master-Thesis

Dieser Thesis liegt ein einjähriges Projekt (Project AMEE) zugrunde, wodurch einige Vorarbeiten entstanden sind. Zudem hat sich der Autor in seiner Bachelorarbeit (2.8.1) [Ruh11] mit den Grundlagenthemen beschäftigt und diese Thesis setzt darauf auf. Bedingt dadurch und den begrenzten Umfang einer Master-Thesis können nur einige Teilaspekte dargestellt werden.
Der in dieser Thesis vorgestellte Roboter ist eine komplette Eigenentwicklung und verfolgt in seinem Aufbau und Kontrollerkonzepten einige neue Ansätze. Die Problemstellungen des vierbeinigen Laufens in unebenem Gelände wurden in Einzelprobleme zerlegt. Dadurch haben sich starke Abstraktionen der Teilprobleme ergeben, die *in* den einzelnen Kapiteln diskutiert werden damit der Zusammenhang zwischen Teilproblem und Lösung erhalten bleibt. Einige dieser Verfahren wurden von anderen Arbeiten inspiriert und weichen im Detail ab.
Parallel zu dieser Thesis entstand eine zweite Master-Thesis im Projekt AMEE mit dem Titel: *„Lernverfahren für die Wahl sicherer Schrittpositionen“* [Bet14]. Der Schwerpunkt der zweiten Thesis beschäftigt sich mit dem Reinforcement Learning und es wird mehrfach darauf verwiesen. Die dort entwickelten Verfahren können als Erweiterung für das hier vorgestellte Konzept verwendet werden.
Diese Thesis hat folgenden Aufbau:
**2. Kapitel Analyse:** In diesem Kapitel werden die Probleme erläutert, die sich ergeben wenn ein laufender Roboter sich in einer nicht urbanen Umwelt bewegt. Zudem werden einige grundlegende Lösungen aufgezeigt, die aber wiederum andere Probleme aufwerfen. Für die Lösung dieser Probleme werden einige wissenschaftliche Arbeiten vorgestellt, die diese Thesis stark beeinflusst haben.
**3. Kapitel Plattform Design:** Das Kapitel 3 diskutiert die Designentscheidungen, die die Hardware betreffen. Die mechanische Hardware wird an einigen Konstruktionsdetails erläutert und zudem der Bezug zum Kontrollerkonzept hergestellt. Die Entscheidungen für die elektronische Hardware werden anhand von Architekturmerkmalen der Software diskutiert.
**4. Kapitel Software Design:** Das Kontrollerkonzept wird in diesem Kapitel vorgestellt. Im ersten Teil dieses Kapitels werden die zu lösenden Probleme abgegrenzt und Lösungsansätze erläutert. Der zweite Teil dieses Kapitels stellt ein Gesamtkonzept für ein vierbeiniges Laufsystem vor.
**5. Kapitel Implementierung und Ergebnisse:** Hier werden einige Details der Implementierung und erste Ergebnisse erläutert. Ein weiterer Schwerpunkt dabei ist die veränderte Implementierung in den Beinkontrollern.
**6. Kapitel Schluss:** Abschließend wird das Gesamtkonzept zusammengefasst. Da dieses Projekt mit seinen Konzepten nicht abgeschlossen ist, werden auch definierte Arbeitspakete für die Weiterentwicklung erläutert.

# 2 Analyse

Dieses Kapitel stellt kurz die Grundprobleme von laufenden Robotern dar, wenn sie eine urbane Umgebung verlassen (2.1). Der Schwerpunkt dieses Kapitels beschäftigt sich mit bekannten Lösungen bei Laufsystemen und im Besonderen mit Lösungen für *Quadruped Rough Terrain Robots (QRTRs).* Im Kapitel 2.2 werden einige grundlegende Verfahren und Strategien erläutert. Das Kapitel 2.3 stellte zwei grundsätzliche Kontrollerkonzepte vor. Die Ziele des studentischen Projekts AMEE und die Ziele dieser Arbeit werden in Kapitel 2.4, 2.5 formuliert. Diese Arbeit wurde von einigen wissenschaftlichen Veröffentlichungen inspiriert und werden in Kapitel 2.6 zusammengefasst. Diese Arbeiten fußen, wie diese Thesis, auf einer eigenen Roboterplattform, die in Kapitel 2.7 kurz erläutert wird. Aus diesen Arbeiten sind eigene vorarbeiten zu dieser Thesis entstanden und werden zum Kapitelabschluss in 2.8 zusammengefasst. Dieses gesamte Kapitel dient der Orientierung und als Grundlage für die Designentscheidungen in Kapitel 3 und 4.

Die grundlegenden Konzepte hinter mobilen Robotern sind so vielfältig wie die verschiedenen Anforderungen. Zurzeit (Stand 2014) gibt es keinen allgemeingültigen Aufbau der Mechanik, Elektronik und Software für laufende Roboter. Seit den 1990ern gibt es aber Versuche ein standardisiertes Betriebssystem bzw. tiefgreifendes Framework für Roboter zu etablieren. Dies liegt vor allem an folgendem Umstand: Viele wissenschaftliche Untersuchungen von Laufsystemen sind oft nicht allgemeingültig, da es sich entweder um theoretische Untersuchungen handelt oder für einen bestimmten Robotertyp gelten. Für bipedale Roboter gibt es allgemeingültige Untersuchungen, die sich mit den grundlegenden Verfahren zur Stabilitätskontrolle beschäftigen. Diese Konzepte sind nur teilweise auf vierbeinige Roboter übertragbar. Es gibt aber wenige Arbeiten, die einen kompletten Überblick zu den Verfahren für vierbeinige Roboter liefern. Eine der umfassendsten Sammlung über vierbeinige Roboter stammt aus dem US-DARPA Programm *Learning Locomotion*[1] und deren Folgeprogrammen. Diese Arbeiten wurden an verschiedenen US-Universitäten erstellt und haben eine gemeinsame Roboterplattform. Diese Roboter wurden von der Firma Boston Dynamics® hergestellt und als „LittleDog“ (2.7.1) bezeichnet. Die in diesem Programm entwickelten Verfahren lassen sich aber nur teilweise für die Ziele dieser Arbeit anwenden.

[1] *Offizielle Kurzfassung des „Learning Locomotion“ Programms: In November of 2005, the **D**efense **A**dvanced **R**esearch **P**rojects **A**gency initiated a new robotics program, Learning Locomotion, designed to solve some of the key outstanding issues. The expectation was that a combination of machine learning techniques and smart development would accelerate the pace of making autonomous legged systems robust and useful.* [Def05]

## 2.1 Fragestellungen bei Rough Terrain Robots

In diesem Unterkapitel werden einige spezielle Fragestellungen bei Rough Terrain Robots aufgezeigt. Neben den offensichtlichen Fragestellungen [Ruh11] bei laufenden Robotern erschwert eine nicht urbane Umwelt und unebenes Gelände die Entwicklung. Für einen *Quadruped Rough Terrain Robots* (QRTR) ergeben sich weitere Probleme.

### 2.1.1 Positionierung abseits von Wegen

*Autonome* und *mobile* Roboter sind Maschinen, die sich im Raum frei bewegen und für die Erreichung eines Ziels bzw. Auftrags keines menschlichen Eingriffs bedürfen. Im Gegensatz zu Industrierobotern haben diese Roboter / Systeme keinen festen Fixpunkt, sondern haben meist sich selbst als Fixpunkt in relativer Beziehung zur veränderlichen umgebenen Umwelt. Bei Servicerobotern, die sich in geschlossenen und definierten Räumen bewegen, können Orientierungspunkte hinterlegt werden. An diesen Punkten kann sich der Roboter relativ zu diesen positionieren. Wird eine definierte Umwelt verlassen und begibt sich in eine nicht erschlossene Umgebung, wird diese Möglichkeit eingeschränkt. Für diese Umgebungen werden sogenannte *Rough Terrain Robots* eingesetzt und sind meist auch an extreme Umweltbedingungen angepasst. Die einfachste Möglichkeit der Orientierung für diese Roboter ist die Verwendung von *NAVSTAR®-GPS*, *GLONASS* oder beides als Sensorfusion. Dafür ist aber ein Einsatzzweck mit teilweise direktem Sichtkontakt auf den Himmel nötig. Ist diese Möglichkeit nicht gegeben, hat ein Rough Terrain Robot keinen verlässlichen Anhaltspunkt seiner absoluten Position. Der Roboter hat damit nicht nur das Problem, keine Pfadplanung anhand von z.B. hinterlegten topografischen Karten erstellen zu können, sondern auch das Problem, die eigene Interaktion mit der Umwelt überprüfen zu können. Hierfür ist eine zivile Satellitennavigation, auch als Sensorfusion, zu ungenau. Ein schwieriges Problem für Rough Terrain Roboter ist der Schlupf der Füße auf z.B. losem Untergrund, da er sich in seiner Umwelt bewegt, ohne selbst diese Aktion durchzuführen – da er rutscht. Allgemein wird die Erfassung des real zurückgelegten Wegs als *Odometrie* (2.2.5) bezeichnet.

### 2.1.2 Ungenaue Umwelterfassung

Ein Grundproblem von mobilen Robotern sind ungenaue oder fehlende Sensordaten der Umwelt und die folgerichtige Interpretation dieser Daten. Bei einem laufenden Rough Terrain Roboter erschwert beispielweise der Sichtschatten von kleinen Erhebungen die Auswahl von Trittpositionen. In unebenem Gelände wird der Sichtschatten so groß, das oft nur ca. 60% des Bodens erfasst werden können. Ein QRTR muss aber anhand dieser unscharfen Informationen komplexe Entscheidungen treffen und auch Fehler (z.B. Fehltritte) riskieren und kompensieren können. Dies steht im starken Gegensatz zu Industrierobotern und Numerischen-Maschinen, bei denen eine Ungenauigkeit außerhalb der definierten Toleranzen zu einem Fehlverhalten bzw. einer Notabschaltung führt. Bei Rough Terrain Robots sind diese Ungenauigkeiten sehr groß und normal. Dies ist auch bedingt durch die stärkere Ausprägung der Durchsetzungsfähigkeit im Vergleich zu anderen mobilen Robotern. Ein QRTR soll sich beispielweise effizient durch einen Wald bewegen. Um dies zu erreichen darf er nicht

jedem möglichen Hindernis ausweichen, was in unerschlossenen Gebieten oft nicht möglich ist. Um dieses Problem zu lösen, werden einige Konzepte in Kapitel 2.3 vorgestellt.

### 2.1.3 Schwerpunkt und Gleichgewicht

Um mit vier Beinen effizient auf unebenem Untergrund vorranzukommen, wäre es gut über jedes Hindernis steigen zu können. Dies würde bedeuten, dass ein QRTR möglichst lange Beine hat und extrem schmal ist. Dadurch liegt der Schwerpunkt sehr hoch und der Roboter würde leicht kippen. Beim Laufen wäre er schwer zu kontrollieren. Hat der Roboter kurze Beine und hätte eine große Schulterbreite, könnte er in fast jeder Gliederpose stabil stehen und laufen. Er müsste aber mehr Schritte machen, um die gleiche Strecke wie ein langbeiniger Roboter zu bewältigen. Damit wäre er langsamer und müsste komplexe Strategien verfolgen, um ein Hindernis zu bewältigen.
Diese beiden extremen Auslegungen beschreiben das Problem der Schwerpunktwahl bei der Konstruktion des Roboters und das Problem, wie komplex eine Steuerungssoftware sein muss. Diese beiden Aspekte sind untrennbar miteinander verknüpft und ergeben zudem die Herausforderung einer interdisziplinären Lösung. Die konstruktive Strategie wird in Kapitel 2.2.1 erläutert. Der Steuerungsaspekt (Softwarestrategie) wird in 2.2.2 diskutiert.

## 2.2 Verfahren und Strategien

Einige der oben vorgestellten Problemstellungen werden mit verschiedenen Strategien oder Verfahren gelöst. Diese Strategien sind immer eine Abwägung zwischen Extremen. Bei QRTRs gibt es bis heute (2014) keine bevorzugte Lösung. Die Entscheidung für eine bestimmte Strategie ist stark vom Einsatzzweck des Roboters abhängig.
In diesem Kapitel werden einige grundlegende Verfahren und Strategien vorgestellt. Viele Verfahren und Strategien sind durch die Forschung an zweibeinigen Robotern entstanden und bildeten die Basis für vierbeinige Roboter.

### 2.2.1 Konstruktiver Stabilitätstypen für QRTRs

In der Fachliteratur werden bei *Quadruped Rough Terrain Robot* (QRTR) keine Aussagen über die Lage des Massenschwerpunkts getroffen. Die Lage des Massenschwerpunkts[2] beeinflusst aber entscheidend die Implementierung des Gleichgewichtssystems und die Fähigkeiten des Roboters.
Generell werden nur zwei Läufertypen unterschieden, der *statische* und *dynamische* Läufer. Zudem gibt es den *pseudodynamischen* Läufer als Mischform beider Typen. Dies betrifft in der Klassifizierung alle laufenden Roboter.
**Statischer Läufer:** Bezogen auf vierbeinige Roboter und stark vereinfacht dargestellt, hat der *statische Läufer* seinen Massenschwerpunkt in Bodennähe und ist damit *nicht* kopflastig. In der Draufsicht ist der Masseschwerpunkt immer innerhalb einer gedachten Fläche (Stützpolygon) zwischen den Füßen,

[2] Massenschwerpunkt = realer Massenmittelpunkt. Masseschwerpunkt, Massepunkt = Modellvorstellung

damit steht der Roboter stabil. Unabhängig von seiner Beinstellung kann der statische Läufer nicht umkippen, solange mindestens eine *abstrakte* Dreipunktauflage über die Füße besteht. Oft wird eine Mehrpunktauflage, und damit ein größeres Stützpolygon, über große Füße erreicht. Das Stützpolygon wird dann über die Außenkanten der Füße auf dem Boden aufgespannt. Der Roboter muss sich aktiv bewegen und läuft dadurch langsamer als der dynamische Läufer.

**Dynamischer Läufer:** Im Gegensatz dazu hat der *dynamische Läufer* meistens einen hoch liegenden Masseschwerpunkt. Der reine dynamische Läufer hat oft nur einen abstrakten Punkt als „Stützpolygon“. Daraus resultiert auch, dass die meisten dynamischen Läufer, ohne aktives Gleichgewichtssystem, umkippen. Beim Laufen fällt er von einer Schrittposition in die nächste. Dazu wird der Masseschwerpunkt in die Laufrichtung verlagert und der Läufer kippt in diese. Abstrakt betrachtet, fangen die Beine nur das Kippen ab. In komplexeren Systemen wird dazu auch die Massenträgheit des Roboters genutzt (Zero-Moment Strategie 2.2.2). Damit ist es dem dynamischen Läufer möglich, schnell zu laufen. Beispielsweise erreichte der vierbeinige Roboter Cheetah (Boston Dynamics®) unter Laborbedingungen im Jahr 2013 einen Geschwindigkeitsrekord von über 46km/h [Bos131]. Der dynamische Läufer läuft energieeffizienter, da er nur Energie aufwenden muss, um den Schwerpunkt zu verlagern. Er bewegt nicht aktiv die Masse des Roboters. Die näheren Details wurden in den Vorarbeiten genauer geschildert [Ruh11].

**Pseudodynamische Läufer:** Der *pseudodynamische Läufer* ist eine Mischform beider Typen. Je nach Konstruktion tendiert sein Verhalten zu einem der obigen Typen. Wird das Stützpolygon vergrößert (z.B. große Füße) und der Schwerpunkt niedrig gehalten, verhält der Roboter sich grundsätzlich wie ein *statischer* Läufer. Um aber auch das dynamische Verhalten zu erreichen, wird das gedachte Stützpolygonmodell vergrößert. Es wird nur so groß gewählt, dass eine große Beinauslenkung den Masseschwerpunkt aus dem realen Stützpolygon schiebt. Damit ist es möglich kurze statische Schritte zu machen und lange *dynamische Schritte*. Der pseudodynamische Läufer kann gezielt in die Laufrichtung gekippt werden, wenn das Stützpolygon nur in die Laufrichtung vergrößert wird (Abbildung 2-1). Befindet sich der Läufer in einer stabilitätsgefährdenden Situation, kann er sich sofort in das statische Verhalten zurückziehen.

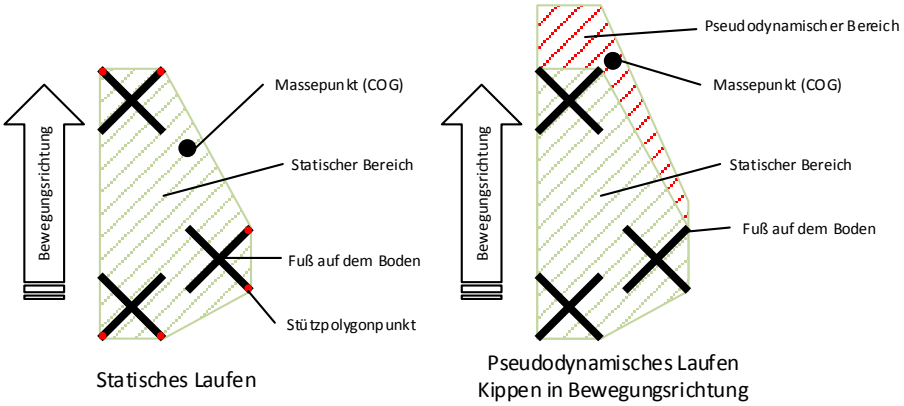


*Abbildung 2-1 Statisches und pseudodynamisches Laufen*

Bei jeder Schrittfolge muss der pseudodynamische Läufer sich vom dynamischen Bereich durch den statischen Bereich bewegen. Damit ist es nur schwer möglich, das anliegende Massenträgheitsmoment als flüssige Bewegung zu nutzen. Die Abstimmung der Größe des Stützpolygons ist eine Balance zwischen Sicherheit und nutzbarem Trägheitsmoment. Eine stark abstrakte Analogie wäre, sich das dynamische Laufen als rollende Kugel vorzustellen – schwierig zu kontrollieren, aber weniger Kraftaufwand. Das statische Laufen, entspricht dem holprigen Rollen eines Würfels – leicht zu kontrollieren, aber mit höherem Kraftaufwand. Beim „rollen" verharrt er auf der nächsten Fläche. Der pseudodynamische Läufer würde dann einem Würfel mit abgerundeten Kanten entsprechen. Soll dieser bewegt werden, ist die nötige Kraft schon kleiner als beim „statischen Würfel". Der abgerundete Würfel lässt sich auch leichter kontrollieren als die „dynamische Kugel". Wie stark die Kanten des Würfels abgerundet werden, beeinflusst sein Verhalten. Diese Vorstellung entspricht grob der Abstimmung zwischen dem statischen und dynamischen Bereich beim pseudodynamischen Läufer. Hohe Stabilität und hoher Kraftaufwand – versus - geringe Stabilität und wenig Kraftaufwand. Bedingt durch diesen Zusammenhang kann der pseudodynamische Läufer nicht so schnell laufen wie ein reiner dynamischer Läufer.

**Vor- und Nachteile:** Diese drei grundlegenden Konstruktionsarten wirken sich direkt auf die Fähigkeiten des QRTR aus. Der *dynamische Läufer* kann in ebenem Gelände schnell laufen. Für stark unebenes Gelände eignet sich der rein dynamische Läufer weniger, da er ständig in Bewegung bleiben muss um das Gleichgewicht zu halten. Verbesserte Gleichgewichtssysteme können zwar ruhiger stehen, beim Steigen aber nur kurzzeitig einen Fuß vom Boden heben. Dieses Zeitfenster beläuft sich, je nach Fähigkeiten des Gleichgewichtssystems, auf ca. 100ms bis max. 5 Sekunden [Bos10]. Bei komplexeren Gleichgewichtssystemen wird bei angehobenem Fuß der gesamte Torso des Roboters gegen die Fallrichtung verschoben, um die Auslenkung des Masseschwerpunkts möglichst klein zu halten. Dies ist eine vereinfachte Betrachtung. Der rein *statische Läufer* kann **nicht** die Zero-Moment-Balance Strategie (siehe 2.2.2) nutzen, da er nicht in eine Richtung kippen kann. Er muss aktiv seine eigene Masse bewegen und kann damit auch nicht das Massenträgheitsmoment nutzen. Die Folge ist ein langsames Schreiten des Roboters und ein höherer Energiebedarf. Dafür können reine vierbeinige statische Läufer im Passgang stabil auf zwei Beinen stehen, ohne ein aktives Gleichgewichtssystem zu verwenden. Der *pseudodynamische Läufer* (3.2.4) vereint diese Fähigkeiten. Beim Laufen kann er aber nicht die Geschwindigkeit des dynamischen Läufers erreichen.

Die Konstruktion ist damit vom Einsatzzweck abhängig bzw. vom geplanten Einsatzort. Diese Einschränkungen der Läufertypen gibt es auch in der Natur. Einem Rennpferd ist es kaum möglich, auf Steilhängen zu steigen. Für stark unebenes Gelände ist der *pseudodynamische* und *statische* Läufer interessant, da der vorausschreitende Fuß auch zum Ertasten der Tragfähigkeit des Bodens genutzt werden kann. Rutscht beispielsweise der Boden beim Ertasten unter dem Fuß, können diese Typen den Fuß zurückziehen. Der *dynamische Läufer* kann dies meist **nicht**. Er muss erst den bewegten Fuß belasten, um seinen Körper zu stabilisieren. Die in dieser Thesis getroffene Designentscheidung wird in Kapitel 3.1 konkretisiert.

### 2.2.2 Grundlegende Gleichgewichtsstrategie

Um dynamische Läufer im Gleichgewicht zu halten, wird meist ein Verfahren aus dem Bereich der *Zero-Moment-Balance* Strategie verwendet, das auf den Prinzipien bzw. Implementierungen zum invertierten Pendel basieren. Dabei wird der sog. *Zero-Moment-Point (ZMP)* [Vukobratovic & Juricic 1969] so verlagert, dass das resultierende Massenträgheitsmoment einer Bewegung den Roboter stabilisiert. Dieses Verfahren funktioniert nur bei ausreichender Bewegungsfreiheit. Bei kleinen und langsamen Bewegungen, wie beispielsweise dem Balancieren auf einem schmalen Steg, ist das Trägheitsmoment als gesteuerte Gegenkraft zu klein. Hierfür wird als zweites Verfahren die sog. *Moment-Balance Strategy* [Pop05] verwendet. Beide Verfahren finden oft in einer Maschine Anwendung und werden je nach Situation umgeschaltet.
Diese Verfahren wurden für die Verwendung in Ein- und Zweibeinern eingeführt und beispielsweise für den vierbeinigen LittleDog (2.7.1) modifiziert. Einige Arbeiten aus dem Learning Locomotion Programm benutzen die obigen Verfahren nur als Grundüberlegung für simplere Verfahren [Reb08], da sie einen großen Modellierungs- und Kalibrierungsaufwand haben. Das simplere Verfahren (4.1.2) zeigt ein ähnliches Verhalten wie die *Moment-Balance* Strategie, wenn es in einem bestimmten Takt angestoßen wird. Das System reagiert nicht sofort auf ein Kippen, sondern lässt den Roboter für eine bestimmte Zeit einfach kippen. Dieser veränderte Ansatz wird ab Kapitel 4.1.1 erläutert.

### 2.2.3 Dynamische Laufmuster – „Walking Gaits“

Als Walking Gaits wird die Abfolge von Fußpositionen und Bewegungsabläufen bezeichnet, die dazu führen, dass der Körper sich fortbewegt (Body Shift). Feste Laufmuster sind im Gelände eher nachteilig und finden in heutigen Laufrobotern wenig Anwendung. Vor allem im Gelände ist eine statische Abfolge der Schritte wenig erfolgreich, da oft nur kleine Schrittkorrekturen zur Massenschwerpunkt-stabilisierung des Roboters durchgeführt werden müssen. Dies gelingt aber nicht immer mit einer starren Abfolge von Schritten. Verwendet der Roboter ein planendes Verfahren (hybrides System 2.3.1) besteht *mit festen Laufmustern* das Problem, dass sogenannte Ausfallschritte explizit modelliert werden müssen. Berechnet der planende Teil des Gleichgewichtssystems auch die Fußauswahl für den nächsten Schritt, sind die Ausfallschritte - um den Roboter im Gleichgewicht zu halten – das zwangsläufige Resultat dieser Berechnung. Diese Strategie wird genauer in Kapitel 4.3.3.3 erläutert.

### 2.2.4 Durchsetzungsfähigkeit von Rough Terrain Robots

QRTRs sollen auch effizient in unwegsamen Gelände vorankommen. Dieses Problem wurde in Absatz 2.1.2 bereits angedeutet. Eine Strategie, mit unbekannten Hindernissen umzugehen, wird durch ein Beispiel verdeutlicht, bei dem ein Roboter sich abseits von Wegen durch einen Wald bewegt. Bei dieser Aufgabe erfasst der Roboter seine Umwelt, beispielsweise optisch mit einem Laserscanner. Die erfassten Sensormesswerte werden vom Roboter über eine 3D-Punktwolke und schließlich als separierte Objekte abstrahiert. Anhand dieser Punktwolke bzw. abstrakten Objekten kann eine massive Eisenstange nicht von einem dünnen Ast, mit gleichem Umfang, unterscheiden werden. Um

sich aber effizient in einem Wald zu bewegen, müssen kleinere Hindernisse einfach ignoriert oder auf Tragfähigkeit getestet werden. Es wird also nicht einfach jedes Hindernis umgangen, sondern abgewogen, ob der Ast wirklich ein Hindernis darstellt und vom Roboter zerbrochen werden kann. Dieses Verhalten wird als *Durchsetzungsfähigkeit gegenüber der Umwelt* bezeichnet und ist nicht für die Interaktion mit Menschen geeignet[3]. Die obige Strategie ist eine Abwägung zwischen Effizienz und Vorsicht der Maschine. Eine rudimentäre Umsetzung diese Strategie wird durch das einfache Ignorieren von Objekten erreicht und in Kapitel 4.3.3.1.2 beschrieben.
Eine vollständig funktionierende Umsetzung müsste das implizite und gelernte Wissen von Menschen und Tieren nachbilden. Menschen wissen, wie beispielsweise ein Busch aussieht und können daraus folgern, dass sie einfach hindurch laufen können. Das Umgehen von jeglichen Hindernissen und der damit einhergehende Zeitverlust wären für einen Rettungsroboter in dieser Umgebung eher nachteilig.

### 2.2.5 Odometrie bei laufenden Robotern

Bei Robotern mit Beinen eignen sich oft nur optische Verfahren, um das unabsichtliche Rutschen der Füße (Drift) zu registrieren. Es gibt eine Reihe von Verfahren, wie vom Drift einer Punktwolke bis hin zum völligen Ignorieren dieses Umstands. Der Ansatz des Ignorierens und der ständigen Neuorientierung scheint bei professionellen Robotern, wie vermutlich dem LS-3[4] von Boston-Dynamics®, erfolgreich zu sein. Eine optische Odometrie wäre beispielsweise durch eine SIFT & SURF Lösung machbar. Auch neuere laufzeiteffizientere Verfahren auf Clusterbasis in Kombination mit Farbinformationen bietet sich hier an [Fra12]. Für den verwendeten Kinect® V2 Sensor liefert die Kinect® Fusion API diese Odometriedaten. Dazu werden Matrizenüberdeckungen der Punktwolken gesucht und deren Verschiebung beobachtet (3.4.3.1). Diese Softwarekomponente ist auf den Sensor angepasst und kalibriert. Weiterhin ist Kinect® Fusion out-of-the-box lauffähig. Leider lag zum Zeitpunkt dieser Arbeit noch keine lauffähige Fusion Version für den Kinect® V2 Sensor vor.

## 2.3 Grundlegende Kontrollerkonzepte

Intuitiv betrachtet könnte das reine geradeaus Laufen auf **ebenem** Untergrund ein Regelungssystem sein, das vollständig beschreibbar über ein Gleichungssystem sein könnte. Diese erste Vermutung ist nicht haltbar. Das vierbeinige Laufen auf **unebenem** Untergrund ist komplexer. Wie im Kapitel 2.1.2 beschrieben wurde, muss das System Entscheidungen anhand von ungenauen Sensordaten treffen. Hinzu kommt, dass das System Annahmen treffen muss, obwohl keine Sensordaten vorliegen. Dies ist bedingt durch den Sichtschatten der Sensoren (Abbildung 2-2). Bei QRTRs bedeutet dies konkret, dass der Roboter oft nicht wirklich „sehen“ kann, wohin er tritt. Der Roboter trifft die Annahme, dass beispielsweise hinter einer Erhebung der Boden tatsächlich weitergeht. Aus dem Blickwinkel der meisten QRTRs gilt dies sogar für den Sichtschatten von kleinen Unebenheiten auf kurze Distanz

[3] Hierfür sollte eine zuverlässige „Personenerkennung“ verwendet werden.
[4] Der Boston-Dynamics LS-3 ist ein militärischer Prototyp. Aus diesem Grund werden kaum wissenschaftlich belastbare Details veröffentlicht.

[And09]. In den nächsten zwei Kapiteln werden anhand des obigen praktischen Beispiels einige Strategien vorgestellt, wie mit diesen Ungenauigkeiten umgegangen werden kann.

### 2.3.1 Reaktive und hybride Systeme

Ein hybrides System besteht aus einem reaktiven und planenden Teil. Bei einem Laufsystem reagiert der reaktive Teil auf Umwelteinflüsse und hält ein Laufsystem möglichst stabil. Der planende (deliberative) Teil plant Aktionen im Voraus. In vielen Umsetzungen trifft dieser Teil auch Annahmen aufgrund von Wahrscheinlichkeiten. Diese Planungen werden vom reaktiven Teil des Systems ausgeführt. Um die Unterschiede –zwischen einem reinen reaktiven und einem hybriden Systemen - und die Vorteile des hybriden Systems zu erläutern, wird das Problem des Sichtschattens bei beiden Strategien betrachtet.

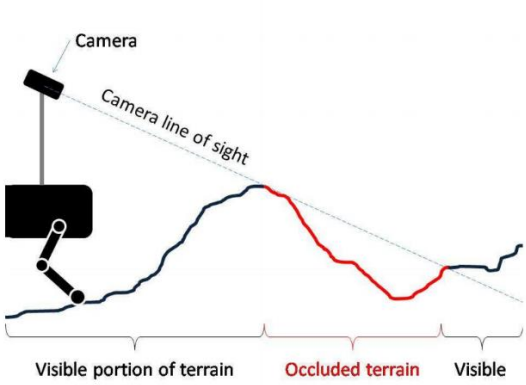


*Abbildung 2-2 Sichtschatten [And09]*

Um das Problem der Fehltritte quantifizieren zu können, wurde die Veröffentlichung von Kolter, Andrew und Kim [And09] (Abbildung 2-2) betrachtet. In dieser Arbeit werden Messergebnisse über Fehltritte vorgestellt. Tests mit dem Roboter LittleDog (2.7.1) haben ergeben, dass die erfassten Trittpositionen[5] nur zu 80% richtig waren. Die Schlussfolgerung daraus ist, dass bei einhundert Schritten – zwanzig fehlerhaft sind. Dafür muss das System eine schnelle Fehlerbehandlungsstrategie bereithalten. Ohne eine Strategie tritt der Roboter in ein Loch und stürzt oder der Roboter wirft sich selbst um, da er den Boden weiter entfernt „vermutet“.

**Reaktives System:** Ein rein reaktives System nutzt die internen Sensoren, um diesen Fehltritt zu erkennen. Dazu wird oft ein interner Lagesensor genutzt. Wird ein Fuß auf eine erfasste Trittposition bewegt, wird der Fuß über die Position gefahren. Danach wird der Fuß solange nach unten bewegt, bis der Torso sich neigt und die Beinbewegung wird gestoppt. Das System versucht ständig den Torso, über die Beine des Roboters, in Waage zu halten. Dieser einfache Ansatz führt aber schon bei leicht unebenem Untergrund zu Problemen. Tritt der Roboter in ein tiefes Loch und der Fuß erreicht den Boden nicht, fällt der Roboter trotzdem um. Das System kann nicht den Grund für die Neigung erkennen und reagiert zu stumpf. Um eine bessere Erkennung zu ermöglichen, werden Drehmomentsensoren, Trägheitssensoren und Drucksensoren in den Füßen kombiniert. Eine Möglichkeit wäre, auf den Kontakt der Füße mit dem Boden zu reagieren und zusätzlich das Drehmoment der Antriebe zu überwachen. Mit einem Ausschlussverfahren kann folgerichtig reagiert werden und beispielweise bei einem Loch der Fuß zurückgezogen werden.

[5] Diese Arbeit verwendet ein hybrides System mit zwei planenden Schichten. Der Schwerpunkt der Untersuchung beschäftigt sich mit einer Strategie zur vorhersagte von Bodenformationen im Sichtschatten.

Das reine reaktive System hat aber das Problem, das bei einem Fehler die komplette Verfahrenskette durchlaufen werden muss. D.h. erfolgt ein Fehltritt und der Roboter kippt, muss in dieser Zeit eine neue Position für den Fuß gefunden werden. Bei einem dynamischen Laufverhalten bleibt meistens nur ein Zeitfenster von 100ms [Pop05]. Ist dies nicht möglich, kann nur versucht werden, den Fuß auf die alte und bekannte Position zurückziehen. Dabei muss auch die physikalische Bewegungszeit der Mechanik berücksichtigt werden. Das reaktive System kann keinen komplexen Alternativplan erarbeiten, um ihn für eine kritische Situation bereitzuhalten.

**Hybrides System:** Das reaktive System wird mit einem planenden Teil erweitert. Für den planenden Teil gibt es unterschiedliche Strategien. Ein Merkmal dieser Systeme, bei laufenden Robotern, ist das der deliberative Teil keinen Zugriff auf die Mechanik hat. Dieser Teil plant die nächsten Aktionen und versucht mehrere Alternativen bereitzuhalten. Tritt nun ein fataler Fehltritt auf, der vom reaktiven Teil nicht kompensiert werden kann – ruft der reaktive Teil, einen vom deliberativen Teil vorberechnete Alternative ab und führt den alternativen Plan aus. Im Detail wird dieses Vorgehen im Kapitel 4.3.3 vorgestellt. Dieses Konzept konnte auch erfolgreich im Learning Locomotion Programm verwendet werden und bildet die Grundlage des BigDog [Bos10].

Eine weitere Form ist eine KI bzw. ein Machine Learning Verfahren im deliberativen Teil zu verankern. Das Feedback für den Lernerfolg des Systems, erfolgt dann vom reaktiven Teil. Weitere Details werden in der Thesis von AMEE Teammitglied Bettzüche [Bet14] vorgestellt, die parallel zu dieser Thesis erstellt wird.

Es gibt auch Versuche, den deliberativen KI-Teil auf markante Geländeformationen zu trainieren. Anish Adukuzhiyil und Team [Ani09] haben dafür ein neuronales Netz auf optimale Trittpositionen trainiert. Das Feedback erfolgte durch einen menschlichen Trainer. Hier konnte erfolgreich gezeigt werden, dass der Roboter – nach der Trainingsphase - bei ähnlichen Situationen richtig reagiert hat. Für das hybride Konzept gibt es eine Vielzahl unterschiedlicher Ansätze. Es konnte aber fast immer eine erhebliche Verbesserung des Laufverhaltens gegenüber einem reinen reaktiven System gezeigt werden.

Die Anwendung dieser Strategie wird als hybrides System mit einem *reactive- / deliberative Layer* [Bra95] bezeichnet. Der Reactive-Layer folgt dabei der Subsumption von Brooks [Rod86], wobei der Deliberative-Layer verschiedene Ausprägungen haben kann. Seit dem Jahr 2003 wird die Mehrheit der semiprofessionellen QRTRs mit einem hybriden System betrieben.

Eine eigene Architektur für ein hybrides System wird in Kapitel 4 vorgestellt. Der reaktive Teil wird in Absatz 4.3.4 erläutert. Der deliberative Teil wird in 4.3.3. vorgestellt, aber beschränkt sich auf ein einfaches planendes Verfahren und ist vorbereitet für einen weiteren KI-Layer.

### 2.3.2 Alternativer Embodiment Ansatz

Ein nicht weiter nicht betrachteter alternativer Ansatz für ein QRTR wäre der „embodied intelligence“ Ansatz, dieser ist eine These aus der neueren Kognitionswissenschaft. Nach dieser These braucht ein Bewusstsein einen Körper und damit die physikalische Integration mit seiner Umwelt. Angewendet auf die KI bedeutet dies, dass ein lernendes System die Interaktion mit Sensoren und Aktoren braucht, um zu lernen. Diesen Zusammenhang wird beispielweise von Rolf Pfeifer in seinem Buch „How the body

shapes the way we think : a new view of intelligence" [Pfe07] beschrieben.
In diesem Forschungsbereich besteht ein rudimentärer und grundlegender Versuchsaufbau für laufende Roboter aus einem Laufsystem, bei dem alle Aktoren und Sensoren direkt von einem KI-Verfahren gesteuert werden. Beim Reinforcement Learning [Sut98] werden die „Belohnungen" und „Bestrafungen" so modelliert, dass das System lernt, wie es die Aktoren ansteuern muss um sicher zu laufen. Der „supervised learning" Ansatz beinhaltet beispielweise ein neuronales Netz, bei dem ein Trainer die richtige Ansteuerung der Glieder mit dem System „übt". Neben den „learning" Verfahren gibt es hunderte andere Ansätze, die in Laufsystemen erprobt wurden. Für diesen Zweck wurden beispielweise einige vierbeinige Roboter an der ETH-Zürich[6] entwickelt (Projekt: Schmaroo, Geoff, MiniDog).
Der Trend im KI-Forschungsumfeld geht zu einer Kombination mehrerer Ansätze, indem unterschiedliche Learning-Verfahren in Schichten angeordnet werden. Die Hoffnung der Wissenschaftler ist, dass sich diese Systeme verhalten wie Menschen oder Tiere. Menschen können meistens genau vorhersagen, dass sich im Sichtschatten einer 5cm hohen Unebenheit kein 2m tiefes Loch befindet. Obwohl viele Erfolge erzielt wurden, bleibt das Problem der Merkmalsabstraktion. Zudem scheinen oft viele unabhängige Merkmale wichtig zu sein, was zu einer Explosion des Merkmalsraum führt. Ein Grundproblem dieser Systeme ist aber die Vorhersagbarkeit des Verhaltens, um die Zuverlässigkeit einschätzen zu können. Für den Roboter AMEE bedeutet dies, dass in dieser Thesis diese Ansätze nicht weiter betrachtet werden. Das Risiko von körperlichen Verletzungen scheint zu hoch, da der Roboter AMEE zu schwer und kräftig ist.

## 2.4 Ziele des Projects AMEE

Durch das Programm *Learning Locomotion* rückten *Quadruped Rough Terrain Robots* (QRTR) in den Focus einiger Wissenschaftler und es wurden in kurzer Zeit erhebliche Fortschritte erzielt. Diese Master-Thesis und das Projekt AMEE orientierten sich hauptsächlich an den Veröffentlichungen aus diesem Programm. Die genaue Abgrenzung dieser Master-Thesis wird in Kapitel 2.5 erläutert.
**Projekt AMEE:** Das studentische Projekt AMEE hat das Ziel, die benötigten Verfahren für einen vierbeinigen und geländegängigen Roboter zu ermitteln und zu realisieren. Dieses Laufsystem soll mechanisch und softwaretechnisch möglichst einfach aufgebaut sein. Durch eigene Vermutungen und Andeutungen aus dem „Learning Locomotion" Programm war zu erkennen, dass bekannte Verfahren von kleinen vierbeinigen Robotern (z.B. Little Dog) sich nicht direkt auf größere Maschine übertragen lassen. Zum Zeitpunkt dieser Arbeit war kein Roboter mit einer Masse von mehr als 7kg erwerbbar. Dadurch war es auch ein Ziel, einen QRTR zu konstruieren und kostengünstig zu fertigen. Erschwerend kam hinzu, dass es für diese Leistungsklasse keine generisch verwendbare Elektronik gibt. Diese Elektronik musste entwickelt und realisiert werden. Es musste ein eigenes Kontrollerkonzept entwickelt werden.

[6] http://www.ifi.uzh.ch/ailab/robots.html

Diese Rahmenbedingungen führen zu folgenden Zielen für das Projekt:

a. Der Roboter soll konstruiert und realisiert werden.
b. Der Roboter soll autonom auf leicht unebenem Gelände laufen können.
c. Der Roboter soll erweiterbar für extremes Gelände sein.
d. Der Roboter soll agil sein und keine Totpunkte in den Beinen haben.
e. Der Roboter soll eine Schrittlänge von ca. einem Meter haben.
f. Der Roboter soll kostengünstig sein.
g. Der Roboter soll einfach gefertigt werden können.
h. Die Hard- und Software soll möglichst modular aufgebaut.
i. Die Software soll ein verteiltes System sein.
j. Die Software soll eine Multiprozessorplattform ausnutzen können.
k. Die Vorarbeiten (Simulation und Bachelorarbeiten) sollen verwendet werden.
l. Die Software soll möglichst schlanke Algorithmen verwenden.
m. Der Echtzeitanteil soll vom restlichen System getrennt sein.

Die Unterpunkte a. - g. betreffen zwar die Konstruktion des Roboters, sind aber entscheidend für das Design der Software (4).

## 2.5 Ziele dieser Thesis

Eine Master-Thesis bietet nicht den zeitlichen Spielraum, um alle Ziele des Project AMEE zu untersuchen und zu realisieren. Aus diesem Grund werden auch nur einige Punkte bearbeitet. Ein Ziel dieser Thesis ist es den Roboter so weit zu realisieren, dass auch andere diese Maschine weiterentwickeln können.
Die Ziele in dieser Thesis sind wie folgt:

I. Die Konstruktion des Roboters AMEE-XW2 unter dem Gesichtspunkt der Softwarearchitektur (3.3).
II. Die Erstellung von 3D Modellen für die Simulation und die CNC-Fertigung des Roboters AMEE-XW2 (3.3).
III. Die Fertigung und Montage des Roboters AMEE-XW2.
IV. Die Entwicklung und Realisierung der Elektronik für den Roboter AMEE-XW2 (3.4).
V. Die Aufstellung eines Gesamtkonzepts für den Roboter AMEE-XW2 (4).
VI. Die Softwaremigration der Beinkontroller aus der vorausgegangen Bachelorarbeit [Ruh11] auf eine leistungsfähigere MCU Plattform (5.1.1).
VII. Die Integration der Beinkontroller in den reaktiven Layer des Roboters AMEE-XW2 (4.3.4).
VIII. Die Entwicklung eines sog. Upstream-Realtive-Layers zur Synchronisation und Koordination der Beinkontroller (4.3.4).
IX. Die Entwicklung eines Verfahrens zur Umweltabstraktion (4.3.2.2).

X. Die rudimentäre Entwicklung eines Verfahrens zur planenden Schrittauswahl (4.3.3).
XI. Die Konzeption des Deliberative-Layers (4.3.3).

Die Punkte I. bis IV. begründen einige Designentscheidungen zu den Punkten V. bis XI. Verzichtet werden auf Verfahren zur Odometrie und zum Wenden auf der Stelle mit vier Beinen. Auch auf die reale Integration des Deliberative-Layers in das Gesamtsystem wird verzichtet.

## 2.6 Ähnliche Arbeiten

Eines der Ziele, ist es einen eigenen QRTR zu entwickeln. Viele wissenschaftliche Arbeiten zu vierbeinigen Robotern sind nur isolierte Teillösungen und auf unterschiedlichste Roboterplattformen angepasst. Die Vielzahl der hochwertigen wissenschaftlichen Arbeiten aus dem Learning Locomotion Programm basieren alle auf einer Roboterplattform, dem LittleDog (2.7.1). Zudem sind hier einige Arbeiten entstanden, die ein gesamtes Konzept beschreiben und dabei neue Wege beschreiten. Bedingt durch diesen Umstand konzentriert sich diese Thesis auf diesen Forschungskreis, ohne dabei andere Arbeiten zu vernachlässigen. Folgend werden einige Arbeiten vorgestellt, die die Grundlage für diese Thesis liefert. Andere Arbeiten werden an gegebener Steller in den Folgekapiteln erwähnt.
*Anmerkung: Als Indikator für die Verwendbarkeit im eigenen Projekt wurde der hohe Anteil an realisierten Arbeiten gesehen. Zudem wurden viele Konzepte aus diesen Arbeiten auf professionelle QRTRs übertragen.*

### 2.6.1 A Control Architecture for Quadruped Locomotion over Rough Terrain

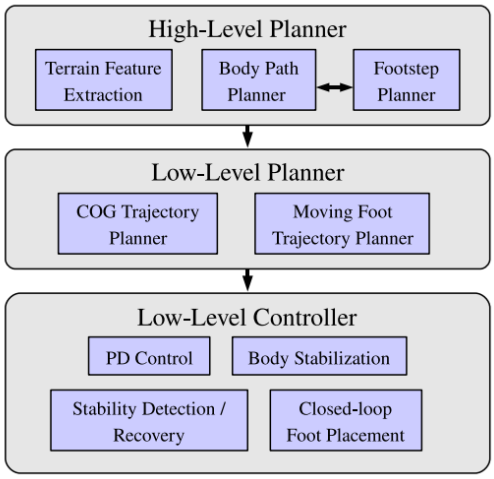


*Abbildung 2-3 Layer-Architektur Kolter [Kol09]*

Die Strategien in dieser Thesis sind von der Arbeit *„A Control Architecture for Quadruped Locomotion Over Rough Terrain“* [Kol09] inspiriert. Das Team stellt in der Arbeit eine komplette Architektur für einen QRTR vor (Abbildung 2-3). Der Kontroller ist ein hybrides System mit drei Schichten. Einen high-level-planner (Deliberative-Layer), der Schritte im Gelände plant und einen low-level-planner (Deliberative-Layer), der dies in einzelne Bewegungen der Beine auflöst. Die Mechanik wird vom low-level-controller (Reactive-Layer) angesteuert, indem er die einzelnen Planungen aus dem höheren Schichten abruft. Für die Planung der Schritte wird die Umwelt in eine Kostenkarte nach Erreichbarkeit der Positionen abstrahiert. Die Arbeit beinhaltet alle Aspekte von der Umweltabstraktion bis zur Beinansteuerung. Diese Arbeit ermöglicht einen Einstieg in das Thema, um die Verfahrenskette in einem QRTR nachzuvollziehen.

### 2.6.2 A Controller for the LittleDog Quadruped Walking on Rough Terrain

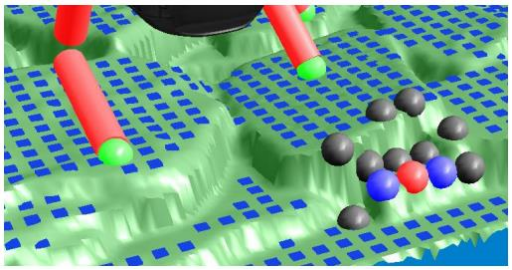

*Abbildung 2-4 Visualisierte Trittpositionen Rebula [Reb08]*

Eine weitere umfassende Arbeit ist von Rebula und einem fünfköpfigem Team, mit dem Titel „A Controller for the LittleDog Quadruped Walking on Rough Terrain“ [Reb08]. Auch hier wird ein hybrides System verwendet. Die Berechnung von alternativen Trittpositionen wird hier detaillierter beschrieben und begrenzt das Verfahren auf drei „beste“ Positionen (Abbildung 2-4). Die Arbeit beschreibt komplexe Verfahren anhand von Pseudocode. Weiterhin ist interessant, dass fast alle Module aus einem hierarchischen Automaten bestehen. Diese Automaten sind ähnlich aufgebaut wie die Beinkontroller (Reactive-Layer) aus der vorausgegangen Bachelorarbeit [Ruh11](2.8.1) zu dieser Thesis. Dabei deutet sich in dieser Arbeit auch an, dass ein statisches Schwerpunktverhalten im Gelände erhebliche Vorteile gegenüber einem dynamischen Verhalten (2.2.2, 3.1) bietet.

### 2.6.3 Stereo Vision and Terrain Modeling for Quadruped Robots

Die zweite Arbeit von Kolter und Team [And09] beschreibt den planenden Layer genauer. Das Team ergänzt in dieser Arbeit den Problemkomplex um den Sichtschatten von Bodenunebenheiten. Alle vorherigen Arbeiten nutzen eine „perfekte Weltsicht“ [And09], da der Sensor zur Umwelterfassung stationär über der Geländeformation montiert ist. In dieser Arbeit ist der Umweltsensor an den LittleDog (2.7.1) montiert. Die vorgestellte Strategie trifft einfach die Annahme, dass die nicht sichtbare Rückseite einer Erhebung der gespiegelten Vorderseite entspricht. Diese Strategie wurde in dieser Thesis nicht berücksichtigt, gab aber einige Hinweise zur Anbindung der Umweltabstraktion (4.3.2.2) an den Deliberative-Layer (4.3.3).

### 2.6.4 Robot Motion for Obstacle Negotiation

Die Arbeit „Robot Motion for Obstacle Negotiation“ [Ani09] von Adukuzhiyil und Team basiert auf den obigen Arbeiten und verwendet ein hybrides System. Der Schwerpunkt dieser Arbeit ist der planende (deliberative) Layer und beschreibt ein Machine-Learning-Verfahren. Dabei werden Trittpositionen auf unterschiedlichen Geländeformationen trainiert. Es konnte gezeigt werden, dass das System bei ähnlichen Geländeformationen richtig reagiert.
Diese Arbeit lieferte die Idee, wie mögliche Schnittstellen modelliert werden müssen, um das Machine-Learning-Verfahren von Bettzüche [Bet14] anbinden zu können.

### 2.6.5 CC-RANSAC: Fitting planes in the presence of multiple surfaces in range data

Diese Arbeit ist kein Teil des Learning Locomotion Programms. Gallo und Team beschreiben in dieser Arbeit [Gal10] eine Abstraktion von einer Punktwolke zu einer Kachelwelt mithilfe des RANSAC-Algorithmus. Das vorgestellte Verfahren ist zwar für die Navigation von radbasierenden Robotern entwickelt, aber sie hat diese Thesis bei der Umweltabstraktion inspiriert. Dies wird im Kapitel 4.1.3 genauer erläutert.

### 2.6.6 Compliant Leg Design for a Quadruped Robot

Auch diese Arbeit stammt nicht aus dem Learning Locomotion Programm. Die Arbeit von Sproewitz [Spr09] und Team beschreibt den mechanischen Aufbau von Beinen, speziell für einen vierbeinigen Läufer. Der Schwerpunkt liegt auf Gewichts- und Kostenersparnis. Obwohl die Beine in dieser Arbeit ein Federungssystem haben, kann sie doch die Annahmen in dieser Thesis zur Beinkonstruktion (3.3.3) untermauern.

### 2.6.7 Quadruped robot obstacle negotiation via reinforcement learning

Der Schwerpunkt dieser Arbeit [HLe06] liegt zwar auf dem Reinforcement Learning, aber es verwendet ein ähnliches Konzept wie in dieser Thesis. Es wird ein hybrides System vorgestellt. Die Umwelt wird in „Boxes“ bzw. Kacheln abstrahiert. Der planende Teil bewertet die Umwelt und die daraus resultierenden Trittpositionen bzw. Robotergliederposen nach „how good“ [HLe06]. Diese Bewertung basiert auf der Erreichbarkeit der Trittpositionen und verfeinert dies bis zu „guten“ Gliederposen.
*Anmerkung: Auf diese Arbeit wird in dieser Thesis leider nicht mehr genau eingegangen, da sie erst kurz vor Abgabeschluss gefunden wurde.*

## 2.7 Ähnliche Roboter

Neben den Robotern der Firma Boston Dynamics® wurden auch die Roboter der ETH-Zürich und vor allem die des DFKI in Bremen betrachtet. Hier stechen besonders die Roboterprojekte ARAMIES [Spe05] und das Projekt iStruct hervor. Diese Roboter haben aber spezielle Konstruktionsmerkmale, die sich tiefgreifend auf die Modellierung des Kontrollerkonzepts auswirken. Deshalb wurde ein Roboter gesucht, der von vielen Arbeiten begleitet wird und einen einfachen Aufbau hat. Der LittleDog erfüllt diese Anforderungen, obwohl er wesentlich weniger Masse aufweist als der Roboter AMEE-XW2. Aus diesem Grund werden keine anderen Roboterplattformen vorgestellt.

### 2.7.1 LittleDog

Der LittleDog ist ein vierbeiniger pseudodynamischer[7] Läufer in der Größe einer Aktentasche und mit einem Gewicht von 3kg (Abbildung 2-5). Er wurde von der Firma Boston Dynamics® für das DAPRA-Programm „Learning Locomotion“ hergestellt. Der LittleDog bildet die Grundlage für den

[7] Durch den hohen Schwerpunkt liegt sein Schwerpunktverhalten größtenteils im dynamischen Bereich.

akademischen Teil des Programms und wird beispielweise von den US-Universitäten *MIT, Stanford, Carnegie Mellon, USC, Univ. Pennsylvania and IHMC* eingesetzt.
Jedes Bein kann über drei Achsen bewegt werden und wird mit elektrischen Servomotoren angetrieben. Der LittleDog verfügt über Sensoren für die Gliederwinkel, das Drehmoment und die Lage im Raum. Weiterhin kann auch der Bodenkontakt ermittelt werden. Laut Herstellerangaben sind die Antriebe schnell und kräftig genug, um ein dynamisches Laufverhalten zu ermöglichen. Die Leistungsfähigkeit des integrierten Rechners kann schwer eingeschätzt werden. Viele Arbeiten benutzen ein Remote-System für den Hauptkontroller und erstellen die Verbindung zum integrierten Low-Level-Controller per WLAN. In der Grundkonfiguration besitzt LittleDog keine Umweltsensoren. Er wird mit einem Framework für das Laufsystem und einer komplexen physikalischen Simulationsplattform geliefert.

*Abbildung 2-5 LittleDog Boston Dynamics [Bos131]*

Der LittleDog ist die verkleinerte und kostengünstigere Variante des BigDog von Boston Dynamics®. Diese Roboter sind wahrscheinlich die prototypischen Testplattformen für den seriennahen LS3. Da die Roboterplattformen BigDog, LS3 und WildCat einen militärischen Hintergrund haben und durch das *Department of Defense* der USA finanziert werden, gibt es nur wenige öffentlich zugängliche Informationen. Aus diesem Grund wird in vielen Arbeiten, wie auch in dieser, oft über Verhaltensweise und die Modellierung spekuliert. Einer der wenigen Möglichkeiten, um Zugang zu den Grundprinzipen dieser Roboter zu erhalten, sind die wissenschaftlichen Veröffentlichungen, die den LittleDog verwenden.
Aufgrund der Exportbeschränkungen, ist es dem AMEE Team leider nicht möglich, einen LittleDog zu erwerben.

## 2.8 Vorarbeiten

Für das Projekt AMEE wurden vor dieser Master-Thesis einige Vorarbeiten erstellt. Diese Arbeiten untersuchen einzelne Aspekte eines vierbeinigen Laufsystems für Roboter. Weiterhin lieferten diese Vorarbeiten die benötigten Daten für das Gesamtkonzept in dieser Thesis. Folgend werden einige verwendete Vorarbeiten kurz vorgestellt.

### 2.8.1 Bachelorarbeit semiautonome Beine


*Abbildung 2-6 Erster Prototyp eines AMEE Beins*

Die vorausgegangene Bachelorarbeit [Ruh11] zu dieser Thesis beschäftigt sich mit den Grundkonzepten eines vierbeinigen Laufsystems. Der Schwerpunkt dieser Arbeit liegt auf den zeitkritischen Abläufen in einem Bein. An einem vollständig realisierten Bein wird gezeigt, dass eine Trennung zwischen harten und weichen Zeitanforderungen möglich ist. Für dieses Bein wurde ein physikalisch gekapselter Beinkontroller auf MCU-Basis entwickelt. Die vorgestellte Implementierung enthält alle Funktionen, um ein Bein mit abstrakten Befehlen zu steuern. Dazu war es nötig, auch die komplexe Berechnung (wie z.B. die inverse Kinematik) in die MCU zu integrieren und die Algorithmen zu optimieren.

Das vorgestellte Bein bildet für diese Thesis die Grundlage für den Reactive-Layer (4.2.3) dieser Thesis. Die Tests in dieser Arbeit zeigten aber auch, dass die verwendete MCU die minimale benötigte Rechenleistung zur Verfügung stellt. Für diese Thesis wird die MCU-Plattform (Beinkontroller 3.4.1) weiterentwickelt und auf eine rechenleistungsstärke Plattform angepasst (5.1.1). Die Arbeit wurde vom Autor dieser Thesis erstellt.

### 2.8.2 Physikalische Simulation der Beine

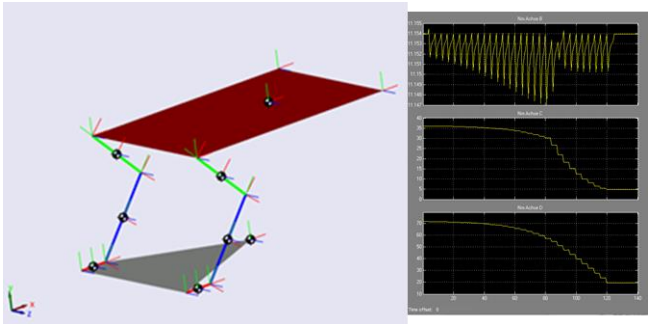

*Abbildung 2-7 Matlab Simulationsmodell*

In der Studienarbeit „Simulation der Drehmomente in einem Roboterbein“ [Ruh12] wurde eine physikalische Simulation der Beine implementiert. Das Ziel der Matlab-Simulation ist die Ermittlung der auftretenden Dreh- und Trägheitsmomente in den Beinantrieben. Das Simulationsmodell bildet das realisierte Bein aus der obigen Bachelorarbeit ab. Die nötigen Gleichungs-systeme wurden hierfür hergeleitet. Für die

auftretenden Trägheitsmomente wurde das Gleichungssystem über den Lagrange-Formalismus und dem Mehrfachpendel hergeleitet. In die Simulation wurden auch Teile der Beinkontrollersoftware übertragen. Das Ergebnis ist ein Simulationssystem, bei dem die Gliederlänge, Gliedermasse und die zu tragende Torsomasse variable eingestellt werden können. In dieser Arbeit zeichnete sich schon ein geringes Trägheitsmoment bei der Konstruktionsart der AMEE-Beine ab. Die Arbeit wurde vom Autor dieser Thesis erstellt.

### 2.8.3 Roboter Eventsystem

Die Bachelorarbeit [Bet10] vom Teamkollegen Bettzüche untersucht die Verwendbarkeit von Microsofts Robotics Development Studio® für ein vierbeiniges Laufsystem. Sie gibt einen umfassenden Überblick zu einer eventbasierten Kontrollerarchitekur in Robotern.

### 2.8.4 Proof of Conecpt Little AMEE

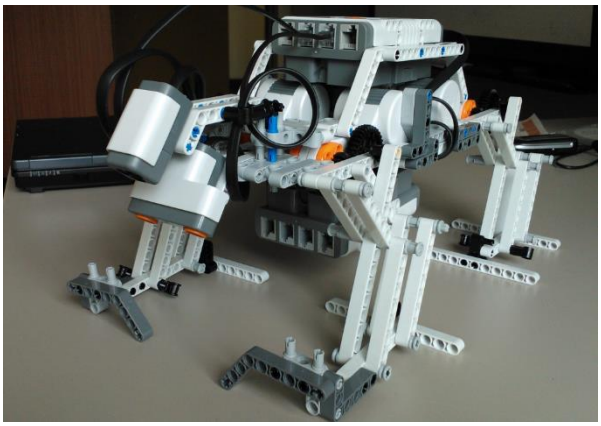

*Abbildung 2-8 Little AMEE*

In dieser Studienarbeit [Ruh121] wurden die Konzepte der gekapselten Beinkontroller (2.8.1) und die eventbasierten Kontroller (2.8.3) [Bet10] miteinander verbunden. Hierfür wurde ein rudimentärer Prototyp des AMEE Roboters mit Lego Technics® realisiert und die beiden Konzepte implementiert. Der Roboter kann die Beine unabhängig voneinander bewegen. Die Beine bewegen sich aber auf mechanisch festgelegten Bahnen. Als Kontroller werden zwei NXT-Bausteine der Firma Lego® genutzt. Das eventbasierte System wird auf einem externen Rechner betrieben und per Bluetooth® mit den NXTs verbunden. Die Tests zeigen eine gute Funktion und bestätigen die Annahmen der Bachelorarbeiten des Teams.

## 2.9 Zusammenfassung

In diesem Kapitel wurden die Grundprobleme beim Laufen in unebenem Gelände aufgezeigt. Ein Grundproblem ist die Positionsbestimmung des Roboters in seiner Umwelt, da beispielweise GPS oder Landmarken kaum genutzt werden können (2.1.1). Um auch das Rutschen der Füße zu erkennen, ist eine leistungsfähige optische Odometrie nötig (2.2.5). Aufgrund von teilweise fehlenden Sensordaten muss das System Annahmen treffen und sein Handeln planen (2.3). In einigen Situationen muss ein QRTR rücksichtslos gegenüber seiner Umwelt entscheiden, damit er durchsetzungsfähig im Gelände bleibt. Es wurde diskutiert, warum diese Anforderungen nicht mit einem einfachen Regelalgorithmus beschrieben werden können.
Um zu ermitteln, wo der Schwerpunkt des Roboters liegen soll, wurden einige Stabilitätstypen (2.2.1)

untersucht. Dafür wurde auch gezeigt, dass der Schwerpunkt des Roboters nicht nur die Konstruktion beeinflusst (3.1), sondern auch die Strategie für das Gleichgewichtssystem (2.2.2). Für die obigen Anforderungen wird ein pseudodynamischer Läufer gewählt, bei dem der statische Bereich dominiert. Zudem wurde festgelegt, dass ein vereinfachtes Gleichgewichtssystem ausreichend ist. Durch die ungenauen Umweltdaten wird eine hybride Softwarearchitektur mit planendem Layer gewählt (2.3.1). Aufgrund dieser Entscheidungen wird das Learning Locomotion Programm mit seinen Arbeiten als Schwerpunkt für einen eigenen Ansatz gewählt. Dazu wurden einige Arbeiten (2.6) mit ähnlichen Anforderungen vorgestellt. Diese Arbeiten haben auch die Vorarbeiten (2.8) im Projekt AMEE beeinflusst.

# 3 Plattform Design

In diesem Kapitel werden die Designentscheidungen für die Mechatronik getroffen. Physikalisch bedingt haben *Quadruped Rough Terrain Robots* hohe Abhängigkeiten zwischen Mechanik und Software. In dieser Thesis sind diese Entscheidungen getrennt dargestellt. In der realen Entwicklung sind diese Entscheidungsprozesse stark verwoben. Die Konstruktion und die Entwicklung der Elektronik sind vom geplanten Softwarekonzept initial getrieben. Beispielweise wird die Positionen der Achsen in den Beinen durch die Implementierung der inversen Kinematik [Ruh11] bestimmt. Hierzu wird im Voraus bestimmt, welche Modellierung zu einer möglichst geringen Laufzeit (Aufwand) führt. Diese Vorgaben werden gegen die Mechanik geprüft und dies ergibt wieder andere mechanisch machbare Bedingungen, die die Softwaremodellierung beeinflussen. Diese Iterationen werden mehrfach durchgeführt und es entsteht eine ungewöhnliche Konstruktion. „Die interdisziplinären Teillösungen sind aneinander gewachsen." Diese agile Entwicklung lässt sich leider kaum textuell darstellen.

*Anmerkung: Das Projekt AMEE ist ein studentisches Projekt und verfügt nur über begrenzte finanzielle Mittel. Das Design ist von diesem Umstand geprägt und daraus resultieren einige suboptimale Designentscheidungen. Der realisierte Roboter wird als „**AMEE-XW2**"[8] bezeichnet und steht als synonym für die Team-Interpretation eines Quadruped Rough Terrain Robots.*

## 3.1 Abstraktion des Laufsystems

Eine stark abstrakte Analogie der Läufertypen stammt aus der Natur: das Raubtier und das Fluchttier. Das Fluchttier entspricht hier dem dynamischen Läufer. Es kann schnell laufen aber nur schlecht klettern (z.B. Rentier). Das Raubtier muss bei der Jagd im Lauf verharren können und zeigt eher ein *pseudodynamisches* Verhalten (Wolf) [Nac10] [Sch96]. Der Roboter AMEE soll Umweltdaten „jagen" bzw. erfassen. Dabei muss er vorsichtig laufen, um nicht umzufallen. Aus dieser Analogie wird auch das Laufsystem aus der Natur abgeleitet und wie in der Bionik [Nac10] abstrahiert.

[8] Bedeutung der Abkürzung AMEE-XW2:
**A**utonomous **M**apping, **E**xploration and **E**vasion - E**x**perimental **W**alker, Version 1.**2**

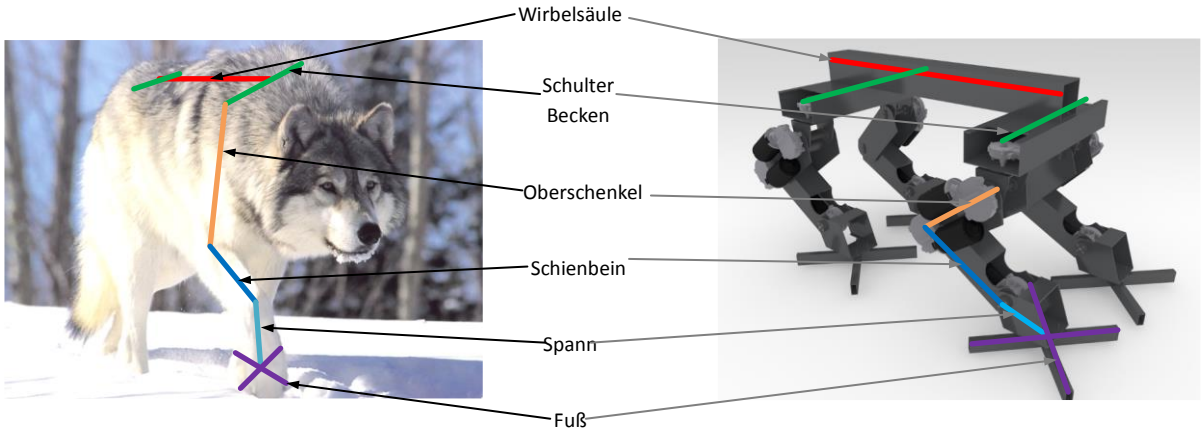


*Abbildung 3-1 Abstraktion des Laufsystems mit menschlichen Entsprechungen (rechts AMEE-XW2)*

Der Roboter AMEE-XW2 ist von Säugetieren inspiriert und wird in Abbildung 3-1 dargestellt. Die Entsprechungen sind nicht in jedem Detail anatomisch korrekt. Beispielsweise sind die Hinterbeine eigentlich verdreht, so dass die „Knie“ zueinander stehen würde. In der Konstruktion ist es aber vorgesehen, dass alle Beine um 180° gedreht werden können. Stark umstritten ist das seitliche Heben des Beins aus dem Schulter- bzw. Hüftgelenk. Um eine zu komplexe Konstruktion zu vermeiden, wurde auf diese Möglichkeit verzichtet. Die Kraftübertragung auf ein Kugelgelenk ist mechanisch kompliziert und anfällig. Wie der LS-3 zeigt (Abbildung 3-2 rechts), kann wahrscheinlich darauf verzichtet werden. Die Nachteile daraus werden in Kapitel 3.3.3 genauer erläutert.

## 3.2 Mechanischer Stabilitätstyp für AMEE-XW2

Für die Ziele (2.4, Punkt b - g) im Projekt AMEE ist ein *pseudodynamischer Läufer*, mit großem statischem Bereich, gut geeignet (2.2.1). Die Entscheidung gegen einen *dynamischen Läufer* gliedert sich in zwei Hauptpunkte (3.2.1, 3.2.2). Im Unterpunkt 3.2.3 wird eine Vermutung, anhand von professionellen QRTRs, erläutert. Die Zusammenfassung (3.2.4) erläutert kurz die Hauptvorteile des pseudodynamischen Läufers.

### 3.2.1 Einsatzzweck

Der Roboter AMEE-XW2 soll nicht springen und schnell laufen können, sondern in schwierigem Gelände gut steigen. Das Ziel aus der einleitenden Vision (1) ist es, sicher und autonom zu laufen und dabei Daten zu sammeln. Diese Anforderungen sprechen für einen *statischen* oder *pseudodynamischen* Läufer, da dies ein reiner *dynamischer* Läufer nicht leisten kann.
Entscheidung gegen einen dynamischen Läufer: In den wissenschaftlichen Arbeiten im Kapitel 2.2.2 wird auf die Notwendigkeit einer Umschaltung zwischen Zero-Moment-Balance (ZMB) Strategie und einer einfachen Gleichgewichtsstrategie hingewiesen, da sich die ZMB-Strategie nicht zum Balancieren auf der Stelle eignet. Auf unebenem Untergrund kommt es oft zu Fehltritten (2.3), was vermutlich bei

einer Umschaltung zwischen den Gleichgewichtsstrategien zu folgenden Problemen führt: Erfolgt der Fehltritt im ZMB Modus (schneller Lauf), wirken noch die Trägheitsmomente auf den Roboter ein. Findet in dieser Situation der betreffende Fuß keinen Halt, müssen die anliegenden Kräfte abgeleitet werden. Damit ist ein Sturz kaum vermeidbar. Vermutlich wird bei professionellen QRTRs ein Sturz sogar gezielt eingeleitet, um die Kräfte kontrolliert abzuleiten. Um dieses Verhaltensmuster zu erproben, ist das Projektumfeld nicht geeignet bzw. es fehlen die Mittel, um mögliche mechanische Schäden zu reparieren. Diese Punkte sprechen für einen *statischen* oder *pseudodynamischen* Läufer. Damit ist auch eine Zero-Moment Strategie in der Modellierung überdimensioniert, da das resultierende Trägheitsmoment zu klein ist (4.1.2).

### 3.2.2 Mechanik – Kosten –Fertigung

Ein *dynamischer* Läufer nutzt das Trägheitsmoment (2.2.2) für ein energieeffizientes Vorankommen. Diese Trägheitsmomente werden beim Auftreten abrupt abgebremst und benötigen eine hochfeste Konstruktion. Einige Roboter verwenden dafür eine Federungssystem [Bos10] in den Schienbeinen. Diese Konstruktion macht den Aufbau erheblich komplexer und erfordert auch spezielle Lager in den Gelenken, die nur schwer selbst gefertigt werden können.

Bei einem dynamischen Läufer wirken sich die auftretenden Trägheitsmomente auch auf die Antriebe aus. Die Antriebe müssen nicht nur die Kräfte halten, sondern zusätzlich eine ausreichende Gegenkraft erzeugen, um sich sofort wieder abzustoßen. Die Folge sind drehmomentstarke und schnell drehende Antriebe in den Beinen, da die nächste Schrittposition in ungefähr 500ms erreicht werden muss [Pop04]. Für relativ günstige Elektroantriebe sind diese beiden Ansprüche ein Widerspruch [Ruh123]. Elektroantriebe, die dies leisten können (beispielsweise aus Industrierobotern[9]), würde das Budget des Projekts AMEE überschreiten.

Professionelle QRTRs setzen deshalb hydraulische Antriebe mit einem Drehmoment von bis zu 5.000Nm pro Glied ein. Um mit einem hydraulisch betriebenen QRTR zu experimentieren, bedarf es auch einer bestimmten Infrastruktur. Zudem steigt auch die Gefahr von Verletzungen, was für das Projektumfeld nicht vertretbar ist.

Aus Sicht der Mechanik und der Kosten ist ein *dynamischer* Läufer unter den Rahmenbedingungen kaum realisierbar. Die obigen Zusammenhänge werden in der vorausgegangen Bachelorarbeit [Ruh11] im Kapitel 2.4.1.ff genauer erläutert. Auch diese Punkte sprechen für einen *statischen* oder *pseudodynamischen* Läufer.

[9] Beispielweise die 1MB High-Speed- Motors Serie der Firma Rexroth (MS 240H: 4.000U/min mit 169Nm)

### 3.2.3 Indiz - Vermutungen

*Abbildung 3-2 (links) Boston Dynamics BigDog [Bos10], (rechts) "AlphaDog" -LS3 [Bos13]*

Einer der ersten seriennahen QRTRs war der sogenannte BigDog der Firma Boston Dynamics® (Abbildung 3-2 links). Er kann schnell laufen, galoppieren und springen, was die Domäne von *dynamischen* Läufern darstellt. Wie einige veröffentliche Videos [Bos131] zeigen, scheint BigDog aber Probleme mit dem Steigen auf schwierigem Gelände zu haben. Für diese Vermutung spricht auch der Nachfolgetyp LS-3 (Abbildung 3-2 rechts), der mehrfach für Tests an die US-Army ausgeliefert wurde [Def14]. Er hat einen niedrigeren Schwerpunkt und einen größeren Abstand zwischen den Füßen. Der LS-3 wirkt eher wie ein *pseudodynamischer* Läufer, was die Designentscheidung für AMEE-XW2 untermauert.

### 3.2.4 Zusammenfassung der Designentscheidung

Zusammenfassend betrachtet wird der Roboter AMEE als pseudodynamischer Läufer, mit einem großen statischen Bereich, ausgelegt. Ein dynamisches Verhalten kann mit langen *dynamischen Schritten* gezielt eingeleitet werden. Damit kann der Roboter AMEE sich in kritischen Situationen und schwierigem Gelände immer auf das statische Verhalten zurückziehen.

**Laufgeschwindigkeit:** Der Roboter AMEE muss sich vorsichtig und langsam bewegen. Für flaches Gelände kann er dynamische *Schritte* mit entsprechendem Laufmuster (4.1.5) nutzen, um schneller zu laufen. Dieser Roboter kann aber nicht die Geschwindigkeiten eines dynamischen Läufers erreichen.

**Statischer Bereich:** Damit der statische und dynamische Bereich nachträglich verändert werden kann, wird der Roboter AMEE mit großen Füßen (Zehen) konstruiert. In der jetzigen Auslegung (3.3.4) ist der statische Bereich so groß, dass das Bein zu ca. 50% ausgestreckt sein muss, um das dynamische Verhalten (kippen) einzuleiten. Durch die einfach konstruierten Füße kann der statische Bereich – durch Kürzen der Zehn - bis auf ca. 15% verkleinert werden.

Damit das Gleichgewicht leichter kontrolliert werden kann, wird auch der Massenschwerpunkt tief gelegt. Die Hauptmasse (4.3.3.3) des Roboters wird von den Antrieben gebildet. Diese Antriebe sitzen nahe dem Boden und direkt in den Beinen. Die Auslegung ist aber eine Abwägung zwischen nutzbarer

Beinlänge und Höhe des Schwerpunkts. Durch diese Maßnahmen wird ein „gutmütiges“ Gleichgewichtsverhalten erreicht und der Roboter kippt langsam.

**Zeitverhalten:** Im statischen Bereich gibt es keine kritischen Zeitanforderungen. Bei dynamischen Schritten sind sie zudem relativ weich. Damit vereinfacht sich die Modellierung des Systems, da ein Überschreiten der maximal geplanten Rechenzeit nicht zwangsläufig zu einem Sturz des Roboters führt. Dies kann natürlich kein Argument für einen seriennahen QRTR sein, bietet aber für einen experimentellen Roboter Vorteile. Bei der Entwicklung kann vorsichtig der dynamische Bereich getestet werden, ohne mechanische Schäden zu riskieren.

**Modellierung:** Die Modellierung des Gleichgewichtssystems (4.3.3.3) vereinfacht sich durch das „gutmütige“ Stabilitätsverhalten. Bei den geplanten Laufgeschwindigkeiten ist das gesamte auftretende Trägheitsmoment zu klein, um einen wirklichen Nutzen zu erzielen. Es wird auf eine komplexe *Zero-Moment* Strategie (2.2.2) verzichtet.

## 3.3 Konstruktion Roboter AMEE-XW2

Für das Team im Projekt AMEE ist es wichtig, eine Roboterplattform zu realisieren, die auch nach dieser Arbeit für weitere Versuche geeignet ist. Deshalb wird eine modulare Konstruktion gewählt. Viele Baugruppen, wie beispielweise die Wirbelsäule oder die Beine, können leicht ausgetauscht oder erweitert werden. Die Konstruktion ist eine Eigenentwicklung und wurde durch einen Maschinenbauer, Sensorhersteller und einen Antriebshersteller unterstützt.

Im Gegensatz zu allen erwerbbaren vierbeinigen Robotern können auch relativ schwere Sensoren (z.B. Kinect V2 mit 1,5kg) direkt am Roboter montiert werden. Weiterhin ist der Roboter AMEE dafür ausgelegt, eine autarke Stromversorgung zu tragen. Um diese Nutzlast zu tragen, sind drehmomentstarke Antriebe nötig, die wiederum die Größe des Roboters beeinflussen. Mit einem MatLab Simulationsmodell (2.8.2) wird das nötige Drehmoment ermittelt. In Zusammenarbeit mit einem Antriebshersteller[10] sind die Motor- und Getriebetypen festgelegt worden.

Als Funktionsmodell dient ein Cinema 4D Dynamics® (C4D) Modell und basiert auf den oben ermittelten Antrieben. Dieses C4D Modell ist wiederum die Grundlage für separate Einzelteilzeichnungen, die zu einem SolidWorks® (CAD-Software) 3D Modell gehören. Das SolidWorks 3D Modell wird für die Fertigung mit einer CNC-Maschine benötigt und besteht aus sog. Baugruppen. Mit diesem Baugruppenmodell wird die endgültige Konstruktion mit allen benötigten Kleinteilen in virtuellen Bewegungen getestet. Hierzu enthält die Baugruppe auch die Sensoren und teilweise die Kabel, die im Roboter verbaut sind. Bei der Konstruktion sind der Kostenfaktor und die einfache Fertigung maßgeblich. Durch die einfachen Einzelteile kann das Projektteam die Bauteile unter Anleitung selbst fertigen.

[10] Das Projekt Team wurde von der Firma Ott GmbH & Co KG in D-78652 Deisslingen unterstützt.

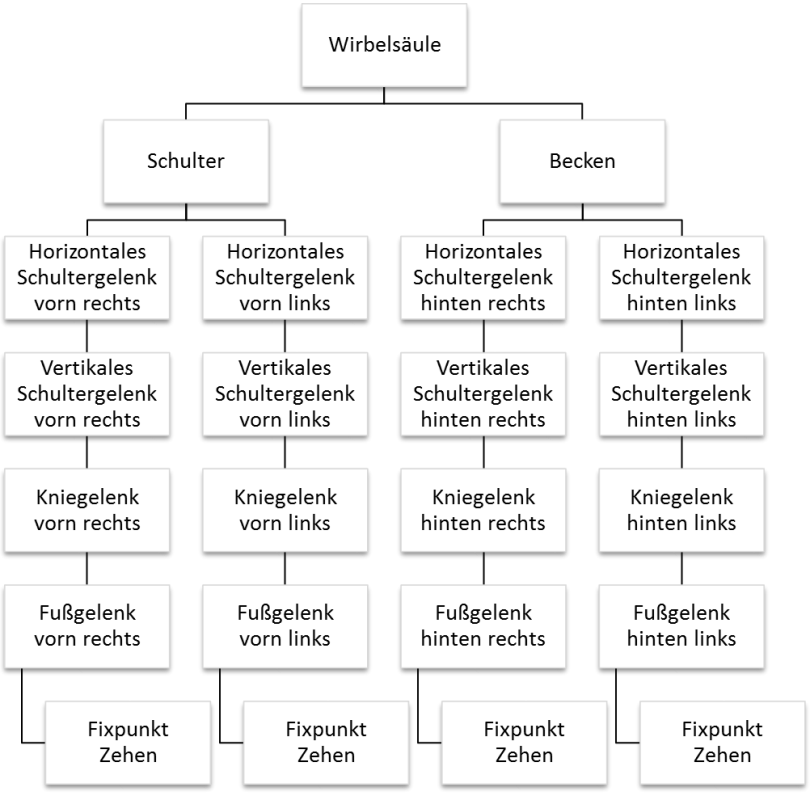


*Abbildung 3-3 Hierarchisches Grundmodell*

Aus einer hierarchischen Modellsicht wird die Mechanik des Roboters als Baum aufgebaut(Abbildung 3-3). Ausgehend von der Wirbelsäule als Wurzel verzweigt das Modell auf die Schulter und das Becken. An diesen befindet sich der Knoten für das horizontal drehbare Schultergelenk und jedes weitere Gelenk darunter. Die Modellstruktur endet in den Zehen als Blätter. Alle weiteren Bauteile werden in diese Hierarchie an entsprechender Stelle eingehängt. Dieses Grundmodell stellte den Ausgangspunkt für alle verwendeten Simulationsmodelle dar. Eine Besonderheit in diesem Modell ist die Mischung aus Gelenken (Joints) und starren Bauteilen als Wurzel und Blätter. Dies ist bedingt durch die Simulationslogik von CAD Programmen und dem sog. Maschinenmodell in MatLab. Vereinfacht betrachtet wird die Wirbelsäule (Wurzel) durch die Fixpunkte (Blätter) bewegt. Die Fixpunkte (Zehen) bewegen sich dabei und haben nicht immer Kontakt zum Boden. Diese Abstraktion ist auch der Ausgangpunkt für das Softwaredesign (4) und erleichtert die Vorstellung von Bewegungsabläufen.

### 3.3.1 Überblick

Das Laufsystem des realisierten vierbeinigen Roboters AMEE-XW2 besitzt 16 angetriebene Gelenke und hat ein Gewicht von ca. 70kg. Zurzeit wird der Roboter von drei 12V Blei-Gel-Akkus angetrieben, bei denen zwei 24V für die Antriebe liefern und einer 12V (22Ah) für die Elektronik liefert. Die robusten Antriebe aus dem Landmaschinenbau liefern in der Spitze ein Drehmoment von 90Nm.

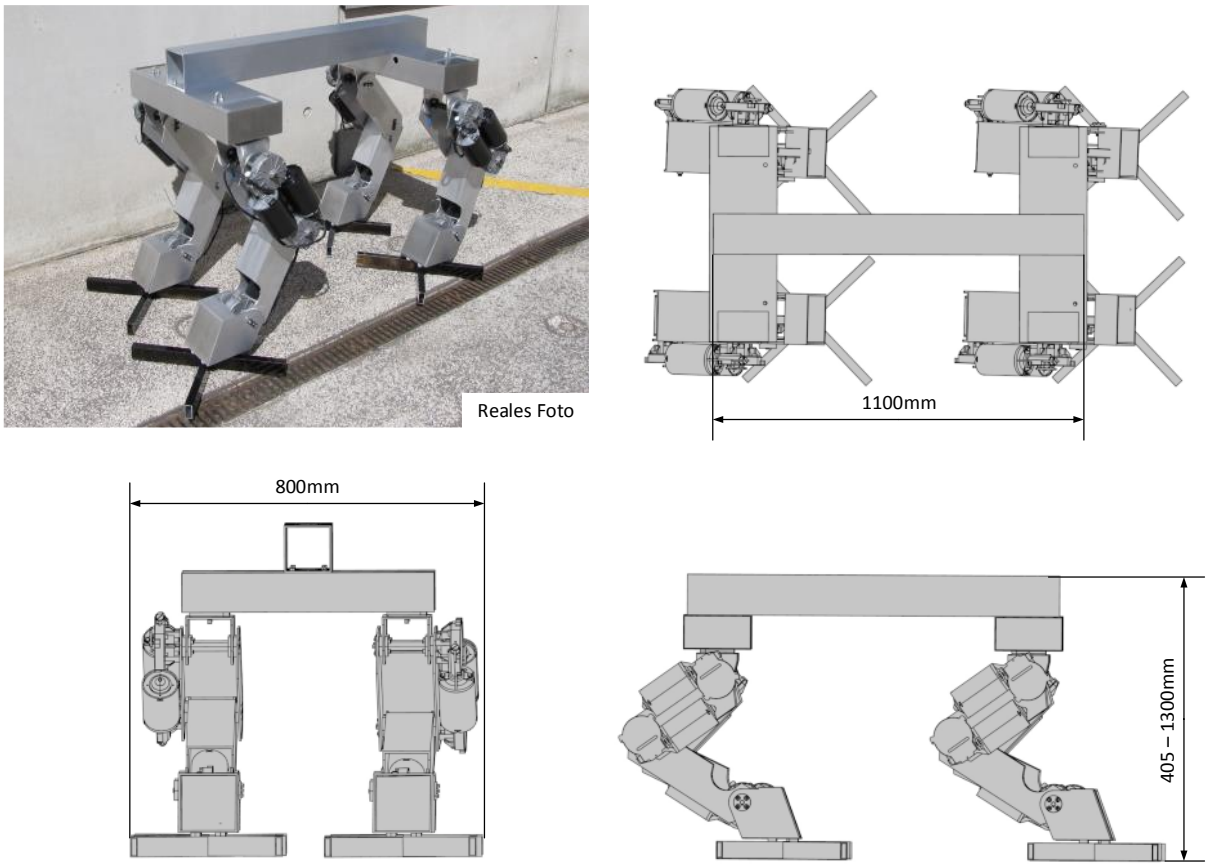


*Abbildung 3-4 Konstruktionsübersicht AMEE-XW2*

Der Roboter AMEE-XW2 besitzt aktiv bewegliche Füße mit jeweils vier Drucksensoren, um die Belastung auf den Untergrund zu ertasten. Die Wirbelsäule dient unter anderem auch als Befestigungspunkt für Sensoren, Stromversorgung und die Hauptrechner. Um dabei möglichst flexibel zu bleiben, hat der Roboter keine Verkleidung des Torsos. Es wäre beispielweise auch ein Benzingenerator als Stromversorgung denkbar. Die Abbildung 3-4 stellt die Konstruktion ohne Elektronik (Umweltsensoren/Hauptrechner) mit den Hauptantrieben dar.

### 3.3.2 Torso

Der Torso des Roboters AMEE-XW2 besteht aus der Wirbelsäule, dem Becken und den Schultern (Abbildung 3-5). Diese sind untereinander starr fixiert. Auf die Wirbelsäule wirken hohe drehende Kräfte, wenn der Roboter auf zwei diagonal gegenüberliegenden Füßen steht (Fast-Walk 4.1.5).

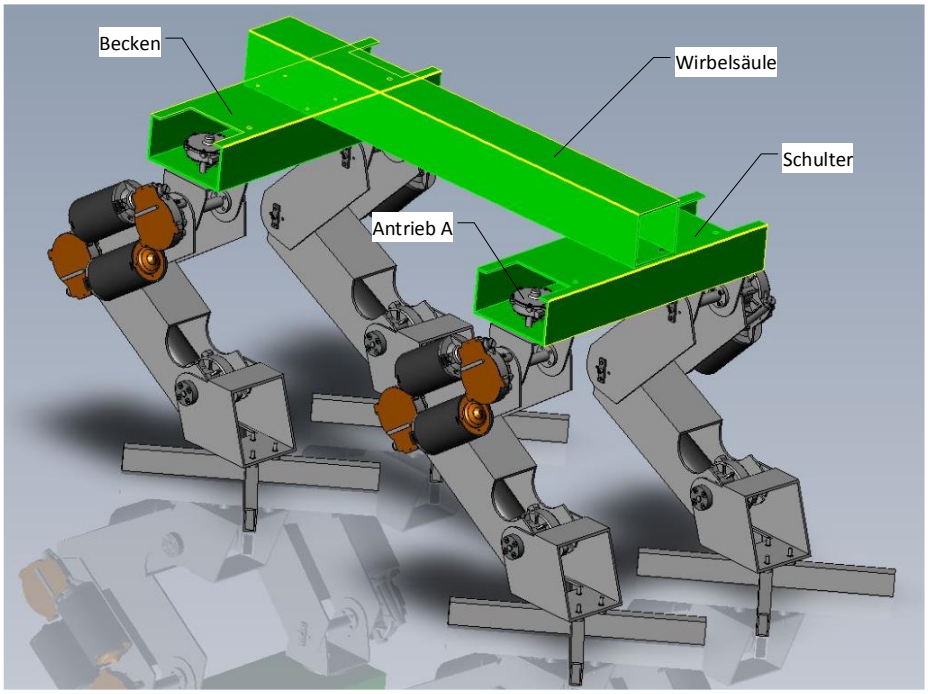


*Abbildung 3-5 Torso AMEE-XW2*

**Wirbelsäule:** Die Wirbelsäule ist zurzeit ein starres Vierkantprofil und hält das Becken und die Schultern zusammen und dient auch als Kabelführung, Befestigungspunkt für die Sensoren, die Hauptrechner und der Stromversorgung. Sie ist leicht austauschbar und kann mit einer aktiven beweglichen Wirbelsäule erweitert werden, wie im DFKI Projekt *iStruct* [Dan12].

**Schultern und Becken**: Die Schulter und das Becken sind identisch aufgebaut. In ihnen sind die Antriebe A (Abbildung 3-5), um die Beine horizontal zu drehen. Sie haben große Ausschnitte, die als Wartungsklappe für die Leistungselektronik dienen (3.4.1).

### 3.3.3 Beine

Die Beine des Roboters AMEE-XW2 basieren auf der vorausgegangen Bachelorarbeit (2.8.1) [Ruh11] und sind in einigen Details verbessert. Die Glieder werden direkt an den Achsen mit Gleichstrom-getriebemotoren (3.3.4) angetrieben. Als Basis dienen Vierkant- und Rechteckprofile, die mit Hilfe einer CNC-Fräse bearbeitet sind. Die Gliederachsen sind aus massiven Rundstählen gedreht, da sie die Kräfte von den Antrieb auf die Glieder übertragen.

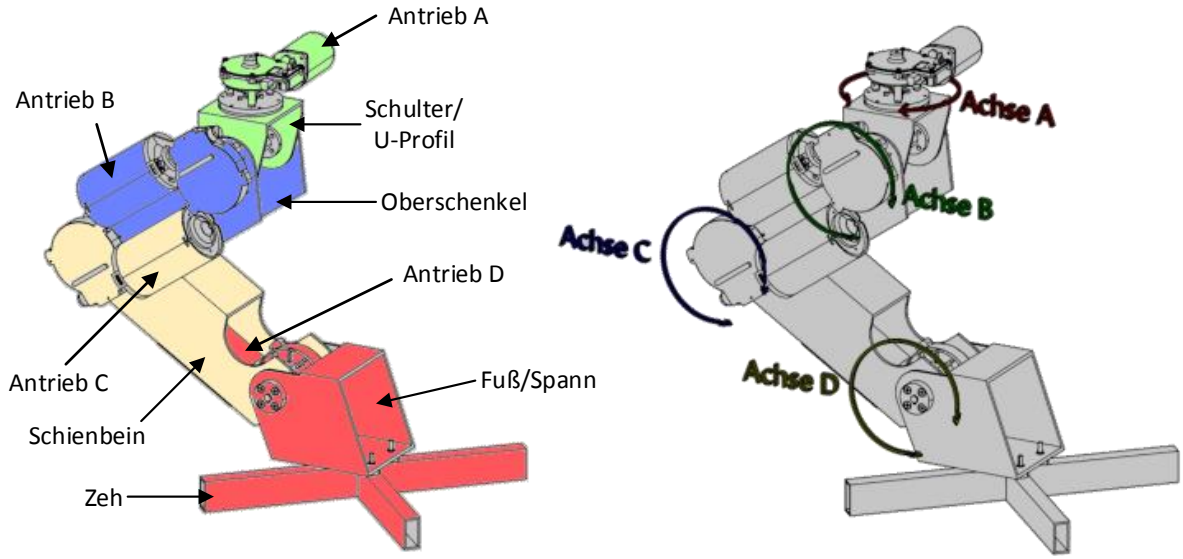


*Abbildung 3-6 Bein AMEE-XW2*

Jedes Bein ist ein 4-DOF System mit einem halbkugelförmigen Workspace unterhalb des Torsos. Die Beine können 360° Winkelgrad horizontal um die Achse A (Abbildung 3-6) gedreht werden. Dieser Drehwinkel wird durch die Verkabelung und die Logik des Beinkontrollers auf 180° beschränkt, wobei jeweils 90° zur Seite geschwenkt werden kann. Die Achse B ist der vertikale Teil des Schultergelenks und ist unter dem Torso um 180° schenkbar. Das Schienbein kann um Achse C geschwenkt werden. Der Bewegungsfreiraum ist auf ca. 170° begrenzt und kann von einem ausgestreckten Bein bis zur sog. Parkposition gefahren werden. In der Parkposition (7.3) greift das Schienbein mit seinem Ausschnitt über die Achse B. Der Fuß kann um 270° über die Achse D kann gedreht werden.

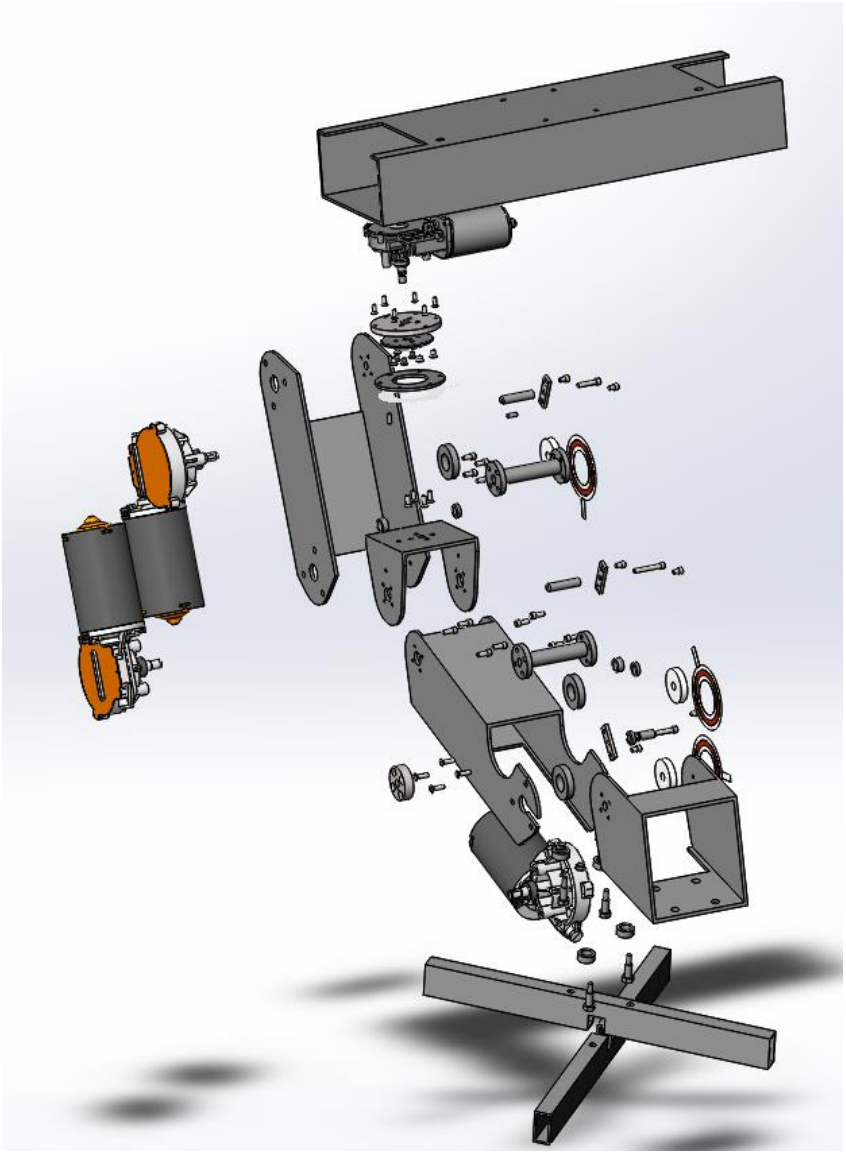

*Abbildung 3-7 Explosionsdarstellung eines Beins*

Die diversen Einzelteile werden in Abbildung 3-7 dargestellt und geben einen kleinen Einblick in die vielen Einzelentscheidungen bei der Entwicklung. Eines der Konstruktionskriterien ist aber die Implementierung der Beinkontroller (4.2.3).

**Schulter**: Die Schultergelenke sind in zwei Achsen unterteilt, da die Konstruktion eines Kugelgelenks – wie in der Natur – zu aufwendig für das kleine Projekt Team ist. Die Schwierigkeit bei einem Kugelgelenk besteht in der Kraftübertragung. Ein Kugelgelenk hat keine feste Drehachse und die Kräfte müssten über beispielweise Drahtseile auf die Glieder übertragen werden. Durch diesen Umstand wird auf die Möglichkeit des seitlichen Hebens des Beins verzichtet, da auch andere QRTRs darauf verzichten (3.2.3).

Das Bein wird horizontal über ein massives Gleitlager gedreht. Das Gleitlager trägt die Hebelkräfte der gesamten Beinlänge (Achse A). In diesem Gleitlager sind auch die Winkelsensoren (3.4.3.3) integriert (Abbildung 3-8). Die Gleitlager sind fest mit der Schulter und dem Becken verschraubt. Auf der drehbaren Seite des Gleitlagers ist ein U-Profil befestig. Die Antriebsachse A (Schulter A) wird durch eine Bohrung (durch das Gleitlager) zu diesem U-Profil geführt. Diese Achse ist nur mit dem U-Profil fixiert, womit die Drehbewegung übertragen wird.

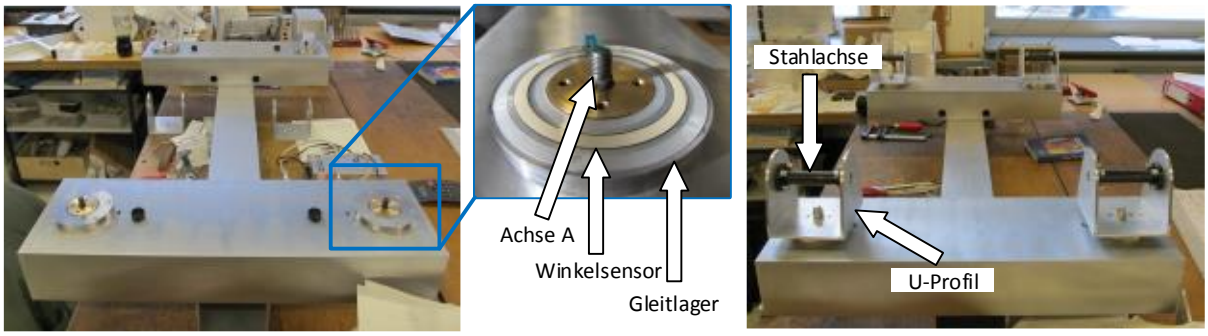


*Abbildung 3-8 Unterseite des Torsos*

Mit der zweiten Achse (B) des Schultergelenks kann das Bein vertikal geschwenkt / gehoben werden. Hierfür ist eine massive Stahlachse quer im U-Profil fixiert. Die Stahlachse verfügt über eine Aufnahme (sog. Nut Abbildung 3-9) für die Achse des Antriebs (B) am Oberschenkel. Das Antriebsgehäuse (B) ist mit dem Oberschenkel verschraubt und die Achse des Antriebs B wird durch eine Bohrung durch ihn geführt. Der Oberschenkel ist im Querschnitt breiter als das U-Profil und greift über das U-Profil der Schulter (Abbildung 3-9 rechts). Die Achse von Antrieb B wird in die Stahlachse gesteckt. Dadurch wird die Kraft der Drehbewegung, über eine sog. Passfeder, auf die Stahlachse übertragen. In dieses Gelenk ist wiederum ein Winkelsensor (3.4.3.3) integriert.

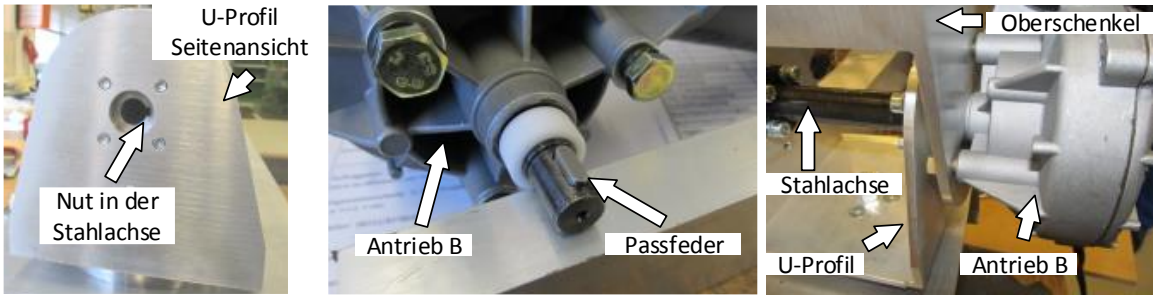


*Abbildung 3-9 Schulter und Oberschenkel*

**Knie:** Der Antrieb C für das Kniegelenk ist auch auf dem Oberschenkel fest montiert und die Antriebsache ragt wieder durch eine Bohrung auf die Innenseite des Oberschenkels. Das Schienbein hat einen kleineren Querschnitt als der Oberschenkel und der Oberschenkel greift über das Schienbein (Abbildung 3-10). Im Schienbein ist die Stahlachse (Achse C) verschraubt und die Antriebsachse C greift

(über Nut und Passfeder) in diese Stahlachse. Über der Achse C befindet sich ein Winkelsensor. Im Schienbein ist zudem der Antrieb D für den Fuß integriert.

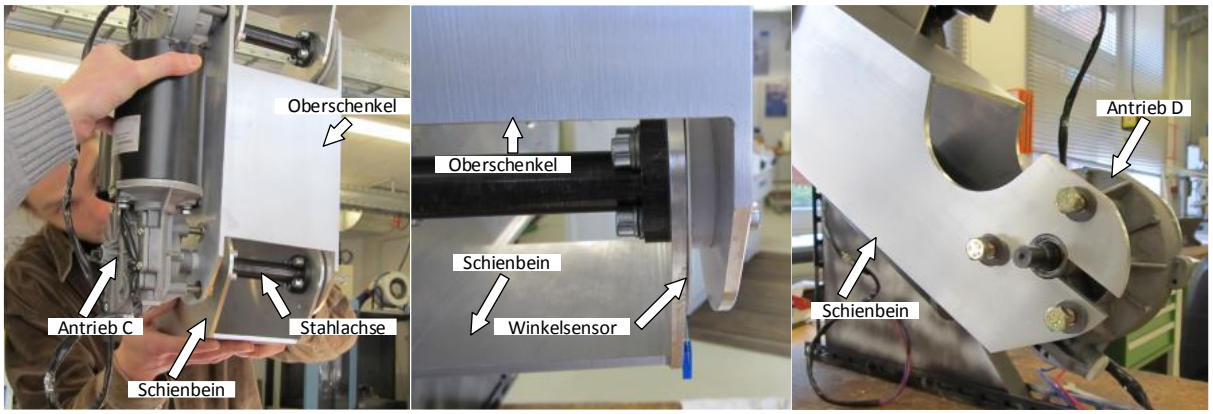


*Abbildung 3-10 Knie und Schienbein*

### 3.3.4 Füße

Der Fuß bzw. Spann wird vom Antrieb D (Abbildung 3-6) im Schienbein bewegt. Im Gegensatz zur Vorarbeit (2.8.1) wird eine andere Zehkonstruktion verwendet. Die Zehen bilden ein starres Kreuz aus Rechteckprofilen und sind am Spann beweglich gelagert, um den Druck auf die Zehen messen zu können. Dazu sind am Boden des Spanns vier Bolzen montiert mit denen das Zehkreuz gehalten wird (Abbildung 3-11). Auf jedem Bolzen steckt eine Druckfeder, die das Zehkreuz vom Spannboden drückt. Damit ist das Zehkreuz zwar fest montiert, kann aber gegen den Spannboden gedrückt werden. Diese Konstruktion ist für die Drucksensoren (3.4.3.4) in den Füßen nötig.

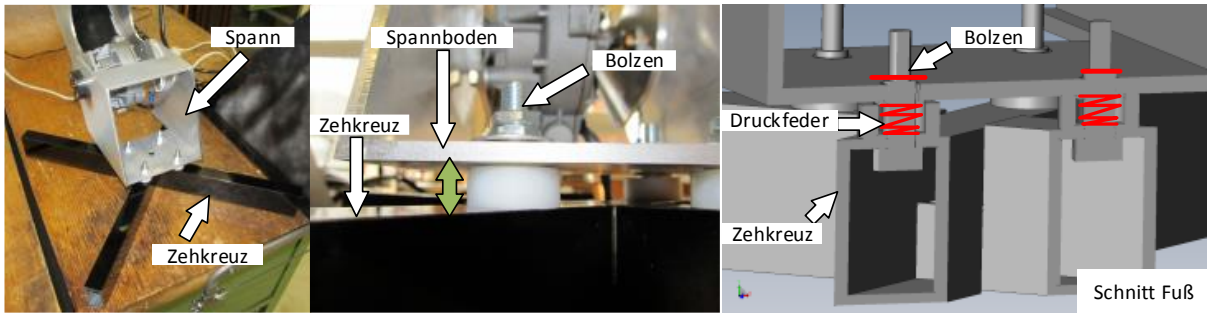


*Abbildung 3-11 Spann und Zehen*

Um zu verhindern, dass die Drucksensoren durch das Eigengewicht des Zehkreuzes falsch ausgelöst werden, sind die Federn mit jeweils ca. 1,625kg vorgespannt. Die Drucksensoren sprechen damit erst bei 6,5kg (ca. 10% der Robotermasse) Belastung auf einem Fuß an. Diese Konstruktion ermöglicht es, den Belastungsvektor auf die Zehen sicher zu ermitteln. Eine direkte Messung der Kräfte in Einheiten pro Zeh ist damit nicht möglich, da die Zehen zwar individuell aufgehängt sind, aber untereinander verbunden sind.

*Abbildung 3-12 Der Roboter Lemur IIB mit neuartiger Fußkonstruktion, hängend unter der Decke [Par12]*

Die Zehkonstruktion ist ein Kompromiss aus Konstruktionsaufwand und Funktion. Die Zehen sind nicht ideal, da sie sich nicht dem Untergrund anpassen und damit leicht abrutschen können. Sie können sich nur um ca. 2mm bewegen. Ein ideale Konstruktion wären Zehen, die unabhängig um 30-80mm beweglich wären und durch die Software versteift werden könnten. Tritt der Fuß mit dieser Konstruktion auf unebenen Boden auf, würden die Zehen nachgeben und sich dem Untergrund anpassen. Wäre die Trittposition eingenommen, würden die Zehen versteift werden und könnten voll belastet werden, um den Massenschwerpunkt zusätzlich abzustützen. Eine ähnliche Konstruktion wird von von Aaron Parness (Jet Propulsion Laboratory) [Par12] vorgestellt. Dabei handelt es sich um fast einhundert kleine Anker, die sich zusätzlich auf dem Untergrund verhaken. Diese Anker / Zehen können versteift werden und sind eigentlich als Bohrhilfe für extraterrestrische Roboter entwickelt worden. Die Entwickler haben aber auch das Potential als Roboterfuß erkannt und gaben 2012 die Weiterentwicklung bekannt: „Future efforts are focused on using the Lemur IIB robot to perform climbing demonstrations on vertical and inverted natural rock surfaces.“ [Par12]

### 3.3.5 Antriebe

Die Simulation zur Ermittlung der Drehmomente in den Beinen eines vierbeinigen Roboters (2.8.2) ergibt Drehmomentspitzen von 70Nm. Eine Vorgabe für diesen Roboter ist, dass das System keine Totpunkte hat. D.h. es soll keine Gliederstellung geben, aus der sich der Roboter nicht befreien kann. Die ermittelten Daten sind die Entscheidungsbasis für die gewählten Motor-Getriebeeinheiten.
Für die Schultern (Achse A, Abbildung 3-6 ) wird ein relativ leichter Antrieb mit 1,7kg Gesamtgewicht und 83Nm verwendet, da er über dem Masseschwerpunkt des Roboters liegt. Der Antrieb ist ein 24V Gleichstromantrieb mit einem Kunststoffgetriebe (Tabelle 3-1 ②). Laut Datenblatt des Herstellers Valeo® kann er das maximale Drehmoment aber nur kurzzeitig liefern, da die Kühlung des Motors nicht

für einen Dauerbetrieb ausgelegt ist. Dies ist aber für die horizontale Beindrehung ausreichend.
Die anderen Achsen sind mit Antrieben ausgestattet, die im Dauerbetrieb 90Nm leisten und ein mechanisch belastbareres Getriebe bieten (Tabelle 3-1 ①). Auch hier handelt es sich um einen Gleichstromgetriebemotor mit 24V (27A) und einem Gewicht von 4,3kg (Tabelle 3-1 ②).

| Antriebstyp | XDW092001-01 ① | 404360 ② |
|---|---|---|
| Anlaufdrehmoment | 90Nm | 83Nm (kurzzeitig) |
| Nenndrehmoment | 20Nm | 8Nm |
| max. Drehzahl | 27 U/min | 48 U/min |
| Nennspannung | 24V | 24V |
| Gewicht | 4,3 kg | 1,7kg |

*Tabelle 3-1 Verwendete Antriebstypen*

Die Antriebe treiben direkt die Achsen in den Gliedern an und müssen die gesamte Robotermasse tragen. Dadurch werden die Getriebe und die Lager stärker belastet, als vom Motorhersteller vorgesehen. Aus diesem Grund werden die überdimensionierten 90Nm Antriebe in den Gliedern eingesetzt.

### 3.3.6 Stützgestell

*Abbildung 3-13 Stützgestell "AMEE-Walker"*

Für die Montage und Laufversuche des Roboters wird ein Stützgestell verwendet[11]. In dieses Gestell kann der Roboter AMEE-XW2 „schwebend“ eingehängt werden. Für erste Versuche wird der Roboter mit einer Vierpunktaufhängung an den Schultern aufgehängt (Abbildung 3-13). In weiteren Experimenten wird der Roboter auch über einen mittigen Punkt aufgehängt und das Stützgestell verhindert nur noch einen Sturz des Roboters. Das Stützgestell trägt ca. 200kg (ermittelt durch einen Belastungstest) und ist mit Rollen ausgestattet, damit es bei Laufversuchen dem Roboter folgt. Leider ermöglicht das Gestell keinen Kurvenlauf des Roboters, da die Füße auf die seitlichen Träger treten. Von einer Verbreiterung des Gestells wurde abgesehen, da dieses Gestell im Labor kaum handhabbar ist. Für einen Weiterentwicklung des Roboters AMEE muss eine Umgebung gesucht werden in der auch größere Stützgestelle handhabbar sind.

[11] Das Stützgestell ist ein Geschenk der HAW-Hamburg-Zentralwerkstatt an das Projekt-Team. Es wurde dort, in Anlehnung an eine Babylaufhilfe, als „AMEE-Walker“ tituliert.

## 3.4 Elektronik

Der Roboter AMEE ist eine vollständige Eigenentwicklung. Die entwickelten Konzepte sind zwar an einige Grundkonzepte (2.2.1, 2.3) angelehnt, unterscheiden sich aber von anderen Systemen (4.1.4). Die vorausgegangene Bachelorarbeit (2.8.1) [Ruh11] zeigt die Vorteile von externen Beinkontrollern auf und wird auch in diesem Roboter realisiert. Für die Leistungselektronik werden keine fertigen Modellbaulösungen verwendet, da die Stromaufnahme der Antriebe dafür zu hoch ist (ca. 27A pro Antrieb). Es werden auch alternative Bussysteme für einen Roboter verwendet. In diesem Konzept kommunizieren alle Komponenten per Ethernet (kein Realtime-Ethernet) miteinander. Durch diese Randanforderungen ist die Elektronik aus Einzelkomponenten zusammengestellt.

### 3.4.1 Beinkontroller

Die vier Beinkontroller kontrollieren und koordinieren die Antriebe in jeweils einem Bein und wurden in der Bachelorarbeit (2.8.1) [Ruh11] bereits ausführlich vorgestellt. An dieser Stelle werden nur einige Veränderungen erläutert. In der Bachelorarbeit wird ermittelt, dass die dort verwendeten 8-Bit ATMEL MCUs mit 18MHz die untere Leistungsgrenze für eine autonome Beinsteuerung darstellt. Die Antriebe im Roboter AMEE-XW2 (3.3.5) haben eine höhere Drehzahl, als in der Vorarbeit. Die neu entwickelten Beinkontroller haben eine geringere Reaktionszeit verwenden eine leistungsstärkere MCU Plattform. Eine schematische Übersicht der Beinkontroller ist in Abbildung 3-14 dargestellt

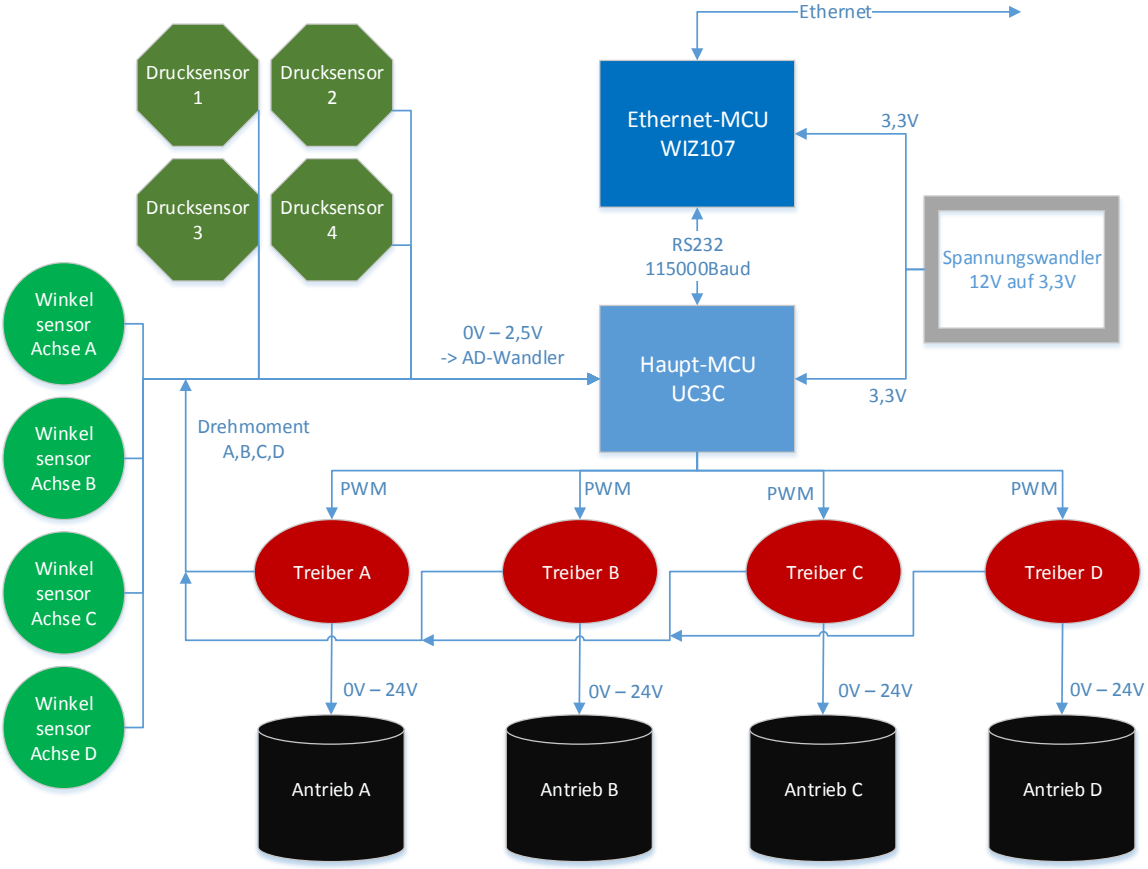


*Abbildung 3-14 Schema-Beinkontroller*

**Haupt-MCU:** Die Beinkontroller verwenden eine 32-Bit ATMEL® MCU AT32UC3C0512C (UC3C) mit 66MHz. Sie bietet für fast jede Schnittstelle eine komplett gekapselte und vom MCU-Kern unabhängige Hardware. Zudem hat sie eine FPU mit DSP Befehlssatz. Diese MCU eignet sich laut Hersteller ATMEL für eine rechenintensive Maschinensteuerung und basiert auf einem ARM® Prozessor. Für den Roboter AMEE sind auch die vollständig getrennten 16 x 20Bit Analog-Digital Wandler und die vier 12Bit PWM Wandler ideal. Als Basis für eine eigene Beinkontrollerplatine, dient das Modul AL-UC3CRAM der Firma ALVIDI. Auf diesem Modul sind alle nötigen Komponenten integriert, die für einen rudimentären Betrieb nötig sind. Um die Schnittstellen leicht zugänglich zu halten sind bei diesem Modul alle Schnittstellen auf Stiftleisten geführt.

**Ethernet-MCU:** Als Bussystem wird für den Roboter AMEE Ethernet und das Transmission Control Protocol (TCP IPv4 lokal) verwendet. Hierfür wird ein sog. TCP-IP-Hardwarestack der Firma WIZNET® verwendet und das Modul WIZ107 ist an die Haupt-MCU angebunden. Die Verbindung erfolgt per RS232 mit 115.000 Baud. Das Modul WIZ107 bietet eine programmierbare 20MHz MCU und wird als TCP-Nachrichtenfilter (mit einer Whitelist) verwendet. In der Whitelist sind IP-Header-Zeichensequenzen hinterlegt. Enthält der Header einer IP-Nachricht nicht die Zeichenfolge „$:X:“ wird

die Nachricht verworfen. Gleiches gilt für Nachrichten, die von der Haupt-MCU an die Ethernet-MCU per RS232 gesendet werden. Diese Maßnahmen stellen einen Schutz vor sog. „bubbling idiots" dar. Zudem soll verhindert werden, dass mögliche Bonjour-Nachrichten des Windows®-Hauptkontrollers (3.4.2) die Reaktionszeit der Haupt-MCU einschränken, da diese sonst aktiv bearbeitet werden müssten. In dieser Hardwarekombination kann eine Nachricht ohne Eingriff der Haupt-MCU empfangen oder gesendet werden. Die Entscheidung für die Verwendung der internen RS232 Schnittstelle ist durch die völlig unabhängige serielle Einheit in der Haupt-MCU gegeben. Die Nachrichten werden ohne Interrupt (per DMA) in den SRAM der MCU geschrieben und es erfolgt nur ein Softevent, wenn dieser Vorgang abgeschlossen ist.
Die Haupt-MCU bietet auch eine interne Ethernetschnittstelle an, aber keinen Hardware TCP-IP Stack, wie das WIZ107 Modul.

**Treiber:** Um die Antriebe mit bis zu 24V anzusteuern, werden jeweils vier Kompakttreiberchips pro Beinkontroller verwendet. Zudem muss ein Treiber ca. 30A regeln und thermisch ableiten können. Für diese Aufgabe wird der Einchiptreiber VNH5019A-E der Firma STMicroelectronics® verwendet. Er bietet einen thermischen Überlastungsschutz und eine sog. Motor-Monitor, der die anliegende Last am Motor zurückgibt. Dabei wird die Stromaufnahme als analoge Spannung (0V – 5V) über einen gesonderten Pin ausgegeben. Diese Spannung wird, über einen Spannungsteiler, an einen AD-Wandler der Haupt-MCU geleitet und ermöglicht dadurch logische Rückschlüsse über einen Bewegungsablauf. Gesteuert werden die Treiber mit zwei digitalen Pins (Recht- Linkslauf) und einem PWM-Signal mit 7,5kHz. Diese Pins sind auch direkt an die Haupt-MCU angeschlossen.
Durch die thermische Ableitung (ca. 600W) dieser Chips ist ein besonderes Platinenlayout nötig. Aus diesem Grund wird das gleichnamige Modul VNH5019 der Firma Pololu® verwendet. Weiterhin bietet dieses Modul auch eine Schutzschaltung gegen zurückfließende Spannungen, die auftreten, wenn der Motor ohne Last nachläuft und wie ein Generator eine Spannung erzeugt. Zudem sind die Pololu-Module mit extra gefertigten sog. Spacern (Alu-Blöcke zur Wärmeübertragung) mit Silberoxid-Epoxidharz in die Schultern geklebt. Die dickwandigen Schultern werden als Kühlkörper genutzt und bieten eine Kühlfläche von ca. 400cm².

**Anbindung Sensoren:** Alle verwendeten Sensoren liefern eine Spannung oder arbeiten wie ein variabler Widerstand. Die AD-Wandler der Haupt-MCU arbeiten mit einer Referenzspannung von maximal 2,5V bei einer MCU-Betriebsspannung von 3,3V. Hierfür sind gesonderte und gefilterte Spannungsteile in die Trägerplatine integriert, um die Winkel- und Drucksensoren mit Spannung zu versorgen.

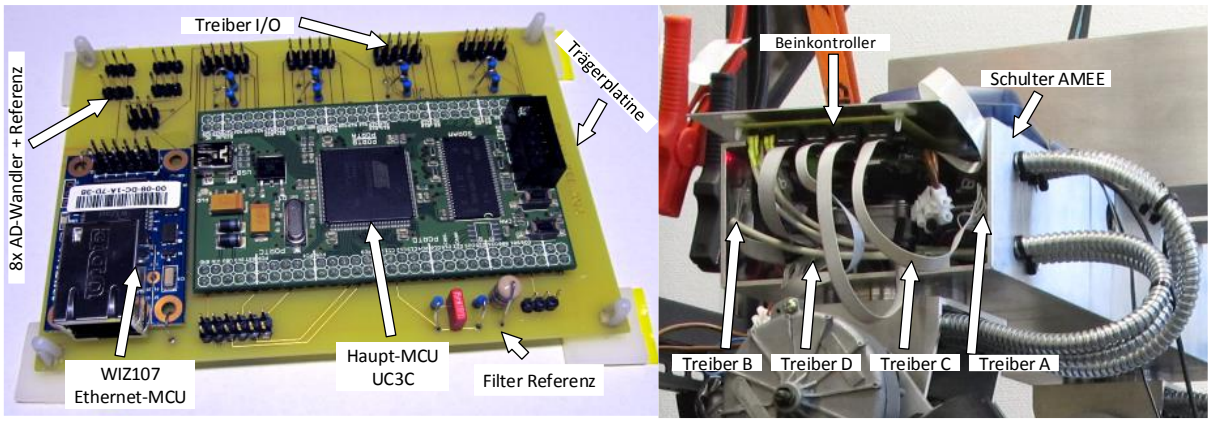


*Abbildung 3-15 Beinkontroller*

Die linke Abbildung 3-15 stellt einen der vier Beinkontroller ohne Verkabelung dar. Auf dem linken Bild ist der Kontroller in die Schulter integriert und vollständig verkabelt. Die Treiber sind nach ihrer erwarteten thermischen Abgabe auf den Boden und die Seiten in die Schultern geklebt. Die Treiber mit der höchsten thermischen Abgabe sind Treiber B und C und nicht nebeneinander angeordnet.

### 3.4.2 Hauptkontroller

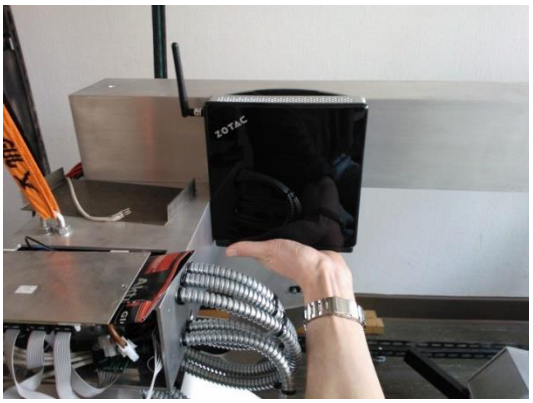


*Abbildung 3-16 Hauptkontroller ZBOX-Plus*

Der Hauptkontroller (die Hauptrechner) ist für die höhere Logik (4.3.2) und die Planung (4.3.3) vorgesehen. In diesem Roboter ist einer der rechenintensivsten Aufgabe die Umwelterfassung und Umweltabstraktion (4.3.2.2). Dazu wird das Microsoft® Framework für die Kinect® V2 genutzt. Die verwendeten Verfahren skalieren über die Anzahl der Prozessoren. Deshalb werden Hauptrechner verwendet, die viele Prozessoren enthalten und mobil sind.

Für diese Anforderungen werden zwei ZOTAC® ZBOX IQ01 Plus verwendet (Abbildung 3-16). Diese Rechner bieten eine Intel i7 4770T® CPU (2,5GHz – 3,7GHz) mit jeweils vier physikalischen bzw. acht logischen Kernen. Zudem bieten die Systeme native USB 3.0 Schnittstellen, was für den Betrieb der Kinect V2 zwingend notwendig ist. Ein kompletter Rechner hat eine Größe von 188mm x 188mm x 51mm. Die beiden Hauptrechner sind an der Wirbelsäule des Roboters montiert.

### 3.4.3 Sensoren

Die Sensoren im Roboter AMEE sind aus dem Low-Cost Bereich und nicht für extreme Umwelteinflüsse geeignet (ausgenommen der Winkelsensoren). Sie sind nach ihrer Abstraktionsebene (4.2.2) an die entsprechenden Kontroller angeschlossen.

### 3.4.3.1 Kinect for Windows V2

*Abbildung 3-17 Montierter Kinect V2 Sensor*

Das AMEE Projektteam arbeitet (unter NDA) mit einer Alpha-Version des Kinect for Windows® V2 (K4W2) Sensors. Er wird im Roboter AMEE für die Umwelterfassung verwendet und starr an der vorderen Wirbelsäule montiert. Für das in dieser Arbeit vorgestellte Verfahren wird nur das Tiefenbild des Sensors verwendet. Das Tiefenbild besteht aus 512 x 424 Messpunkten und wird mit einem verbesserten *time-of-flight* Verfahren erzeugt. Der vertikale und horizontale Erfassungswinkel des Tiefenbildes beträgt ca. 179° Winkelgrad. Microsoft® verzichtet in dieser Version auf einen Motor, um die Kinect zu schwenken. In der verwendeten Version ist der Erfassungsbereich zwischen 0,5m und 4,5m.

### 3.4.3.2 Lagesensor

Als Lagesensor wird ein MPU-9150 Baustein der Firma InvenSense® verwendet. Dabei handelt es sich um eine sog. 9-Achsen (Gyroskop + Beschleunigungsmesser + Kompass) Sensorkombination. Dieser Baustein hat einen integrierten DMP® (Digital Motion Processor), der über Sensorfusion die Einzelsensoren kapselt. Die ausgegeben Lagedaten sind die verrechneten Daten der Einzelsensoren. Dieser Baustein ist über einen ATMEL® ATMega328 an ein weiteres WIZ107 Ethernet Modul angeschlossen. Durch eine modifizierte Programmierung sendet diese Modulkombination nur Lagedaten per Ethernet an den Hauptkontroller, wenn sich die Lage des Moduls im Raum geändert hat. Um das interne Bussystem (Ethernet) des Roboters nicht mit Nachrichten zu fluten, wird bei kontinuierlichen Veränderungen nur alle 200ms eine Nachricht gesendet.

### 3.4.3.3 Winkelsensoren

Die Winkelsensoren der Firma Hoffmann und Krippner® sind eine Spezialanfertigung für den Roboter AMEE-XW2 und wurden kostenlos von der Firma zur Verfügung gestellt. Dabei handelt es sich um Sensofoil® Folienpotentiometer. Wird auf den Sensorring punktueller Druck von 1-3N ausgeübt, verändert der Sensor den Widerstand entsprechend der Druckposition auf dem Ring. Der Druckpunkt wird über einen elektrisch passiven „Schleifer“ ausgeübt (Abbildung 3-18). Die Sensoren haben einen Durchmesser von 95mm und sind nur 2mm dick. Die gesamte Beinkonstruktion ist erst durch diese Sensoren so kompakt zu realisieren. Da die Winkelsensoren wie ein Drehwiderstand arbeiten, ist nur ein einmaliges Kalibrieren nötig.

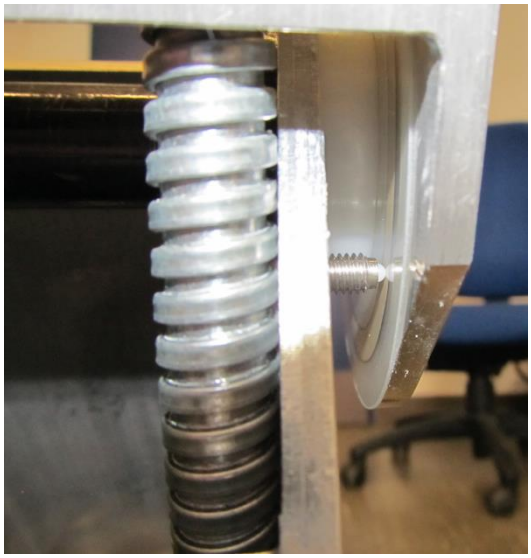

*Abbildung 3-18 Winkelsensor mit "Schleifer" zwischen Knie und Schienbein*

Eine Besonderheit ist der erweiterte Temperatur-bereich von -40°C bis 55°C und der Erfassungswinkel von 350° Winkelgrad bei einer Wiederholgenauigkeit von unter 0,5%.

### 3.4.3.4 Drucksensoren

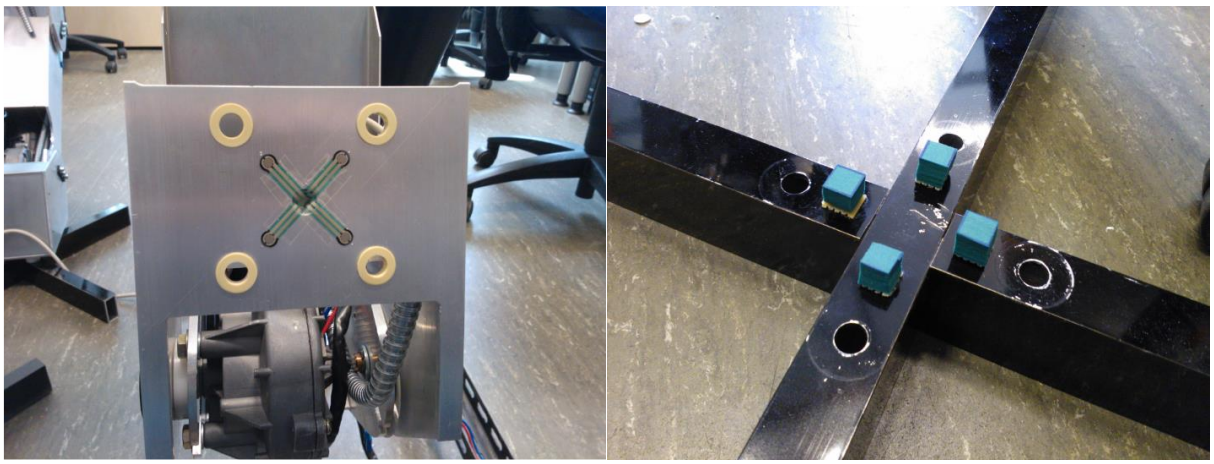

*Abbildung 3-19 (links) Drucksensoren auf der Unterseite des Spann, (rechts) Gummiüberträger auf dem Zehkreuz*

In den Füßen des Roboters sind jeweils vier Drucksensoren verbaut (3.3.4). Die Sensoren sind vom Typ FSR 400 der Firma Interlink Electronics®. Sie basieren wie die Winkelsensoren auf „leitfähigen Kunstoffen“ und bestehen aus drei Folien. Wird ein Druck auf den Sensor ausgeübt, werden mit steigendem Druck mehr „Leiterbahnen“ kurzgeschlossen und dies resultiert in einer Widerstandsveränderung. Die Sensoren detektieren einen Druck von 0,2N bis 100N. Bedingt durch den

Maximaldruck auf den Sensor wird die Kraft über Gummiblöcke von Zehkreuz auf die Sensoren übertragen, die auf der Unterseite des Spanns geklebt sind (Abbildung 3-19). Die Gummiblöcke schwächen die Kraft auf die Sensoren ab.

## 3.5 Zusammenfassung

In diesem Kapitel sind einige Designentscheidungen zur Mechatronik für den Roboter AMEE-XW2 getroffen worden, die sich direkt auf die Softwarearchitektur im nächsten Kapitel auswirken. Der Roboter ist ein pseudodynamischer Läufer mit großem statischem Bereich (3.2.1). Jedes Bein bildet ein 4-DOF System ab (3.3.3). Mit der Festlegung auf die Antriebstypen ist der Roboter kräftig aber langsam (3.3.5). Durch die externen Beinkontroller kann der reaktive Teil direkt in ihnen implementiert werden (3.4.1). Die Rechenleistung der neuen MCUs ist überdimensioniert und bietet Spielraum für zukünftige Erweiterungen. Die Sensoren bilden die Grundausstattung für einen QRTR (3.4.3).
Der Roboter ist mechanisch robust, aber erfüllt nicht ganz die Ansprüche an einen QRTR. Dies ist bedingt durch den fehlenden Spritzwasserschutz der Elektronik. Bis auf die Hauptrechner und die Kinect V2 ist der Roboter auf ein Abdichten der Systeme vorbereitet.

# 4 Software Design

Dieses Kapitel bildet den Schwerpunkt dieser Thesis. Es werden das Softwaredesign und die Systemarchitektur erläutert. Da diese Thesis einen Ansatz für ein komplettes Systemdesign vorstellt und diskutiert, können nicht alle Aspekte in Einzelkapitel unterteilt werden. Die Einzelbetrachtungen wären zu stark fragmentiert und der Zusammenhang wäre nur schwer ersichtlich. Aus diesem Grund werden sekundäre aber wichtige Abhängigkeiten in den Unterkapiteln erläutert.
Der folgende Ansatz verwendet einige Techniken, um das Echtzeitverhalten eines Laufsystems zu kapseln und auszulagern. Trotzdem hat dieses System immer noch zeitkritische Anforderungen und das Design ist Laufzeitabwägungen unterworfen.
Die verwendeten Strategien und Verfahren werden kurz in Kapitel 4.1 vorgestellt. Im Kapitel 4.1.1 werden einige Designalternativen im Vergleich zu dieser Thesis erläutert. Ein Systemüberblick der Architektur wird in Kapitel 4.2 vorgestellt. Im Kapitel 4.3 werden die Verfahren konkret beschreiben und über praktische Beispiele erläutert. Durch die Vielfalt der verwendeten Verfahren werden auch zukünftige Möglichkeiten für eine Erweiterung an gegebener Stelle angerissen.

## 4.1 Strategien und Verfahren

Um einen Überblick über die getroffenen Designentscheidungen zu ermöglichen, werden in diesem Unterkapitel kurz die angewendeten Strategien und Verfahren vorgestellt. Diese werden im Kapitel 4.3ff konkretisiert und eine gesamte Architektur im Detail erläutert.

### 4.1.1 Designalternativen

Die abstrakte Quintessenz aller aufgezeigten Probleme ist: Laufen mobile Roboter in unebenem Gelände, müssen sie entscheiden, wohin sie ihre Füße setzen können. Ob eine Trittposition geeignet ist, hängt von mehreren Faktoren ab. In den folgenden Absätzen wird kurz das Vorgehen in dieser Thesis von anderen Arbeiten abgegrenzt.
**Umwelterfassung**: Ein hybrides System, mit planenden Eigenschaften, sollte die Umwelt möglichst genau erfassen können. Zudem wäre es wünschenswert, dass der Erfassungsbereich der Umwelt möglichst groß ist. Durch die konstruktiven Einschränkungen (z.B. Sichtschatten, Sensorreichweite), das Einsatzszenario (z.B. Wald) und die mobil verfügbare Rechenleistung müssen die erfassten Daten in eine abstrakte Form überführt werden. Alle vorgestellten Arbeiten (Kapitel 2.6) abstrahieren die Sensordaten zu einer Punktwolke der Umwelt. Auch im Projekt AMEE wird die Umwelt durch eine Punktwolke abgebildet.
**Umweltabstraktion**: Die Repräsentation der Umwelt durch eine Punktwolke ist aber nicht direkt von einer Maschine verwendbar. Für das System stellt sich noch die Frage, wie das System die Punktwolke interpretiert. Für ein Laufsystem ist die primäre Abstraktion die Trittposition.
In den Arbeiten von Rebula und Team [Reb08] repräsentieren die Punktwolkenpunkte mögliche Trittpositionen. Dabei werden nur Punktwolkenpunkte betrachtet, die auf einen festen Rasterpunkt

(5mm) treffen. Dieses Raster ist eine zweidimensionale Draufsicht der erfassten Umwelt, bei dem der Rasterpunkt die Höheninformation hält. Danach werden Rasterpunkte aussortiert, die nicht erreicht werden können.
Ähnlich geht auch das Team von Kolter und Team [Kol09] vor. Sie erzeugen aus der Punktwolke ein strukturiertes und reguläres Polygonnetz (tri Mesh) mit festem Punktabstand. Für die Bewertung von Trittpositionen wird die Steigung der einzelnen Vektoren im Polygonnetz betrachtet. Dadurch ist es dem Team möglich, zur Höhenkarte auch eine Kollisionskarte zu erstellen. Für die Kollisionskarte wird eine mögliche Beinstellung auf der Höhenkarte gegen Kollisionen mit dem Boden geprüft, um zu verhindern, dass beispielsweise ein „Knie“ mit einer Bodenkante kollidiert.
Die Folgearbeit von Kolter und Team [And09] verwendet kein festes Raster für das Polygonnetz. Ein Polygonnetzpunkt repräsentiert einen Punkt aus der Punktwolke und eine mögliche Trittposition.
Einen völlig anderen Weg geht das Team von Adukuzhiyil [Ani09]. Aus einer Punktwolke wird eine Höhenkarte erstellt. In dieser Höhenkarte werden markante Geländeformationen gesucht und mit dem Roboter trainiert. Die eigentliche Trittposition besteht aus dem gelernten Wissen aus dieser Geländeformation.
Diese kleine Auswahl von Beispielen bezieht sich direkt auf den vierbeinige Roboter LittleDog (2.7.1). Die verwendeten Roboter haben aber eine punktuelle Auflagefläche pro Fuß (Kugelfüße). Für den Roboter AMEE (und in dieser Thesis) wird eine große Auflagefläche pro Fuß genutzt. Dadurch dass die Auflagefläche quadratisch ist, liegt die Abstraktion zu quadratischen Flächen (Kacheln / Tiles) nahe. Im Gegensatz zu anderen Arbeiten besteht die Welt damit aus quadratischen Kacheln, die die Trittposition repräsentiert (4.1.3). Die Höhenkarte wird in dieser Thesis auch *nicht* binär bewertet (Trittposition: Ja/Nein), sondern erhält eine gewichtete Sicht nach Erreichbarkeit und ähnelt damit einer Kostenkarte (4.3.3.1.1). Alle Flächen, die nicht annähernd horizontal sind, werden ignoriert.
**Pfadplanung**: Die Pfadplanung in dieser Thesis und in den oben vorgestellten Arbeiten plant nur fünf bis zehn Schrittfolgen und wird deshalb in dieser Arbeit als *Laufrichtungsplanung* (4.3.3.1) bezeichnet. Viele Arbeiten kommen zu dem Schluss, dass eine weiterreichende Pfadplanung kaum einen Nutzen erzielt. Dies wird oft mit dem Sichtschatten und der abnehmenden Auflösung der Sensoren begründet, da beides mit zunehmender Entfernungen zu bzw. abnimmt. Alle Arbeiten teilen die Planung in mehrere Bereiche auf. Kolter [Kol09] und Rebula [Reb08] betrachten zwar alle erfassten Trittpositionen, untersuchen aber beispielweise nur die Höhenunterschiede in der unmittelbaren Nachbarschaft des Roboters. Begründet wird dieses Vorgehen mit der Reduzierung des Rechenaufwands.
In dieser Thesis wird ein mehrstufiges Verfahren (4.3.3) vorgestellt. In der ersten Stufe wird die Umwelt in grobe Kacheln eingeteilt. Anhand dieser groben Höhenkarte werden geradlinige Pfade gesucht. Diese Pfade werden dann auf Höhenunterschiede (4.3.3.1.1), Kopffreiheit und andere Hindernisse (4.3.3.1.2) geprüft. Anhand dieser Prüfung wird eine Laufrichtung ausgewählt (4.3.3.1). In der nächsten Stufe werden dann nur noch Kacheln untersucht, die aus der jetzigen Situation in Laufrichtung liegen und auch real erreichbar sind. Dazu wird die Auflösung der Kachelwelt erhöht und genauer untersucht (4.3.3.2). Das komplette Verfahren wird nach jedem Schritt wiederholt.

**Stabilitätsplanung**: Kolter [Kol09] und Rebula [Reb08] kommen auch zu dem Schluss, dass ein dynamisches Gleichgewichtsverhalten in stark unebenem Gelände nachteilig ist. Wie diese Thesis verzichten auch sie auf eine Zero-Moment Strategie (2.2.2). Kolter und Rebula (und viel andere) verzichten dabei sogar auf dynamische Schritte und beschränken den Roboter auf das statische Verhalten. Dafür haben sie die bekannten Verfahren erweitert, um beispielweise im statischen Bereich auf Hindernisse springen zu können. Diese Sprungtechnik berücksichtigt nicht das Trägheitsmoment und funktioniert nur unter bestimmten Bedingungen.
In dieser Thesis wird zwar die Implementierung von dynamischen Schritten und ein Umschaltkriterium (4.3.3.3) – zwischen statischen und dynamischen Schritten – vorgestellt, aber aufgrund des Gewichts von AMEE und des Umfeldes im Projekt AMEE wird diese Sprungtechnik nicht betrachtet.
**Heartbeat / Taktung**: Der Kontroller von Kolter und Team [Kol09] und andere [Reb08] stoßen einzelne Module mit unterschiedlicher Taktung an. Ist das betreffende Modul abgearbeitet, wartet es inaktiv auf den nächsten Takt. Damit wird eine Überreaktion des Systems verhindert.
In dieser Thesis wird der Takt als Heartbeat (4.3.5) bezeichnet und wird mit einem Watchdog erweitert, um Dead- und Livelocks zu erkennen. Auch hier wird der Takt verwendet, um eine Überreaktion zu verhindern. Zudem wird er für dynamische Schritte verwendet (4.3.3.3).
**Hard- / Softrealtime**: Diese Thesis untersucht auch die Abstraktion von Reflexen und die zeitlichen Anforderungen in einem Laufsystem (4.1.4). Dadurch können Echtzeitanforderungen in physikalisch externe Kontroller gekapselt werden (4.2.3). Die verbleibenden Teile des Kontrollers haben nur noch weiche Zeitanforderungen und es kann im realen Hauptkontroller auf ein Realtime-OS verzichtet werden. Damit wird es möglich, nicht echtzeitfähige Frameworks zu nutzen.

### 4.1.2 Gleichgewichtsstrategie

Der Roboter AMEE ist ein *pseudodynamischer* Läufer und verwendet eine einfache planende Stabilitätsberechnung (4.3.3.3). Dieser Läufertyp hat zwei unterschiedliche Verhaltensweisen in der Stabilitätsbetrachtung (Kapitel 2.2.1). In einigen Arbeiten (4.1.1) wird auf eine Berechnung der Trägheitsmomente verzichtet. Ein interessanter Punkt ist vereinfacht ausgedrückt: Wenn der Roboter schon in die Laufrichtung kippt, wird dies bis zu einem kritischen Punkt ignoriert. Diese drastische Vorgehensweise ist nur ab vier Beinen möglich und benötigt eine gleichmäßige Massenverteilung im Roboter. In Kombination mit anderen einfachen Verfahren (Stützpolygon) und einer hohen Wiederholrate des Verfahrens zeigen sich Charakteristika einer Moment-Balance-Strategie. Die Schlussfolgerung aus diesen Arbeiten ist ein ähnliches Vorgehen für den Roboter AMEE zu nutzen und das Stabilitätspolygon mit einer Hülle in Laufrichtung zu erweitern (4.3.3.3).

### 4.1.3 Abstraktion der Umwelt

In dieser Thesis ist die Welt für das Laufsystem in Kacheln bzw. Trittpositionen unterteilt. Diese Abstraktion wird aufgrund seiner simplen Regeln gewählt. Einige Projekte verwenden diese Abstraktion für die Verwendung in Fahrzeugen [Gal10] [Fai11] oder beabsichtigen eine genaue aber vereinfachte Abbildung der Umwelt [Jea10].

Die Annahme in dieser Master-Thesis geht davon aus, dass ein pseudodynamischer Läufer mit großen Füßen besser im Gelände steigen kann als ein dynamischer Läufer (3.2.4). Für die Stabilitäts-betrachtung ist damit *nicht* nur die Trittposition als Punkt (wie z.B. bei LittleDog(2.7.1)) relevant, sondern auch die Fläche, auf die der Fuß gesetzt wird. Als Schlussfolgerung daraus wird die erfasste Umwelt in Trittpositionsflächen vereinfacht. Weiterhin werden alle Flächen ausgeblendet, die nicht annähernd horizontal sind. Durch diese Vereinfachung der Umwelt kann bei der Schrittplanung (4.3.3) mit einfachen Datenstrukturen und geringer Datenmenge gearbeitet werden. Beispielsweise wird eine dreidimensionale (2.5D) Punktwolke mit 210.000 Punkten in ca. 120 dreidimensionale Flächen abstrahiert und ermöglicht erst das Verfahren zur Stabilitätsberechnung in Kapitel 4.3.3.3.
In der Arbeit „*CC-RANSAC: Fitting planes in the presence of mulitple surfaces in range data*“ [Gal10] wird ein Verfahren zur Abstraktion einer Punktwolke zu einer „Kachelwelt“ vorgestellt. Dafür wird eine kartesische dreidimensionale Punktwolke in kleinere Punktwolken unterteilt. Die erfasste Umwelt wird in kleinere Subwürfel zerlegt, wobei die Kantenlänge der Subwürfel einer Kachel (Tile) entspricht. In jedem dieser Subwürfel wird mit dem RANSAC-Plane Algorithmus nach einer Fläche gesucht. In der oben erwähnten Arbeit wird pro Subwürfel nach drei Flächen gesucht, um z.B. auch Absätze (Bordsteine) korrekt abzubilden. Damit können auch geschlossene Pfade über Schrägen für radbasierende Fahrzeuge geplant werden. Ein Laufsystem kann über kleine Absätze und Lücken im Pfad steigen und benötigt keinen geschlossenen Pfad, um einem Pfad zu folgen.
Für einen schreitenden Roboter scheint es ausreichend zu sein, nur eine horizontale Fläche pro Subwürfel zu betrachten. Bedingt durch das iterative Verhalten des RANSAC Algorithmus wird mit hoher Wahrscheinlichkeit die Fläche mit den meisten Punkten gefunden. Da die Messpunkte zur Erzeugung der Punktwolke[12] homogen verteilt sind, kann auch davon ausgegangen werden, dass die meisten Messpunkte die größte Fläche darstellen. Wird die Kachelgröße bzw. Subwürfelkantenlänge so groß wie die Fußauflagefläche gewählt, sollte nur die größte und annähernd horizontale erkannte Fläche von Interesse sein. Dieses Vorgehen ist eine Abwägung zwischen Laufzeit und Genauigkeit. Aufgrund der hohen Datenmenge einer Punktwolke werden hier Ungenauigkeiten zugunsten der Laufzeiten gewählt. Diese Designentscheidung begründet sich auch durch die Funktion der reaktiven Schicht (4.3.4), die ständig eine „Endkontrolle“ der Bewegungen durchführt. Eine Fehlplanung wirkt sich nicht durch einen Sturz des Roboters aus, sondern nur durch eine Verlangsamung des Laufens.

### 4.1.4 Abstraktion von Reflexen – zeitkritische Abläufe

Einige wissenschaftliche Arbeiten aus dem Bereich der Gleichgewichtssysteme für laufende Roboter kommen zu dem Schluss, dass bei Menschen und Tieren viele Reaktionen aus dem zentralen Nervensystem erfolgen [Pop05]. Diese Reaktionen und auch Reflexe sind von höheren Hirnfunktionen nur schwer beeinflussbar und laufen unbewusst ab. Die meisten wissenschaftlichen Arbeiten identifizieren dabei immer zeitkritische Reaktionen des Körpers. Eine Abstraktion sind biologisch

[12] Diese Annahme gilt nur für diese Implementierung und mit einem *Kinect for Windows V2®* Sensor (3.4.3.1). Für andere Sensoren und Verfahren ist dies **nicht** allgemeingültig.

inspirierte Kontroller unter Echtzeitbedingungen. Dies lässt den Umkehrschluss zu, dass viele Funktionen eines Laufsystems nicht unbedingt zeitkritisch sind. Es scheint eine „natürliche“ Trennung zwischen zeitkritischen und *nicht* zeitkritischen Kontrollern zu geben. Die Untersuchungen aus den vorangegangen Bachelorarbeiten [Ruh11] [Bet10] abstrahieren dies in gekapselte Module, die nur über kurze semantische Nachrichten kommunizieren. Durch diese Abstraktion kann auch die Modellierung klarer nach Zeitanforderungen gegliedert werden.
Wird der gesamte Roboter unter diesen Hypothesen betrachtet, kann das zeitkritische Verhalten jedes einzelnen Beins von anderen Funktionen getrennt werden (4.2). Dieser Zusammenhang wurde in der vorangegangen Bachelorarbeit [Ruh11] erfolgreich getestet.

### 4.1.5 Schrittmusterauswahl

Unter der Schrittmusterauswahl wird hier **nicht** das Laufmuster (Walking Gait 2.2.3) verstanden. In diesem Ansatz ergibt sich das Laufmuster dynamisch[13] durch das Vorgehen in der Stabilitätsplanung (4.3.3.3). D.h. es gibt keine feste Reihenfolge, welcher Fuß als nächstes gewählt wird. Das Synonym *Schrittmuster(-auswahl)* wird hier für die Anzahl der Füße benutzt, die auf dem Boden stehen und den stabilen Bereich bilden (4.3.3.3). Für diesen Roboter gibt es zwei Schrittmuster. Der sogenannte *Fast Walk* und der *Save Walk*.
Der *Fast Walk* (Passgang) behält zwei diagonal gegenüberliegende Füße auf dem Boden. Die gebildeten Beinpaare arbeiten dabei gleichzeitig/parallel. Das Beinpaar am Boden schiebt bzw. zieht den Robotertorso in Laufrichtung (Body Shift). Gleichzeitig bewegen sich die Beine ohne Bodenkontakt zur nächsten Position. Danach beginnt der Ablauf von vorn, aber mit vertauschten Beinpaaren. Beim Fast Walk ist der stabile Bereich (Stützpolygon) kleiner, da sich die Fläche nur über zwei Füße bildet. Dieser Modus wird für schnelleres Laufen auf relativ glattem Untergrund benutzt.
Der *Save Walk* (creeping gaint) behält bei der Bewegung des Robotertorsos (Body-Shift) alle Füße auf dem Boden. Ist die schiebende bzw. ziehende Bewegung abgeschlossen, wird nur ein Fuß zur Zeit in die vorgreifende Position bewegt. Sind alle vier Füße in der vorgreifenden Position, beginnt der Ablauf erneut. Durch diese Einschränkung läuft der Roboter langsam, aber die Fläche des stabilen Bereichs (Stützpolygon) wird durch mindestens eine Dreipunktauflage gebildet. Diese Fläche ist erheblich größer als beim Fast-Walk. Dieser Modus wird für unebenes Gelände genutzt und ermöglicht auch ein Laufen auf rutschigem Untergrund (z.B. Kies). Bedingt durch die dynamischen Laufmuster werden manchmal nicht alle Füße in die vorgreifende Position bewegt. Es werden sogenannte Ausfallschritte ausgeführt, um den stabilen Bereich zu vergrößern. Dies ist einer der Vorteile eines dynamischen Laufmusters.
*Mehr Informationen zu den Schrittmustern wurden in der vorausgegangen Bachelorarbeit* [Ruh11] *ab Kapitel 2.5.1ff erläutert.*

[13] Ein dynamisches Laufmuster ist nicht zu verwechseln mit dynamischen Schritten (2.2.3) des pseudodynamischen Läufers. Das dynamische *Laufmuster* ist unabhängig vom Läufertyp.

## 4.2 Systemüberblick

Die grobe Systemarchitektur steht in enger Beziehung zum Plattformdesign der Elektronik. Werden die zeitkritischen Aufgaben vom Rest des Systems (4.1.4) getrennt, kann jedes Bein als eigenständiges Gerät modelliert werden. Das System kann in einen Hauptkontroller und in vier Beinkontroller zerlegt werden.

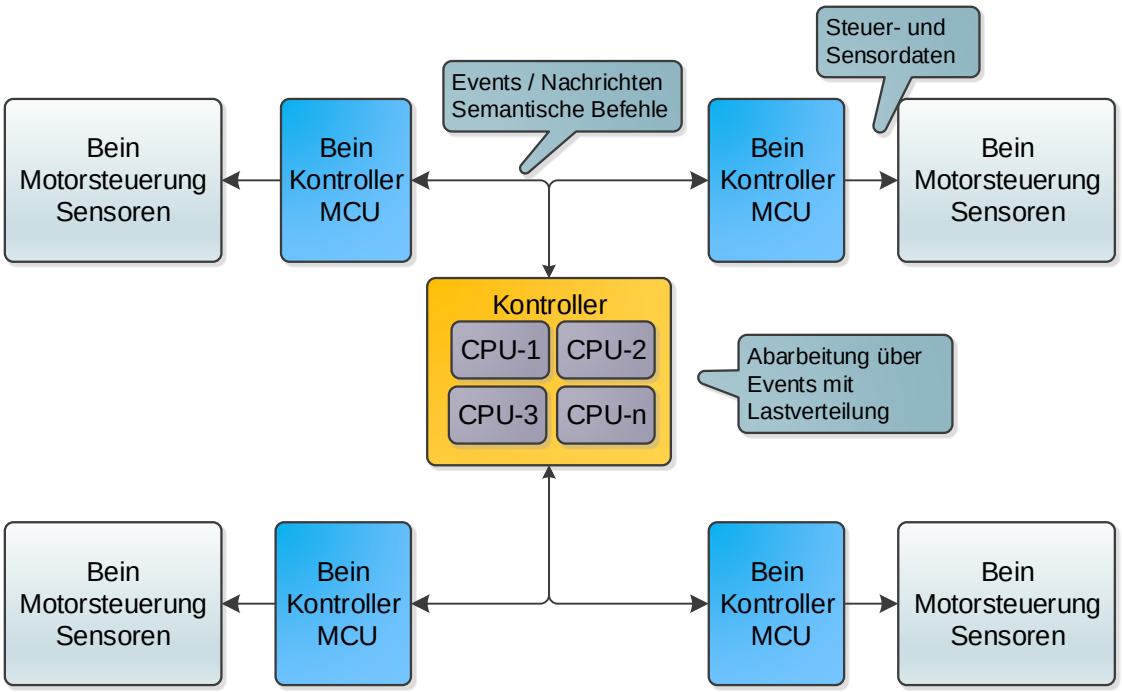


*Abbildung 4-1 Abstrakter Systemüberblick*

Die Beinkontroller enthalten alle benötigten Kontroller, die für eine Bewegung jedes Beins nötig sind. Dadurch ist es möglich jedem Bein, einen Auftrag zu übermitteln, der vom Bein autonom erledigt wird. Die physikalischen Berechnungen und das Verhaltensmodell der Beine sind für den Hauptkontroller vollständig verborgen. Der Hauptkontroller arbeitet als Koordinator zwischen den Beinkontrollern und plant komplexe Bewegungsabläufe. Er übermittelt den Beinen nur wohin und wie schnell sich die *Füße* bewegen sollen. Die Beinkontroller melden sich dann nur mit dem Erfolg oder Misserfolg ihrer Aktion zurück. Dies gilt sogar für eine Kollisionskontrolle mit Fehlerbehandlung. Dadurch tauschen die Beine und der Hauptkontroller keine Rohdaten (Sensordaten usw.) aus sondern nur semantische Informationen (abstrakte Zustände und Befehle).

### 4.2.1 Hybrides Layer Konzept

Im Kapitel (2.3) wurden die Vorteile eines hybriden Systems [Bra95] bereits diskutiert. In diesem Abschnitt wird die spezielle Ausprägung des planenden Teils erläutert. Für diesen Roboter wird ein

hybrides Layer-Konzept gewählt. Diese Entscheidung gliedert sich in zwei Designentscheidungen. Das erste Kriterium ist, das reaktive Verhalten durch ein ständig planendes System zu unterstützen. Der zweite Punkt ist eine Trennung von zeitkritischen und zeitunkritischen Aufgaben.
Die Mehrheit der Arbeiten aus dem „DARPA Learning Locomotion“ Programm (2.6) teilen die Aufgaben in einen reaktiven und planenden Bereich. Dies begründet sich durch das Vorausberechnen von mehreren Trittpositionsalternativen für den reaktiven Teil. Tritt ein Fehler auf, ruft der reaktive Teil eine vorausberechnete alternative Trittposition aus dem planenden (deliberative) Teil ab. Unabhängig von einem Fehlerfall berechnet der Deliberative-Layer ständig mehrere Verhaltensweisen im Voraus. Da diese Berechnungen fortlaufend mit aktuellen Umweltdaten erfolgen, können meistens mehrere Alternativen für den reaktiven Teil bereitgehalten werden.
In dieser Thesis wird die Architektur nach Zeitanforderungen (4.1.4) der Module erweitert. Der Reactive-Layer wird in zwei Layer aufgeteilt. Der neue *Upstream*-Reactive-Layer wird zwischen Reactive-Layer und Deliberative-Layer angeordnet (Abbildung 4-2). Der Reactive-Layer enthält alle Module mit harten Zeitanforderungen. Die verbleibenden Module des Reactive-Layers werden im *Upstream*-Reactive-Layer angeordnet und haben nur noch weiche Zeitanforderungen. Dies ist bedingt durch die Funktion des Reactive-Layers, der nur Aktionen ausführt, die vom Deliberative-Layer geplant wurden. Verzögert sich eine Berechnung im Deliberative-Layer, bleibt der Roboter einfach in einer stabilen Pose stehen – ohne umkippen. Natürlich sollte der Roboter flüssig laufen können und die Rechenzeit (Laufzeit) im Deliberative-Layer kann nicht völlig vernachlässigt werden. Es besteht aber nicht mehr die Not, eine garantierte Laufzeit zu erreichen. Als Überblick werden folgend die einzelnen Layer kurz vorgestellt. Die Auflösung in Module und deren Verfahren werden im Kapitel 4.3 erläutert.

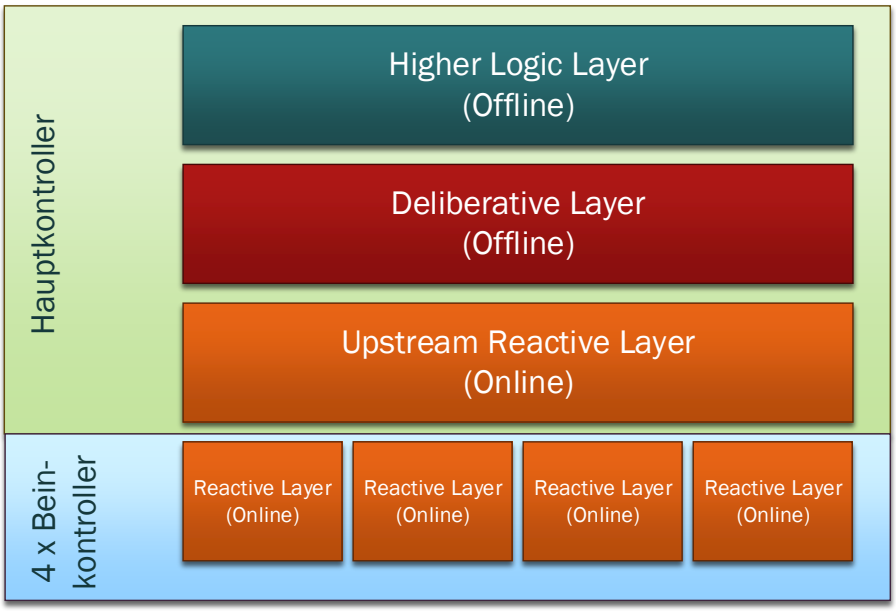


*Abbildung 4-2 Layerkonzept*

**Higher-Logic-Layer:** Das „klassische“ Layerkonzept wird um einen *Higher-Logic-Layer* ergänzt (Abbildung 4-2) und ist im Hauptkontroller angesiedelt (3.4.2). Dieser Layer ist für höhere Aufgaben reserviert und enthält Module, die frei von Laufzeitanforderungen sind. In einigen Arbeiten wird dieser Layer auch als Higher-Planning-Layer (z.B. [Kol09]) bezeichnet. Er erfasst die Umweltdaten und abstrahiert diese zu einem Weltmodell (4.1.3) für das Laufsystem. Zudem sollen hier auch geplante Module, wie beispielsweise die Pfadplanung und die Verhaltens-KI, implementiert werden. Dieser Layer wird auch als *Offline* bezeichnet, da er keinen direkten Zugriff auf das Laufsystem hat.
**Deliberative-Layer:** Der *Deliberative-Layer* ruft das aktuelle Weltmodell aus dem darüber liegenden Layer ab und erfasst die internen Lagesensordaten (4.2.2). Dieser Layer plant eine komplette Schrittfolge, um den Torso des Roboters zu bewegen. Er hat keinen direkten Zugriff auf die Mechanik (Offline). Dabei werden alle möglichen Trittpositionen aus dem Weltmodell ausgewählt und auf Gleichgewichtsstabilität geprüft. Weiterhin werden die daraus resultierenden Bewegungen auf eine Kollision geprüft. Diese Prüfung umfasst die Beine untereinander und den Torso. Auch dieser Layer ist im Hauptkontroller implementiert.
**Upstream-Reactive:** Der hier als *Upstream-Reactive*[14] bezeichnete Layer dient der Synchronisation zwischen den Beinen. Wird beispielsweise der Robotertorso bewegt, wird die Aktion von mehreren Beinen ausgeführt. Durch mechanische Ungenauigkeiten und unterschiedliche physikalische Momente auf die Beine müssen diese Bewegungen von den Beinen synchron ausgeführt werden. Der Roboter würde sonst von einer geplanten Trajektion abweichen. Zudem werden auch hier die Lagesensordaten erfasst und auf absolute Extrema geprüft. Nur dieser Layer enthält Schnittstellen zu den Beinen und wird auch als *Online* bezeichnet. Die Abläufe in diesem Layer unterliegen weichen Zeitanforderungen.
**Reactive-Layers:** Der Hauptteil des *Reactive-Layers* liegt direkt in den Beinen und ist viermal vorhanden (Online). Diese vier Teile sind absolut identisch und in eigener Hardware nebenläufig zum Hauptkontroller. Alle zeitkritischen Aufgaben der Beine sind hier gebündelt. Für diese Betrachtung und aus der Systemsicht ist dies ein einzelner Layer, aber die Beinkontroller haben wiederum eine eigene Layer-Architektur (Abbildung 4-5).

[14] Vorgelagerte reaktive Schicht

### 4.2.2 Systemfeedback – Sensoren

Die Annahmen aus Punkt 4.1.3 ermöglicht auch eine Trennung der „Sinne“ des Roboters. Damit ist es möglich, die Sensoren bestimmten Layern und Zeitkritiken zuzuordnen.

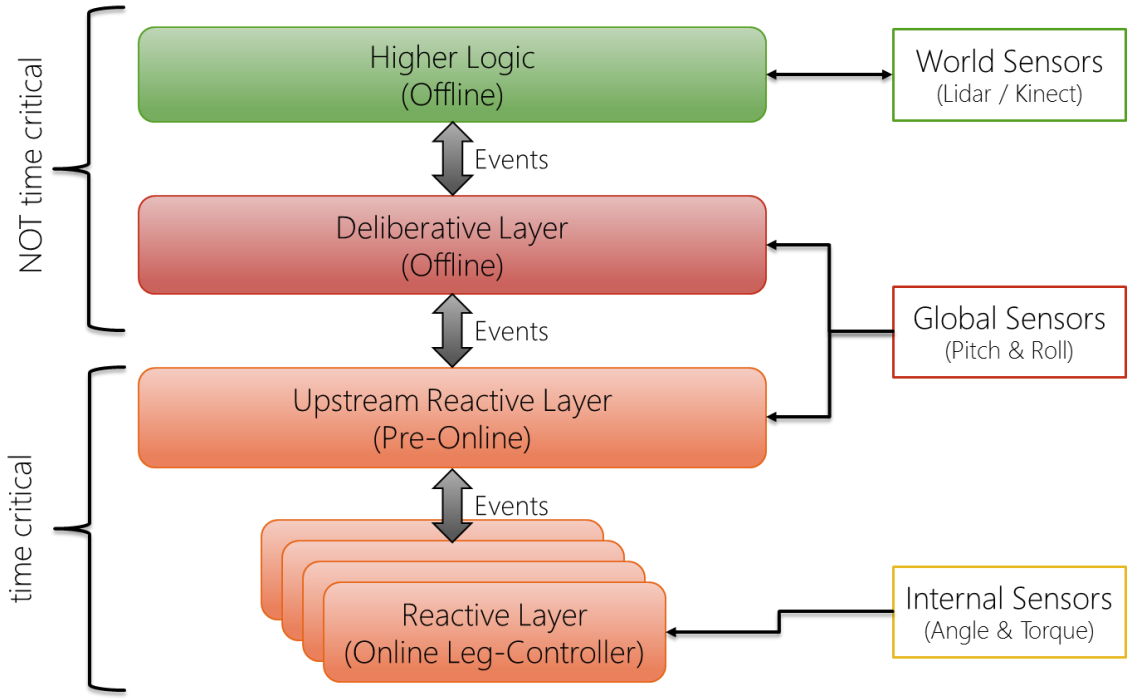


*Abbildung 4-3 Zuordnung der Sensoren*

**World Sensors:** Der Roboter erfasst seine Umwelt durch sogenannte *Umweltsensoren* (*World Sensors*). Diese Umweltdaten sind in diesem Konzept relativ zeitunkritisch und können damit in einer höheren Schicht erfasst und in ein Weltmodell umgewandelt werden.

**Global Sensors:** Als *globale Sensoren* werden Sensoren bezeichnet, die Informationen über den Roboter als Ganzes liefern. In diesem Roboter sind es die Informationen über die Lage und die Ausrichtung des Robotertorsos im Raum. Diese Informationen werden in beiden Zeitkriterien benötigt. Im *Deliberative-Layer* unterliegen diese Daten keinen Echtzeitanforderungen und werden für die Planung von stabilen Körperposen benötigt. Im *Upstream-Reactive-Layer* werden die Informationen zeitkritisch verwendet, um bei einer Fehlplanung ein umkippen zu erkennen und zu verhindern.

**Internal Sensors:** Die *internen Sensoren* (*internal Sensors*) umfassen Daten, die innerhalb der gekapselten Module verarbeitet werden. Dies sind hauptsächlich die Sensoren in den Beinen, wie beispielsweise die Winkel zwischen den Gliedern in einem Bein. Weiterhin werden auch die Drehmomente und die Drucksensoren erfasst.

Die Aufteilung der Sensordaten nach Zeitanforderungen und deren Zuordnung zu Layer ermöglicht eine klare Hierarchie des Informationsflusses und vereinfacht die Modellierung. Beispielsweise sind die Daten der Umweltsensoren (z.B. einer Punktewolke) für die direkte Steuerung eines Antriebs im

Bein sinnlos. Erst wenn alle Schichten hierarchisch durchlaufen wurden und aus den Daten abstrakte Informationen geworden sind, ist diese Information als Trittposition indirekt verwendbar. Dies gilt auch in umgekehrter Richtung, da beispielweise die aktuellen Drehmomentdaten für die Planung von Schritten belanglos sind.

### 4.2.3 Beinkontroller

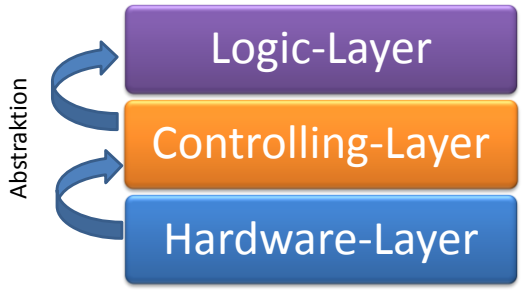


*Abbildung 4-4 Vereinfachte Layer-Architektur der Beinkontroller*

In diesem Konzept ist jedes Bein separat und nebenläufig modelliert und folgt damit der These aus 4.1.3. In den Beinkontrollern existieren harte Echtzeitanforderungen. Um diese These zu prüfen, arbeiten die Beinkontroller jeweils in einer physikalisch getrennten MCU (System-On-a-Chip).

Die Beinkontroller arbeiten nicht mit einem Regelalgorithmus, wie die meisten Systeme, sondern mit einem Regelwerk und einem hinterlegtem Verhalten. Dieses Verhalten steuert direkt die Antriebe und kann einfach um neue Regeln ergänzt werden. Die Beinkontroller bestehen wiederum aus hierarchischen Layern. In jedem Layer sind Module integriert, die meist nur einen physikalischen Zusammenhang bearbeiten. Die sog. „Smoothness“ (weiche und flüssige Bewegungen) wird durch einen Feature-Vektor erreicht, bei dem jedes Modul nur seine gewichtete „Sicht“ beisteuert. Der Durchschnitt dieses Feature-Vektors wird dann an die Antriebe weitergegeben. Durch eine Veränderung der Gewichtung eines Moduls kann das Verhalten des Beins verändert werden. Zudem werden, je nach Situation, vorgefertigte Gewichtungssets ausgelöst. Die Abstraktionsebene dieses Feature-Vektors ist die Drehzahl der einzelnen Antriebe. Mehr Informationen zu den Beinkontroller sind in der Bachelorarbeit „Entwicklung und Realisierung eines vierbeinigen USAR Roboter-Laufsystems“ [Ruh11] zu finden.

Der *Logic-Layer* (Abbildung 4-4), in jedem Beinkontroller, wertet die Nachrichten vom Hauptkontroller aus. Er zerlegt eine abstrakte Anweisung in einzelne Trajektionen und arbeitet diese ab. Dabei wird auch die Durchführbarkeit dieser Anweisung, gegen die Freiheitsgrade der Mechanik, geprüft.

Der *Controlling-Layer* prüft die Sensorinformationen und kontrolliert die Antriebe mit einem modellierten Verhalten. Dabei führt er auch die nötigen Berechnungen in Echtzeit aus. Im *Hardware-Layer* werden die I/Os abstrahiert und die Sensordaten aufbereitet.

Der Hauptkontroller kann dadurch komplexe Anweisungen bzw. Aufträge an die Beine übermitteln und überwacht nicht mehr die Einzelbewegungen. Beispielsweise sieht ein Auftrag wie folgt aus: „Schritt von $x_1$, $y_1$, $z_1$ nach $x_2$, $y_2$, $z_2$ mit Geschwindigkeit v“. Die Auflösung dieses Auftrags in Einzelbewegungen, das Erreichen des Ziels und die zeitkritischen Regelungsaufgaben werden autonom von den Beinkontrollern durchgeführt. Jeder Auftrag kann mit einem neuen Auftrag „überschrieben“ oder gestoppt werden. Die Beine können über ihren Zustand befragt werden.

## 4.3 Systemarchitektur Roboter AMEE

Die Architektur und das Konzept werden in diesem Kapitel genauer erläutert. Sie sind nur an die vorgestellten Arbeiten (2.6) angelehnt und mit eigenen Konzepten verändert worden. Das Kapitel 4.3.1 vermittelt eine kurze Übersicht der Architektur und beschreibt das Zusammenspiel der Komponenten. Die einzelnen Module werden in den Unterkapiteln 4.3.2 bis 4.3.5 detailliert erläutert.

### 4.3.1 Architekturübersicht

Die Softwarearchitektur wird in Abbildung 4-5 dargestellt. Jedes Modul ist lose gekoppelt und logisch einem Controller zugeordnet. Das System ist in drei Hauptlayer aufgeteilt, wobei der Higher-Logic Layer vorerst nur die Umwelterfassung enthält. Im *Online-Reactive-Layer* stellen die Controller reale Hardwarekomponenten dar, die mit dem Hauptsystem per Ethernet kommunizieren. Die Module im *Upstream Reactive-Layer* und der *Deliberative-Layer* werden von einem Timer (*Heartbeat*) angestoßen. Alle Komponenten, die als *Controller* bezeichnet werden, sind nebenläufig.

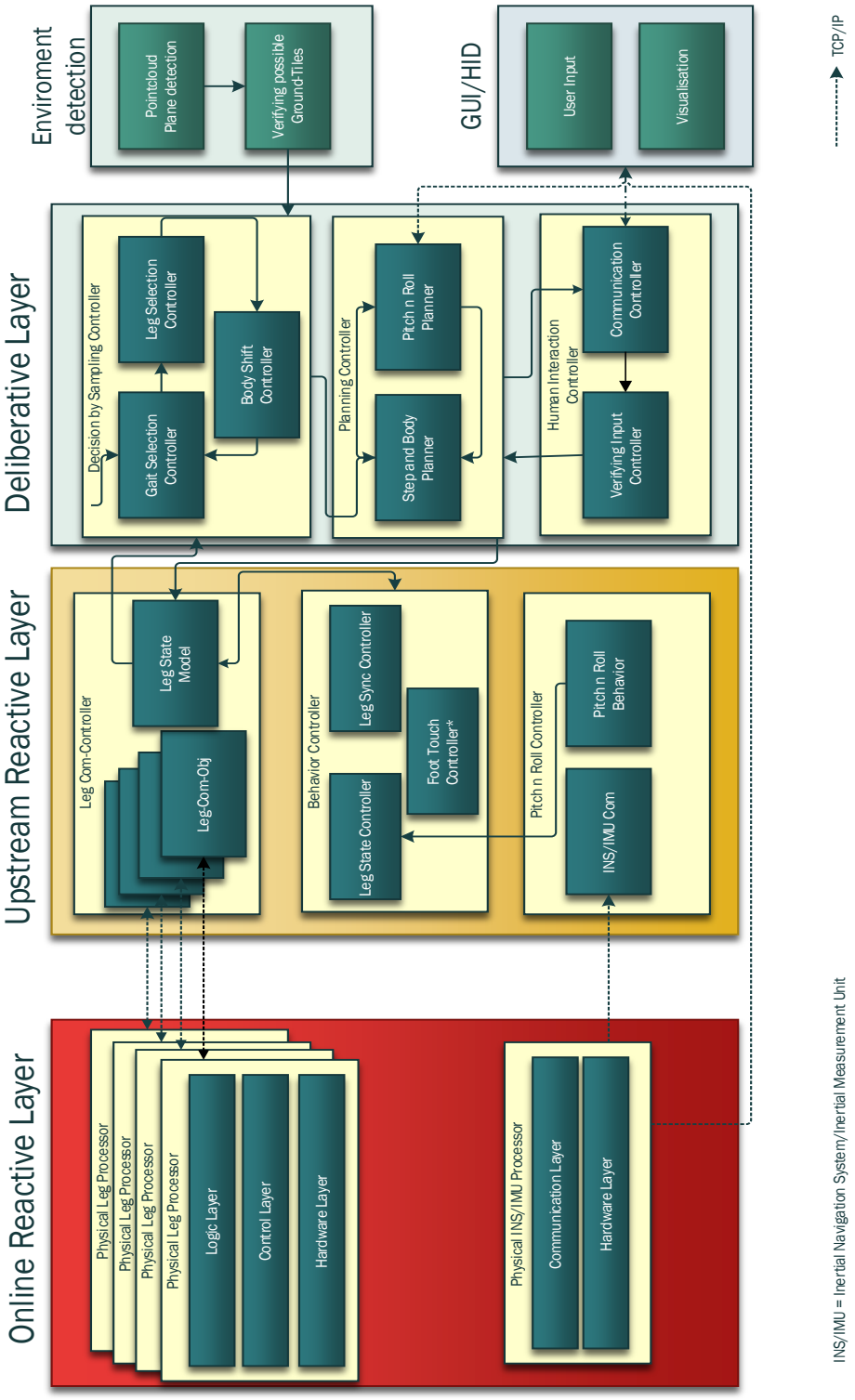


*Abbildung 4-5 Architektur für den Roboter AMEE*

**Online-Reactive-Layer(4.2.3)**: Die vier Beine, mit je einem physikalischen Prozessor, sind logisch einem Layer zugeordnet. Diese kommunizieren mit dem *Leg-Com-Controller* im *Upstream-Reactive-Layer* über Ethernet. Die ausgetauschten Nachrichten enthalten keine Steuer- oder Sensordaten, sondern nur semantische Nachrichten (4.2.3).
Die IMU[15] besteht aus einem Lage-, Trägheitssensor und drei Kompassen (3.4.3.1). Für die Kommunikation ist am Lagesensor eine MCU angeordnet. Diese MCU wandelt die Sensordaten in eventbasierte Lageinformationen um und ist per Ethernet an das Gesamtsystem angebunden.
**Upstream-Reactive-Layer(4.3.4)**: Im *Leg-Com(Communication)-Controller* befinden sich Interfaces für die Beine, die die Aufträge und Antworten verarbeiten und priorisieren. Dieser Controller wird periodisch von einem Heartbeat (4.3.5) angestoßen. Das *Leg-State-Model* bildet die Zustände der Beine ab und hält geplante Aktionen vor. Es ist die zentrale Schnittstelle zwischen den Kontrollern im Hauptsystem. Jeder laufende oder geplante Auftrag wird in das *Leg-State-Model* eingetragen und von dort an die entsprechenden Kontroller weitergegeben. Da fast jede Veränderung der Zustände eine Kommunikation mit den Beinen zur Folge hat, ist dieses Modul im *Leg-Com-Controller* angeordnet worden und unterliegt damit demselben Heartbeat.

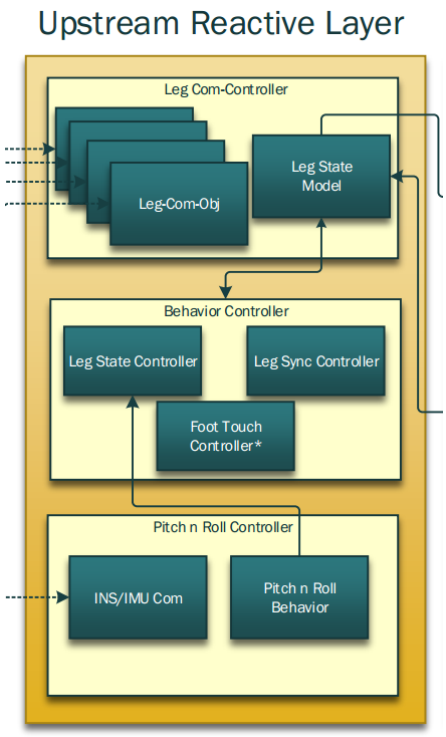

*Abbildung 4-6 Upst.Reactive Layer*

Im *Behavior-Controller* sind die Module für die reaktive Verhaltenskontrolle des Laufsystems gebündelt. Der *Leg-State-Controller* reagiert auf Lagedaten und Fehler der Beine mit einem modellierten Verhalten. Für komplexe Bewegungsabläufe und Schrittfolgen sind hier Automaten implementiert. Die Entscheidungen dieser Automaten werden an das *Leg-State-Model* übertragen. Der *Leg-State-Controller* leitet auch das richtige Verhalten bei einem Kippen des Roboters ein.
Der *Leg-Sync-Controller* überwacht die synchronen Bewegungen, wenn mehrere Beine an einer Bewegung beteiligt sind. Er verlangsamt oder beschleunigt einzelne Beine, wenn beispielweise unterschiedliche physikalische Einflüsse auf die Beine einwirken. Durch die komplett gekapselten Beinkontroller, wird dieses Modul nötig. Der *Leg-Sync-Controller* kann auch als ausgegliederter Koordinator der Beinkontroller verstanden werden.
Der *Foot-Touch-Controller* ist ein zusätzliches Sicherheitsfeature und koordiniert die Events der Drucksensoren in den Füßen. Zusätzlich zur Lagekontrolle kann hiermit ein Rückschluss auf den realen Zustand der Füße erfasst werden und beispielsweise auf ein Kippen des Roboters schnell reagiert werden.

[15] INS/IMU = Inertial Navigation System / Inertial Measurement Unit

Der *Pitch-and-Roll-Controller* (Lage und Bewegungssensor) überwacht die Lage des Roboters im Raum und gibt dem *Leg-State-Controller* Ausnahmeverhaltensweisen vor. Die Kommunikation zwischen der Vorverarbeitung des INS/IMU Prozessor und dem Hauptsystem wird durch das *INS/IMU-Modul* erreicht.

**Deliberative-Layer[16](4.3.3)**: Die Planung der Bewegungen wird über mehrere Vorverarbeitungsstufen realisiert. Dabei wird der Zustandsraum, einer abstrakte „Kachelwelt“ über ein Bewertungsverfahren, verkleinert.

Die erste Stufe ist im *Decision-by-Sampling-Controller* zusammengefasst. Anhand der erfassten und vereinfachten Umweltdaten wird eine Kostenkarte erstellt(4.3.3.1.1). Der *Gait-Selection-Controller* trifft eine Entscheidung über das Schrittmuster (4.1.5). Der *Leg-Selection-Controller* wählt das entsprechende Bein und die Trittpositionen anhand der Kostenkarte aus. Bei dieser Planung wird im *Body-Shift-Controller* entschieden und beurteilt, wieweit der Torso des Roboters bewegt werden kann. In dieser Abfolge der Module werden sukzessive Trittpositionen getestet und bewertet.

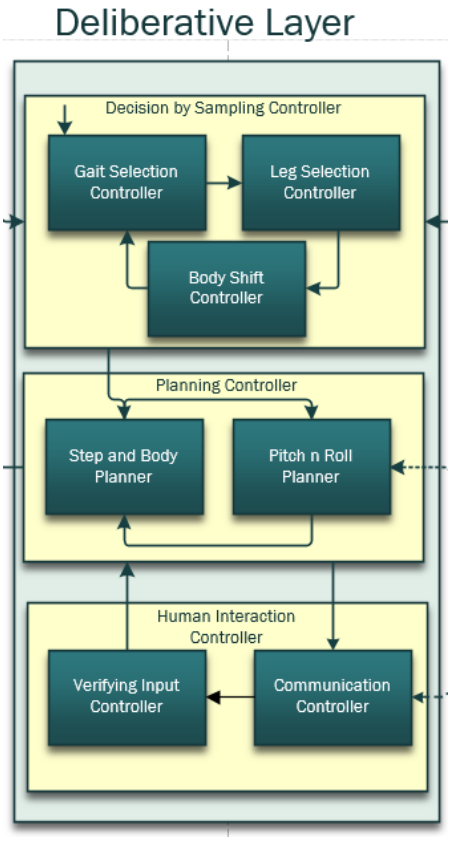


*Abbildung 4-7 Deliberative Layer*

Der *Planning-Controller* verwendet diese vorverarbeiteten Trittpositionen und prüft die daraus generierten Posen und Bewegungen des Roboters (4.3.3.2). Im *Step-and-Body-Planner* wird der Bewegungsablauf virtuell auf die Schwerpunktstabilität getestet. Weiterhin wird auch eine optimale Reihenfolge der Schritte gesucht (4.3.3.3). Der *Pitch-and-Roll-Planner* passt das Posenmodell, im *Step-and-Body-Planner,* an die aktuelle Ausrichtung des Roboters im Raum an.

Für die Interaktion mit einem Benutzer ist im *Deliberative-Layer* ein *Human-Interaction-Controller* mit zwei Modulen vorgesehen. Der *Communication-Controller* bereitet die internen Systemzustände auf und sendet diese per Ethernet an eine externe Bedieneinheit. Weiterhin empfängt er auch die Benutzerbefehle und leitet diese an den *Verifying-Input-Controller* weiter. Dieser Controller prüft die eingehenden Benutzerbefehle und beeinflusst nur die Bewertungsverfahren im *Planning-Controller*, um das gewünschte Verhalten zu erreichen. Da Benutzerbefehle zu einem kritischen Zeitpunkt eingehen können, kann der Benutzer nur die Planung beeinflussen. Beispielsweise kann ein Stoppbefehl des Benutzers nicht sofort ausgeführt werden, da der Roboter erst eine stabile Standposition einnehmen muss.

[16] In der Arbeiten von Kolter [And09] [Kol09] wird diese Stufen als High-Planning und Lower-Planning bezeichnet.

**Enviroment-Detection(4.1.3)**: Die *Enviroment-Detection* ist ein rudimentärer Teil des *Higher-Logic-Layers* und abstrahiert die Umweltsensordaten zu einer „Kachelwelt“. Dabei wird schon eine erste Vorselektion der Kacheln ausgeführt (4.1.3, 4.3.2.2).
Das **User-Interface (GUI/HID)** besteht aus einem externen System, das per Ethernet (WLAN) an das System angebunden ist. Es bestehen aber keine Abhängigkeiten zum Robotersystem. Das System bereitet die Systemzustände grafisch auf und stellt diese dar. Weiterhin wird eine GUI angeboten, die bestimmte Aktionen des Roboters einleiten.
**Hearbeat (4.3.5)**: Nicht dargestellt ist ein zentrales *Timermodul*, das jeden Kontroller in einem bestimmten Takt anstößt und das System überwacht. Das Modul agiert zudem wie ein Watchdog und registriert die Rückmeldung jedes angesprochenen Moduls. Nachdem ein Controller bzw. Modul angestoßen wurde und in einem Zeitfenster keine Rückmeldung erfolgt, kann das Heartbeat-Modul alle Systeme (auch externe) neu starten.

*Anmerkung: Die obige Übersicht soll nur das Zusammenspiel der Layer und deren Module verdeutlichen. Die eigentliche Funktion wird in den nachfolgenden Unterkapiteln anhand der Verfahren erläutert.*

### 4.3.2 Higher Logic Layer

In dieser Entwicklungsstufe befinden sich hier nur die Umwelterfassung /-abstraktion und eine rudimentäre Benutzerschnittstelle. Nach der eigenen Definition (4.1.4), das System nach Zeitkriterien zu unterteilen, gibt es in diesem Layer keine harten Zeitanforderungen. Vorgesehen ist hier beispielweise die Routenplanung, Gestenerkennung und vor allem ein lernendes System [Bet14] des Teamkollegen Bettzüche. Einige geplante Module werden in der Vision (6.2) als Arbeitspakete vorgesellt.

#### 4.3.2.1 Benutzerschnittstelle

Der Roboter ist zwar ein autonomes System, aber die komplexen Aufträge müssen von einen Operator gegeben werden. Zudem sollen die erfassten Informationen auf einem externerem Operatorsystem visualisiert werden. In dieser Entwicklungsstufe ist hier nur eine rudimentäre Schnittstelle vorgesehen, die die Systemzustände per WLAN an ein externes System senden kann. Zudem werden Anweisungen entgegengenommen. Die Nutzerbefehle verändern die Bewertungsmatrizen im *Deliberative-Layer (4.3.3)*.
*Anmerkung: Beispielweise verändert der Befehl, ein bestimmtes Gebiet bzw. Richtung[17] zu erkunden, die Richtungsmatrix in der Laufrichtungsplanung (4.3.3.1). Ein Stoppbefehl verändert die Entfernungs- und Richtungsmatrix in der Laufrichtungsplanung und die Reichweiten- und Richtungsmatrizen in der Trittpositionsplanung (4.3.3.2).*

[17] Eine vollwertige Pfadplanung wurde noch nicht integriert. Siehe dazu 4.3.3.1 und 6.2.2

#### 4.3.2.2 Umwelterfassung

Für die Umwelterfassung / Umweltabstraktion (4.1.3) wird eine Alphaversion des Kinect for Windows V2® Sensors verwendet (3.4.3.1) und ist fest am „Hals“ des Roboters montiert. Dadurch ist die Punktwolke bzw. die Kachelwelt immer nach vorn ausgerichtet und dreht sich mit dem Roboter. Der Sensor ist dabei so ausgerichtet, dass der Erfassungsbereich senkrecht vor der „Brust“ beginnt. In dieser Ausrichtung kann auch der Horizont (bis ca. 4,5m) erfasst werden, was durch den großen Erfassungswinkel (ca. 179°) ermöglicht wird.

Die mitgelieferte API liefert eine 2.5 dimensionale Punktewolke. Die Punktwolke ist bereits auf ein kartesisches Koordinatensystem korrigiert und enthält die Koordinaten in metrischen Einheiten. Diese Punktwolke wird in kleinere, gleichgroße und kubische Subpunktwolken segmentiert. Für die erste Stufe der planenden Schicht (Deliberative-Layer (4.3.3)) wird ein Würfel mit einer Kantenlänge von 40cm gewählt. Dies entspricht der Seitenlänge eines Fußes. In jeder Subpunktwolke wird eine Fläche gesucht. Von diesem Modul werden die erkannten Flächen bzw. Flächennormalen an die Laufrichtungsplanung (4.3.3.1) übergeben. Die segmentierte Punktwolke wird zudem für eine genauere Untersuchung in der zweiten Stufe der Planung (4.3.3.2) bereitgehalten.

In der zweiten Stufe der Planung wird die genaue Trittposition durch den Mittelpunkt der Fläche abgebildet. Dabei wird der Mittelpunkt des Fußes (in der Draufsicht) auf den erfassten Flächenmittelpunkt gesetzt. Wird ein Fuß auf eine Fläche gesetzt, die nicht geradlinig vor dem Körper sitzt, ragen die Zehen leicht aus der erfassten Fläche. Dies wird durch eine zu groß gewählte Kachelgröße kompensiert. Zudem ist in diesem Entwicklungstand die seitliche Drehung eines Beins pro Schritt auf maximal 15° Winkelgrad begrenzt (4.3.3.2).

Für die Erkennung der Flächen in den Subpunktwolken wird das RANSAC-Plane Verfahren verwendet. Diese Verfahren liefert grundsätzlich eine Fläche pro Punktwolke, wenn in ihr mindestens drei Punkte vorhanden sind. Der sog. inlier-Wert bestimmt, wieviel Prozent der Punktwolkenpunkte im Toleranzbereich einer Fläche liegen müssen, um als Fläche interpretiert zu werden. Durch die Rückgabe einer Flächennormale und die bekannte Subwürfelkantenlänge wird die Fläche abgebildet.

Bedingt durch die relativ geringe Streuung des verwendeten 3D Sensors kann der inlier-Wert aktiv genutzt werden. Es kann damit festgelegt werden, wie groß ein Objekt auf einer Fläche sein darf, bis dies als Hindernis bzw. keine Fläche interpretiert wird. D.h. liegen kleine Objekte auf einer Fläche, die den Schwellwert unterschreiten, wird die Punktwolke trotzdem als Fläche erkannt. *Anmerkung: Es ist aber schwierig einen inlier-Schwellwert zu finden, der auf jedem Untergrund nutzbare Ergebnisse liefert. Unter Laborbedingungen und befestigtem Untergrund werden gute Ergebnisse erzielt.*

Weiterhin kann die Erkennungsgenauigkeit auch durch die Anzahl der Iterationen beeinflusst werden. Das verwendete RANSAC-Plane Verfahren wählt pro Iteration drei zufällige Punkte aus der Punktwolke aus. Über diese drei Punkte wird eine Testfläche gespannt. Danach wird mit dem inlier-Wert geprüft, wie viele der Punkte auf dieser Fläche liegen. Mit jeder Iteration nähert sich das Verfahren der zu erkennenden Fläche an. Das heißt, mit jeder Iteration wird die Erkennung genauer, bis ein Maximum erreicht wurde. Es muss die Erkennungsgenauigkeit gegen die Erkennungsgeschwindigkeit abgewogen werden.

### 4.3.3 Deliberative Layer / Planung

Ein hybrides System mit planendem Layer bietet einige Vorteile gegenüber anderen Architekturen (2.3), wenn ein QRTR im Gelände schreiten soll. Die Module in diesem Layer planen über mehrere Stufen das Verhalten des Laufsystems. Das Resultat dieses Layers sind schwerpunktstabile und optimale Bewegungsabläufe und Trittpositionen. Die Planung umfasst nur eine Schrittfolge. Dies beinhaltet alle Bewegungsabläufe, die nötig sind um den Torso in Laufrichtung zu ziehen (Body-Shift). Da die Wahrscheinlichkeit eines Fehltritts im Gelände relativ hoch ist (2.3), werden zu diesem Zweck auch alternative Trittpositionen und Bewegungsabläufe berechnet. Diese werden dem ausführenden Layer (Reactive-Layer (4.3.4)) übergeben.
Durch die Vorausberechnung einer Schrittfolge ist der deliberative-Layer der realen Mechanik eine Schrittfolge voraus (gilt nur für den Fast-Walk 4.1.5). Zum Zeitpunkt $t_1$ , wird die Vorausberechnung von $t_0$ mit der Mechanik ausgeführt. Im Idealfall wird die komplette Vorausberechnung in der Bewegungszeit der Mechanik abgeschlossen und ermöglicht damit einen flüssigen Übergang der Schrittfolgen. Aus dieser Überlegung heraus wurde versucht, Verfahren zu entwickeln, die auf komplexe Operationen und hohen Rechenaufwand verzichten.
Das in dieser Thesis vorgestellte System enthält noch keine Pfad- / Routenplanung, die mehr als eine Schrittfolge berücksichtigt. Um eine generische Pfadplanung anbinden zu können, wird die Laufrichtung im Gradmaß der Himmelsrichtung (Winkel zum Azimut) angegeben.
Die planenden Verfahren, die in den folgenden Unterkapiteln erläutert werden, sind in drei Stufen gegliedert (Abbildung 4-8 blau, grün, lila). Die erste Stufe wertet grob die Umwelt aus und ermittelt eine begehbare Laufrichtung (4.3.3.1, blau). Die zweite Stufe selektiert einen begehbaren Korridor (4.3.3.2, grün). Die dritten Stufe generiert aus den vorselektierten Kacheln der zweiten Stufe, Posen und Bewegungen und prüft zudem die Schwerpunktstabilität (4.3.3.3, lila). Jede Stufe übergibt eine sortierte Liste seiner Ergebnisse an die nächste Stufe. Die erste Stufe übergibt aber nur eine *optimale Richtung*, nicht die Trittpositionen.

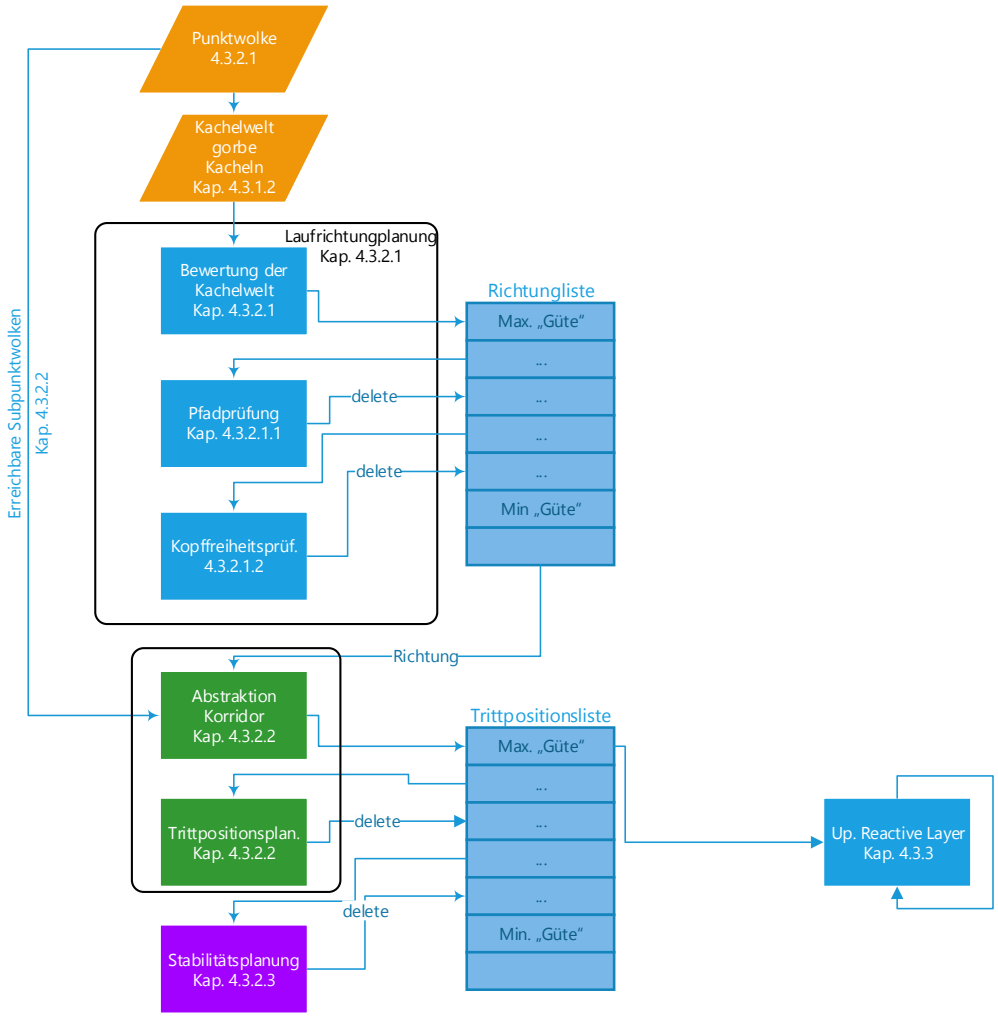


*Abbildung 4-8 Abstrakter Ablauf deliberative Layer*

*Anmerkung: Um das Verfahren nachvollziehbar zu halten, werden einige Module zusammengezogen und nicht wie in der Architekturübersicht (4.3) dargestellt benannt. Dies ist nötig, da die meisten Module lose gekoppelt sind, parallel laufen und nur durch den Heartbeat in der richtigen Reihenfolge angestoßen werden. Die obige Architektur ist zwar strukturiert zu implementieren, eignet sich aber schlecht für eine textuelle Darstellung. In den folgenden Unterkapiteln (4.3.3.1, 4.3.3.2, 4.3.3.3) werden die Verfahren und ihre Ablaufe beschrieben und nicht die modulare Aufteilung. Deshalb werden zusätzlich die Architekturnamen im Text mit geschweiften Klammern dargestellt.*

#### 4.3.3.1 Laufrichtungsplanung

*{Architekturnamen: Step and Body Planner, Pitch and Roll Planner}* Diese Verfahren wählt eine Laufrichtung für den nächsten Schritt des Roboters aus. Auch wenn noch keine Routenplanung implementiert wurde, ist die Laufrichtung von Bedeutung. Um nicht die gesamte erfasste Umwelt auf schwerpunktstabile Trittpositionen zu testen, wird eine Vorselektion der Kacheln durchgeführt.
Nach jeder Schrittfolge (bzw. Bewegung des Torsos) wird eine neue Kachelwelt(4.1.3) aus dem Higher-Logic-Layer (4.2.1) angefordert. Die Kachelwelt repräsentiert direkt den Erfassungsbereich des Umweltsensors (3.4.3.1). In diesem Modul hat die Kachelwelt eine Auflösung von 11x11 Kacheln mit einer Kachelkantenlänge von 40cm (die maximale Reichweite des Sensors beträgt ca. 4,5m). Die Flächenerkennung (RANSAC-Plane 4.3.2.2) wird auf wenige[18] Iterationen begrenzt, um eine grobe Übersicht auf die Umwelt zu erhalten. Die Füße stehen immer in der Mitte des unteren Rands der Kachelwelt (Abbildung 4-9 ②). Für die Laufrichtungsplanung werden jeweils zwei geradlinige Korridore / Streifen aus der Kachelwelt ausgewählt und repräsentieren ein virtuelles durchschreiten der Umwelt. Dabei werden iterativ alle möglichen geradlinigen Streifen von den Füßen im Uhrzeigersinn betrachtet. Der Abstand der Streifen entspricht dem Abstand der Vorderfüße. Um Hindernisse zu erkennen und möglichst effiziente Trittpositionskorridore zu finden, wird das Verhalten über mehrere Gewichtungsmatrizen modelliert. Diese Matrizen werden mit den erfassten Umweltdaten verrechnet, um eine Entscheidung zu treffen [Köc05].
Das folgende Verfahren bildet pro Durchlauf eine Momentaufnahme der Umwelt ab. Dadurch kann sich die Laufrichtung nach jeder Schrittfolge ändern. Dieses Verhalten ist erwünscht und wird in der Kapitelzusammenfassung (4.3.3.1.4) erläutert. Das Verfahren ist aber *kein* Ersatz für eine Pfad-/Routenplanung. Der Begriff „Pfad“ wird folgend für die Richtungsentscheidung und die Abfolge der Einzelentscheidungen verwendet.

[18] Es wurde hier mit 200-500 Iterationen experimentiert. Die untere Anzahl der Iterationen um eine Fläche zu erkennen, schwankte stark nach Umweltbedingungen und bei komplexen Formen.

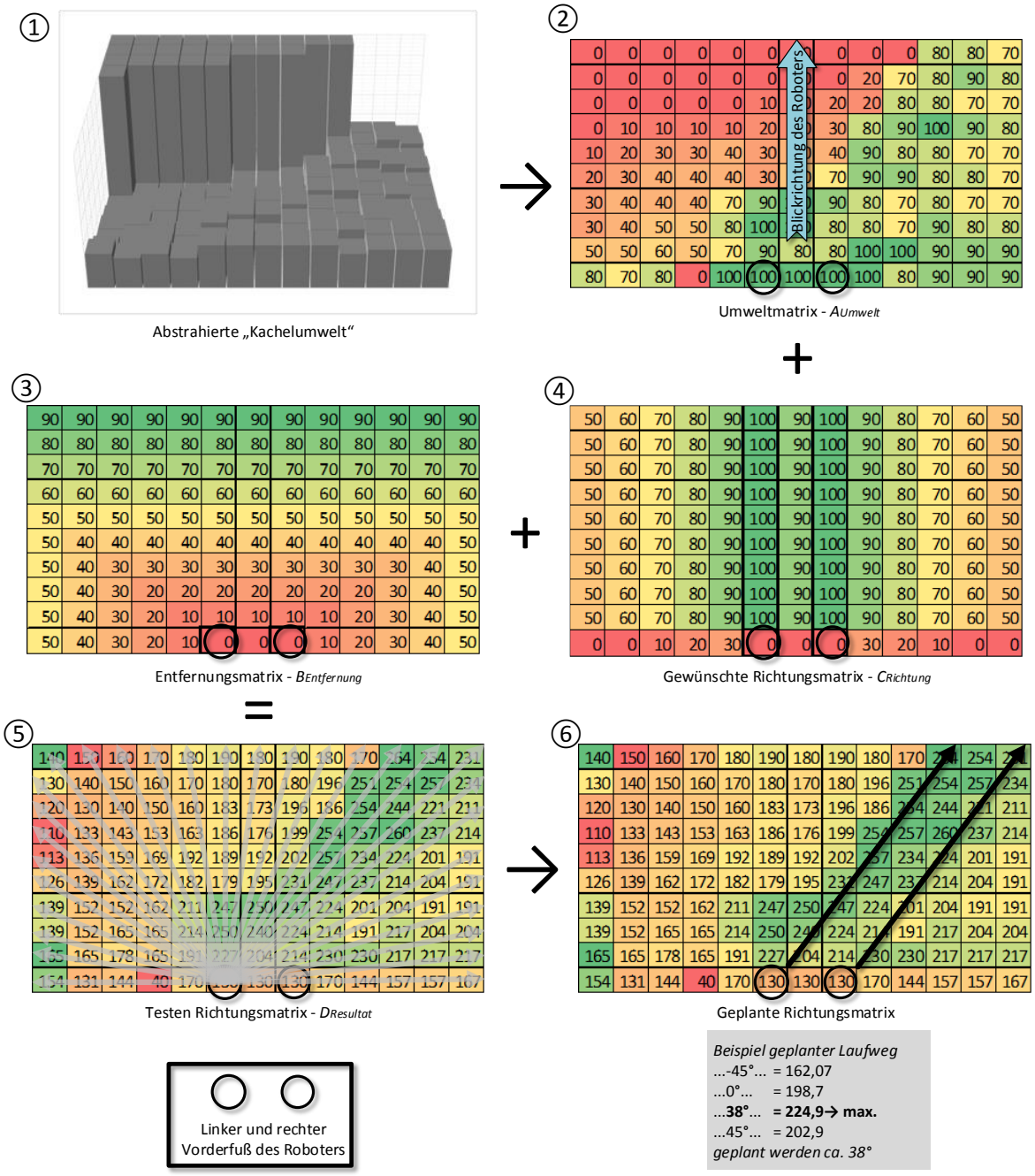


*Abbildung 4-9 Laufrichtung anhand von Gewichtungsmatrizen*

**Umweltmatrix**: Die Umweltmatrix ist eine zweidimensionale Draufsicht auf die Umwelt und abstrahiert die Kachelwelt (Abbildung 4-9 ①, 4.3.2.2) zu einer Kostenkarte. Die Kachelwelt wird aus dem High-Logic-Layer abgerufen (4.3.2) und die Bewertung findet in *zwei* Durchläufen statt. Nach jeder Positionsveränderung des Roboters wird jede Kachel der Kachelwelt nach Höhenunterschieden

bewertet und als $A_{Umwelt}$ ② bezeichnet. Die Bewertung wird in Prozent der Erreichbarkeit durch das jeweilige Vorderbeinbein vorgenommen, wobei die aktuelle Standhöhe jedes Vorderfußes als 100% (gut) festgelegt wird. Damit erhält jede Kachel, die auf gleicher Höhe zum Fuß ist, 100 Prozent. Ein Höhenunterschied wird für höherliegende Kacheln von 99% bis 0% *absteigend* bewertet. Dabei bedeuten 0% (schlecht) einen Höhenunterschied, der so groß ist, das die Kachel nicht erreicht werden kann. Eine Bewertung von 1% repräsentiert die maximal erreichbare Höhe des Beins. Liegen Kacheln tiefer als die aktuelle Standposition, werden sie äquivalent mit negativem Vorzeichen bewertet. Eine Kachel mit -0%, liegt so tief, das der Roboterfuß sie nicht erreichen kann. Eine Bewertung mit -1% repräsentiert ein maximal ausgestrecktes Bein. Die Bewertung ist dynamisch zur momentanen Höhe des Torsos bzw. der momentanen Beinauslenkung normiert. Ist beispielwiese der Torso schon relativ nah am Boden, ist die Erreichbarkeit (durch die eingenickten Beine) nach oben geringer als nach unten. Damit ist die Skalierung des negativen und positiven Bereichs nicht immer symmetrisch. Wurde in einem Subwürfel (4.1.3) keine Fläche/Kachel gefunden, wird auch sie mit -0% gekennzeichnet. Der Intervall des Bewertungsverfahrens reicht damit von $[-0\% \dots -50\% \dots 100\% \dots 50\% \dots 0\%]$, wobei 100% den Nullpunkt des Intervalls abbildet. In diesem ersten Bewertungsdurchlauf werden die Neigung des Torsos und die relative Neigung der Kacheln nicht berücksichtigt. Die Beweggründe für diese Designentscheidung werden in der Kapitelzusammenfassung (4.3.3.1.4) erläutert.

In einem zweiten Durchlauf werden Hindernisse und zu schräge Flächen aussortiert. Es werden nur Kacheln betrachtet, die im vorherigen Durchlauf nicht mit -0% oder 0% bewertet wurden. Alle Kacheln die nicht im Intervall von -20° bis 20° Winkelgrad liegen, werden mit -0% bewertet. Das negative Vorzeichen kennzeichnet eine schräge Kachel als Vertiefung für die Pfadprüfung. Die Unterscheidung zwischen Wänden und kleinen senkrechten Absätzen, erfolgt in der Pfadprüfung (4.3.3.1.1) und wird dort erläutert. Im Gegensatz zum ersten Bewertungsdurchlauf bezieht sich das Winkelintervall auf den realen Horizont und nicht auf die Lage des Torsos. Dafür wird die Kachelneigung (Rückgabe des RANSAC Plane Verfahrens) um die Torsoneigung (aus dem Lagesensor) korrigiert. Diese Winkelbegrenzung[19] verhindert ein zu starkes Rutschen des Roboters auf losem Untergrund.

Durch das obige Vorgehen wird die dreidimensionale Kachelwelt auf eine zweidimensionale Fläche projiziert. Jeder Kachel wird zu einem Matrixelement mit Metainformationen. Die Position eines Elements in der Matrix repräsentiert dabei die Breiten- und Tiefeninformation. Die Höhe wird zur „Güte“ der Erreichbarkeit abstrahiert. Die entstandene *m x n* Matrix bildet die Umwelt, aus der momentanen Sicht des Laufsystems, ab. Das ungewöhnliche Intervall der Bewertung beschleunigt die abschließenden Berechnungen des Verfahrens. Um die beiden Extreme zu unterscheiden, hat die Null ein Vorzeichen. Dies wird über ein Vorzeichen-Flag realisiert.

**Entfernungsmatrix**: Die sog. Entfernungsmatrix ($B_{Entfernung}$) Abbildung 4-9③ veranlasst das System, weit entfernte Kacheln höher zu gewichten. Die Messungen des Umweltsensors werden mit zunehmender Entfernung ungenauer und der Sichtschatten größer. Würden die Kacheln gleich

[19] In einer möglichen Erweiterung könnte hier ein Bewertungsverfahren über die Bodenbeschaffenheit angebunden [Bet14] werden, dass den Grenzwert des Winkels individuell für jede Kachel bestimmt.

bewertet werden, würde das System „übervorsichtig“ und jeder mögliche Störung weiträumig ausweichen. Durch die höhere Gewichtung der entfernten Kacheln, wird ein vorsichtiges Annähern an ein mögliches Hindernis gefördert und die „Störung“ abgemildert. Das System wird damit veranlasst, Hindernisse eher im Nahbereich zu beurteilen. Das Intervall reicht von 0 bis 100. Die aktuelle Fußposition wird mit 0 bewertet und entspricht damit keiner Entfernungskorrektur. Die Werte steigen konzentrisch von der Fußposition auf und 100 stellt die maximale Entfernung dar.

**Richtungsmatrix**: Die gewünschte Richtung wird über die Richtungsmatrix ($C_{Richtung}$ Abbildung 4-9 ④) modelliert. Diese Matrix ist auch vom selben Matrizentyp und wird wieder von 0 bis 100 gewichtet, wobei 100 genau die geplante Richtung repräsentiert. In der Matrix werden zwei geradlinige Korridore mit 100 gewichtet. Jeder Korridor geht jeweils von einem Vorderfuß aus und zeigt auf den Rand der Kachelwelt in die gewünschte Richtung. Dadurch ergeben sich zwei „Streifen“ mit einer Gewichtung von 100. Die Gewichtung nimmt im rechten Winkel vom Korridor ab. Damit wird ein Abweichen von der gewünschten Richtung schlechter bewertet, um den Roboter auf Kurs zu halten.

Für das vorgestellte Laufsystem ist ein Drehen auf der Stelle relativ komplex [Zha05]. Hierfür müssten Laufmuster mit überkreuzenden Beinen modelliert werden. Da diese Laufmuster nur für diesen Zweck verwendet werden können, wird dieser Aspekt auf eine spätere Entwicklungsstufe verschoben. Bedingt durch diesen Umstand sind die Kacheln direkt neben den Füßen extrem schlecht bewertet ($C_{Richtung}$ Abbildung 4-9 ④). Wäre das Drehen auf der Stelle bereits modelliert, würden sich diese Werte nicht von der restlichen Verteilung unterscheiden.

**Resultatmatrix und Korrekturskalar**: Die obigen Matrizen werden mit einem Skalar multipliziert und anschließend addiert. Die Ergebnisse werden in die Resultatmatrix $D_{Resultat}$ (Abbildung 4-9⑤) eingetragen.

$$\alpha \cdot |A_{Umwelt}| + \beta \cdot B_{Entfernung} + \gamma \cdot C_{Richtung} = D_{Resultat}$$

Mit den Skalaren $\alpha, \beta, \gamma$ kann das Verhalten der Richtungsauswahl verändert werden. Mehr dazu in der Kapitelzusammenfassung (4.3.3.1.4). Bei der Addition der Umweltmatrix ($A_{Umwelt}$) werden vorerst nur die absoluten Werte benutzt. Um die Komplexität geringer zu halten, werden extreme Sprünge in den Höhenunterschieden vorerst vernachlässigt und erst in der Pfadprüfung berücksichtigt.

**Richtungsauswahl und Pfadbildung**: Zur Auswahl eines begehbaren Pfads, der möglichst in die gewünschte Richtung zeigt, werden alle geradlinigen Pfade in der Resultatmatrix ($D_{Resultat}$) betrachtet. Ausgehend von den Fußpositio**nen** wird jeder Pfad bis zur Kante der Kachelwelt geprüft (Abbildung 4-9 ⑤). Um auch hier laufzeitoptimal zu arbeiten, sind einige Vereinfachungen eingeführt worden. Die Pfadwinkel werden nicht über Winkel(4.3.3.1.3) berechnet, da die gewünschte Richtung implizit in der Resultatmatrix enthalten ist. Die Robotervorderfüße befinden sich meist an der unteren Kante der Kachelumwelt. Es wird jeweils ein geradliniger Pfad von jedem Vorderfuß bis zur Kante der Kachelwelt untersucht. Es wird auch vernachlässigt, dass die Pfade in der linken und rechten Ecke nicht genau parallel verlaufen. Diese Ungenauigkeiten werden akzeptiert, da dieses Verfahren nur eine

schnelle Richtungsentscheidung treffen soll und diese Bewertung nicht als ausführbare Trittposition verwendet wird.

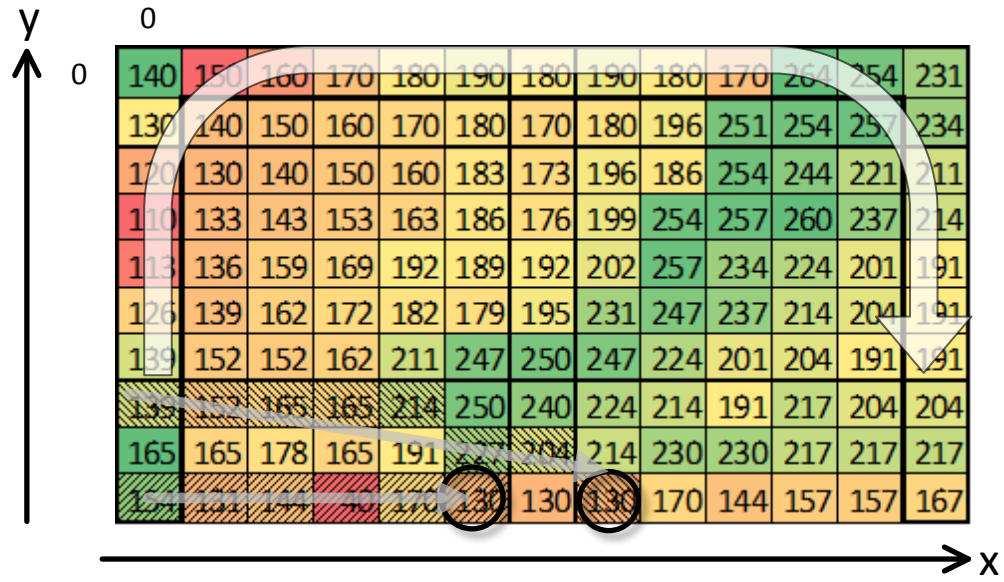


*Abbildung 4-10 Gewichtete Umwelt mit Testpfaden*

Ausgehend von der linken unteren Kachel, am Rand der Resultatmatrix ($D_{Resultat}$), werden Kacheln ausgewählt, die sich auf einem geradlinigen Pfad zum linken Vorderfuß befinden (Abbildung 4-10). Die Werte dieser gewählten Kacheln werden summiert und durch die besuchte Kachelanzahl dividiert. Für den rechten Vorderfuß wird genauso verfahren und auf den Pfad des linken Vorderfußes addiert. Der Versatz am Kachelrand entspricht dem Fußabstand. Die resultierende Summe wird in eine sortierte Liste absteigend einsortiert. Für den ersten Pfad in Abbildung 4-10 würde folgende Rechnung aufgestellt werden:

$$S_n = \frac{\sum d_{x1,y1} + \sum d_{x2,y2}}{Anzahl\ Elemente}$$

$$S_0 = \underbrace{\frac{(154 + 131 + 144 + 40 + 170 + 130)}{6}}_{Linker\ Fuß} + \underbrace{\frac{(139 + 152 + 165 + 165 + 214 + 227 + 204 + 130)}{8}}_{Rechter\ Fuß}$$

*Um das Verfahren nachvollziehbar zu halten wird die **Pfad**- und **Kopffreiheitsprüfung**, im Zuge obigen Additionen, nicht dargestellt. In der Implementierung werden die Kacheln zusätzlich auf extreme Höhenunterschiede untersucht. Dies wird im Absatz Pfadprüfung (4.3.3.1.1) und Kopffreiheitsprüfung (4.3.3.1.2) beschrieben.*

Damit ist die Bewertung eines Pfades beendet und das Verfahren wählt den nächsten Pfad im Uhrzeigersinn aus. Dabei werden die Randkacheln von unten links bis zum Nullpunkt der Y-Achse durchiteriert. Danach wird die X-Achse mit Y=0 betrachtet. Zuletzt wird die rechte Kante mit X = max. und die Y-Achse iterativ betrachtet.

Die entstandene sortierte Liste mit dem gewichteten Durchschnitt aller möglichen Pfade wird an die Trittpositionsplanung (4.3.3.2) übergeben. Der maximale Wert in dieser Liste repräsentiert den optimal begehbaren Pfad. Im Beispiel aus Abbildung 4-9 ⑥ ist das Resultat ein Pfad, der ca. 38° von der geplanten Richtung abweicht und das Hindernis umgeht.

*Anmerkungen: Die Darstellung des Fußabstandes ist zu groß und wurde nur für eine bessere Visualisierung gewählt. Der reale Roboter AMEE hat eine Fußauflagefläche von ca. 38cm x 38cm pro Fuß. Bei gerader Ausrichtung der Beine beträgt der Abstand zwischen den Füßen ca. 3-4cm. Durch die gewählte Kantenlänge der Kacheln von 40cm liegen diese direkt nebeneinander. Damit kann in der realen Anwendung auf eine Hindernisprüfung zwischen den Beinen auf dem Pfaden verzichtet werden.*

#### 4.3.3.1.1 Pfadprüfung

Bei der abschließenden Berechnungen in der Richtungsauswahl (4.3.3.1) liegt erst jetzt eine definierte Bewegungsrichtung vor. Damit ist es möglich, jeden Pfad auf extreme Höhenunterschiede zu untersuchen, ohne die komplette Nachbarschaft jeder Kachel zu betrachten. Die vorherige Betrachtung hat die Umwelt nur nach der Güte der Erreichbarkeit, vom aktuellen Standpunkt aus, untersucht. Würde eine Umweltkachel mit -10% und die Nachbarkachel mit +10% bewertet werden, ergibt sich ein nicht überwindbarer Höhenunterschied. In der realen Umwelt entspricht dies beispielweise einem Loch im Boden, das ungefähr den Durchmesser des Fußes und fast die Tiefe der Beinlänge hat. Auf das Loch folgt eine Erhebung, die wiederum fast die Beinlänge hat, aber vom Scanpunkt erreichbar wäre. Würde der Roboter diesem Pfad folgen und in das Loch treten, könnte er die Erhebung nicht mehr erreichen (Abbildung 4-11 ①). Diese Abfolge von extremen Unebenheiten wird bis jetzt nicht berücksichtigt. Diese Prüfung erfolgt erst in diesem Modul. Ausgehend von der aktuellen Fußposition wird jeweils nur die Nachbarkachel untersucht, die sich in Bewegungsrichtung auf dem Pfad befindet.

$$Pfadkachel_n = P_n = A_{Umwelt\ x,y}$$

FOR ALL $P_n$

$$Kachel\delta_n = \begin{cases} P_{n+1} + 100\ f\ddot{u}r\ P_{n+1} < 0 \\ P_{n+1} - 100\ f\ddot{u}r\ P_{n+1} \geq 0 \end{cases} - \begin{cases} P_n + 100\ f\ddot{u}r\ P_n < 0 \\ P_n - 100\ f\ddot{u}r\ P_n \geq 0 \end{cases}$$

$$IF\ Kachel\delta_n < -100\ THEN \underset{Vertiefung}{\Longrightarrow}$$

$$C_{Resultat\ x,y} = -0$$

$$IF\ Kachel\delta_n = -0\ AND\ Kachel\delta_{n-1} = -0\ AND\ Kachel\delta_{n-2} = -0\ THEN$$

$$C_{Resultat\ x,y} \dots C_{Resultat\ x.end,y.end} = 0$$

$$BREAK$$

$$ELSE\ IF\ Kachel\delta_n > 100\ \ THEN \underset{Erhebung}{\Longrightarrow}$$

$$C_{Resultat\ x,y} \dots C_{Resultat\ x.end,y.end} = 0$$

$$BREAK$$

$$NEXT$$

Als Basis für die Bewertung werden die Werte aus der $A_{Umwelt}$ Matrix verwendet. Der Höhenunterschied $Kachel\delta_n$ wird über die Differenz zwischen der Folgekachel und der aktuell betrachteten Kachel mit obiger Gleichung berechnet. Ist dieser Unterschied kleiner als -100, was einem maximal ausgestreckten Bein entspricht, wird für die Folgekachel der Wert (-)0 in die $C_{Resultat\ x,y}$ Matrix eingetragen und ist damit eine nicht begehbare Kachel. Treten in Folge mehr als drei „negative" Nullen (Vertiefungen) auf, handelt es sich um eine lange Vertiefung (Abbildung 4-11 ②). Das Laufsystem kann nicht über die lange Vertiefung schreiten, da sie länger ist als die maximale Schrittweite[20] des Roboters. Der hinter dieser Vertiefung liegende Pfad kann nicht erreicht werden. In der $C_{Resultat}$ Matrix werden alle folgenden Kacheln auf dem Pfad mit Null bewertet. Der Pfad wird, ähnlich wie in der Netzwerktechnik [DVM Routing Protocol-v1:RFC1075], „vergiftet" und die Bewertung in der Pfadauswahl wird erheblich verschlechtert. Es wird aber nicht der komplette Pfad vergiftet, da der vordere Bereich noch gut begehbar und nutzbar sein kann.

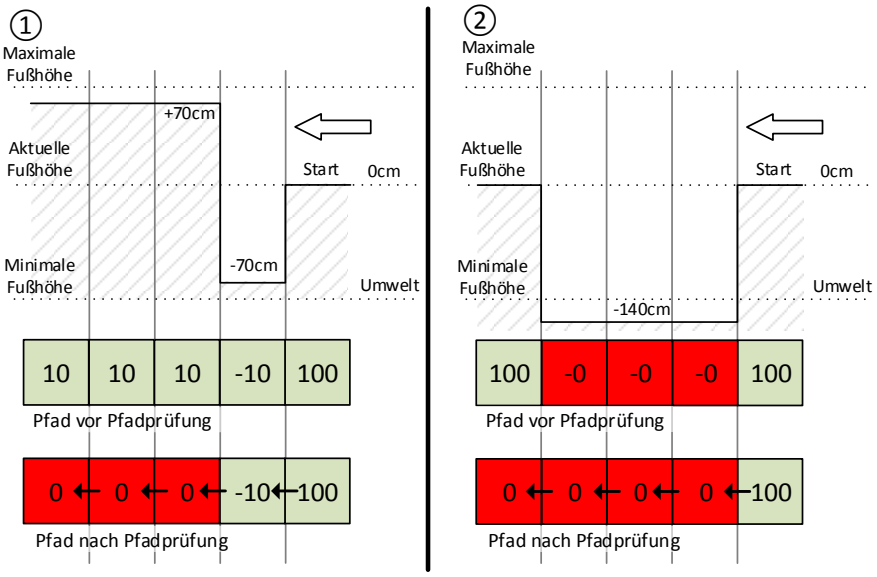


*Abbildung 4-11 Pfadprüfung*

Liegt der Wert $Kachel\delta_n$ über 100, stellt dies die maximal erreichbare Höhe dar. Tritt dies nur einmal auf, ist der dahinterliegende Pfad nicht erreichbar und alle Folgekacheln werden vergiftet bzw. mit Null in der $C_{Resultat}$ Matrix bewertet (Abbildung 4-11 ①). In der realen Umwelt bildet dies eine Erhebung ab, auf die nicht gestiegen werden kann. Als Folge sind auch die dahinterliegenden Kacheln nicht mehr begehbar, da sie auf diesem Pfad nicht erreicht werden können.
Dieses simple Verfahren ist nur für ein Laufsystem sinnvoll, da beispielweise Fahrzeuge sich auf

[20] Die maximale Schrittweite beträgt in diesem Beispiel 1,2m und die Kachelkantenlänge 40cm. Drei Kacheln ergeben 1,2m. In der realen Anwendung ist die maximale Schrittweite von der gewählten Schulterhöhe des Roboters abhängig und damit auch die Anzahl der überschreitbaren Kacheln.

geschlossenen Pfaden bewegen müssen. Es könnten noch mehr Prüfungen am Pfad vorgenommen werden. Dieses Modul dient aber als Vorverarbeitung, um schwerpunktstabile Trittpositionen zu finden und nicht einen komplexen Pfad.
Bei der Bewertung der Umweltmatrix $A_{Umwelt}$ wurden *übersteigbare* schräge oder senkrechte Kacheln und *nicht* übersteigbare Wände gleichermaßen als nicht begehbare Vertiefung gekennzeichnet. Bedingt durch Funktionsweise des verwendeten Sensors werden trotzdem Wände als Hindernisse erkannt. Der Sensor (3.4.3.1) liefert eine 2.5D Punktwolke und jedes Hindernis wirft einen Sichtschatten, der die Umwelt dahinter verdeckt. Hat die Umweltabstraktion keine Fläche in einem Subwürfel gefunden, wurde auch dies als nicht erreichbare Vertiefung bewertet (-0). Erfasst der Sensor eine Wand, werden hinter der Wand (durch den Sichtschatten) keine Kacheln mehr erfasst. In dieser Abstraktion liegen damit mehrere Vertiefungen hintereinander. Die Pfadprüfung interpretiert dies als zu lange Vertiefung, über die nicht gestiegen werden kann (wie in Abbildung 4-11 ②).
Liegt die Wand am Ende des Erfassungsbereichs des Sensors, wird sie vorerst als einzelnes Loch fehlinterpretiert, aber ist noch zu weit entfernt, um problematisch zu werden. Das System nähert sich dieser vermuteten Vertiefung an und erkennt nach drei Schrittfolgen, dass es sich um eine nicht übersteigbare Vertiefung handelt und weicht aus. Mit dem obigen Algorithmus wird dann der dahinterliegende Pfad „vergiftet“. Als Folge wird die Wand als nicht überwindbar erkannt. Das andere Extrem ist beispielweise der senkrechte Absatz einer Treppenstufe. Diese Fläche wird vom System als eine Vertiefung interpretiert, bei dem der Grund nicht erreichbar ist. Mit dem obigen Verfahren schreitet das System über den senkrechten Absatz.

#### 4.3.3.1.2 Kopffreiheitsprüfung

Alle vorherigen Verfahren haben nur Kacheln untersucht, die höchstwahrscheinlich den Boden oder begehbare Hindernisse darstellen. Dabei wird noch nicht berücksichtig, ob der Roboter unter Hindernissen hindurchgehen kann. Dafür werden jeweils drei Subwürfel (4.1.3) geprüft, die senkrecht über einer begehbaren Pfadkachel liegen. Für die Umweltmatrix wurde bereits die Punktwolke des Sensors in Subwürfel unterteilt. In diesem Modul wird nur geprüft, ob die Anzahl von Messpunkten in einem Subwürfel einen definierten Schwellwert überschreitet. Ist dies der Fall, wird der dahinterliegende Pfad vergiftet, wie in der Pfadprüfung (4.3.3.1.1) beschrieben. Dieser Schwellwert (mehr als 25 Messpunkte) wurde experimentell ermittelt und nimmt linear mit der Entfernung des Subwürfels vom Sensor (Nullpunkt des Sensors) ab. Dies ist bedingt durch die Funktionsweise des verwendeten aktiven optischen Sensors (3.4.3.1). Je näher ein Objekt am Sensor ist, desto mehr Messpunkte werden für ihn in der Punktwolke abgebildet.
Für das Laufsystem bzw. den Hauptkontroller ist die Identität eines Hindernisses nicht relevant. In dieser Betrachtung ist nur wichtig, ob sich im Bereich der Kopffreiheit ein Objekt befindet. Dieser Bereich wird durch drei Subwürfel definiert, da dies der maximalen Höhe des Roboters entspricht. Um den Roboter durchsetzungsfähig gegenüber seiner Umwelt zu halten, wird dieser Schwellwert definiert. Der Wert wird nur so groß gewählt, dass beispielsweise kleine Äste, ein Blatt oder Sensorrauschen den Pfad als noch begehbar bewerten. Problematisch sind bei diesem Vorgehen

Hindernisse, die aus vielen zusammenhängenden kleinen Objekten bestehen und aufgrund ihrer Masse weggeschoben werden könnten. Ein Beispiel wäre ein dünner Ast mit vielen Blättern. Dieser wird als Hindernis interpretiert und der Roboter umgeht dies. Eine nicht realisierte Lösung könnte es sein, die Punktwolke auf Cluster zu untersuchen, um viele kleine Objekte von massiven großen Objekten zu unterscheiden. Dabei ergeben sich aber neue Probleme, wie beispielsweise ein massiver Baumstamm, der von Blättern teilweise verdeckt wird. Hierzu gibt es wenig Untersuchungen oder Lösungen. Es ist fraglich, ob in einer späteren Entwicklungsstufe auf eine rudimentäre Identifizierung und Klassifizierung von Objekten verzichtet werden kann.
*Anmerkung: In einem Demonstrationsvideo des Boston Dynamics® LS3 (3.2.3), wird demonstriert, wie der Roboter eine dichte Hecke durchschreitet, ohne dem Hindernis auszuweichen. Leider ist nicht bekannt, wie dieses Problem gelöst wurde. Eine Vermutung des Autors ist, dass hier die „Follow the Leader" Funktion genutzt wird. Der Roboter könnte dabei stumpf dem Pfad des „Leaders" gefolgt sein.*

#### 4.3.3.1.3 Pfadberechnung

Um die Koordinaten in den Matrizen auf einem geradlinigen Pfad effizient auszuwählen, wird auf Winkelfunktionen verzichtet. Ausgehend von der gewählten Randkachel wird die Steigung iterativ addiert. Die vereinfachte Steigung berechnet sich über die Koordinate der Fußposition, dividiert durch die gesamte Kachelanzahl der jeweiligen Achse. Das Verfahren bricht ab, sobald die Fußpositionskoordinaten erreicht werden.

$$x_{St} = \begin{cases} \frac{x_{Fußposition}}{x_{\max Kachel}} \; f\ddot{u}r \left(x_{Fußposition} - x_{Randkachel}\right) > 0 \\ -\frac{x_{Fußposition}}{x_{\max Kachel}} \; f\ddot{u}r \left(x_{Fußposition} - x_{Randkachel}\right) \leq 0 \end{cases}$$

$$y_{St} = \begin{cases} \frac{y_{Fußposition}}{y_{\max Kachel}} \; f\ddot{u}r \left(y_{Fußposition} - y_{Randkachel}\right) > 0 \\ -\frac{y_{Fußposition}}{y_{\max Kachel}} \; f\ddot{u}r \left(y_{Fußposition} - y_{Randkachel}\right) \leq 0 \end{cases}$$

$$x_{Temp} = 0; y_{Temp} = 0$$

$$DO$$
$$x = FLOOR\left(x_{Randkachel} + x_{Temp}\right), y = FLOOR\left(y_{Randkachel} + y_{Temp}\right)$$
$$IF\ x\ \neq x_{Fußposition}\ THEN\ x_{Temp} = x_{Temp} + x_{St}$$
$$IF\ y\ \neq y_{Fußposition}\ THEN\ y_{Temp} = y_{Temp} + y_{St}$$
$$WHILE\ \left(x\ \neq x_{Fußposition}\ AND\ y\ \neq y_{Fußposition}\right)$$

Die Koordinaten einer Kachel auf dem Pfad sind die Werte x und y. Diese werden mit der Funktion Floor bzw. einem cast auf Integer gerundet, um diese direkt als Index zu nutzen. Die Variablen $x_{St}$ und $y_{St}$ stellen die anteilige Steigung pro Kachel dar und werden über $x_{Temp}$ und $y_{Temp}$ aufsummierten. Dieses Näherungsverfahren kann bei geringer Geradensteigung zu einer Mehrfachauswahl einer Kachel führen. Dies tritt aber nur in direkter Folge auf und kann durch einen einfachen Vergleich des

vorherigen x,y Tupels abgefangen werden. Zudem zeigen die Pfade / Korridore einen starken Aliaseffekt.

#### 4.3.3.1.4 Zusammenfassung Laufrichtungsplanung

Das gesamte Verfahren wird bei jeder Schrittfolge erneut ausgeführt, womit auch die Umwelt neu bewertet wird. Damit wird auch die Pfadauswahl immer neu getroffen. Beim Durchschreiten einer Umwelt ergeben sich Pfade, die dem Charakter eines Polynomzugs ähneln. In der Arbeit von Köchritz [Köc05] wird dies als „Smoothness“ bezeichnet und beschreibt das resultierende Verhalten.
Die Torsoneigung des Roboters wird in der zweistufigen Laufrichtungsplanung (4.3.3.1) unterschiedlich behandelt. In der erste Stufe des Umweltbewertungsverfahrens wird die reale Neigung ignoriert und die Bewertung des Höhenunterschieds ist von der Neigung des Torsos abhängig. D.h. läuft der Roboter beispielsweise auf einer langen glatten Rampe bergauf, werden die Kacheln vor dem Roboter mit 100% als ideal gewertet. Obwohl die Kacheln topografisch betrachtet höher liegen, da die Erreichbarkeit durch die aktuelle Fußposition bewertet wird. In einigen Simulationen hatte dieses Verhalten dazu geführt, dass der Roboter beispielweise die Talsohle einer Schlucht bis zum Ende folgte. Wohingegen der Roboter bei zum Horizont nivellierten Kacheln einen Zickzackkurs einnimmt, da er Kacheln auf gleicher Höhe bevorzugt. Im zweiten Schritt des Verfahrens (4.3.3.1), bei dem zu schräge Kacheln aussortiert werden, müssen auf den Horizont nivellierte Kacheln verwendet werden.
Ein interessanter Aspekt dieses simplen Verfahrens ist die Beeinflussbarkeit des Verhaltens. Im Beispiel in Abbildung 4-9 wird nur die Entfernungsmatrix mit $\beta = 1{,}3$ multipliziert. Damit werden entfernte Kacheln höher gewichtet und der Roboter weicht der Wand (Abbildung 4-9 ① hinten) nur „zögerlich“ aus. Zudem vermeidet der Roboter es zu steigen. In einer weiteren Entwicklungsstufe könnten diese Skalare gezielt dynamisch eingesetzt werden. Beispielsweise könnte bei starkem Sichtschatten die Umweltmatrix höher bewertet werden. Dies hätte zur Folge, dass der Roboter Höhenunterschiede bevorzugt, um einen besseren Überblick seiner Umwelt zu erhalten. Die Skalare eignen sich zudem besonders für die Anbindung von Machine-Learning-Verfahren. Mit einer mehrschichtigen KI [Pfe07] würde das System selbständig Strategien entwickeln. Diese Strategien würden ein Verhaltensset abbilden und in bestimmten Situationen ein bestimmtes Verhalten wählen.
Das vorliegende Verfahren, mit fest modellierten Skalaren, kann keine Schleifen im Pfad verhindern. Befindet sich der Roboter in einer Umgebung ohne Ausweg, beispielsweise ein geschlossener Raum, läuft der Roboter ständig an den Wänden entlang. Eine Analogie wäre ein Raubtier im Käfig, das ständig die Gitter auf eine Schwachstelle prüft. Für den Einsatzzweck des Roboters ist dies erwünscht. Aufgrund der Ungenauigkeit des verwendeten aktiven 3D Sensors wird beispielsweise ein Busch mit vielen Blättern auf ca. 3-4m Abstand als ein massives Hindernis interpretiert. Erst ab circa einem Meter werden die Blätter einzeln aufgelöst und so klein, das sie vom RANSAC Algorithmus nicht mehr als Fläche erkannt werden. Für den Roboter verschwindet damit das Hindernis bei Annäherung.
Zudem kann das Verfahren ein sog. „Wormhole“ [And09] bzw. eine spezielle Art der Sackgasse nicht erkennen. Das Wormhole wird zum ernsthaften Problem, wenn der Roboter in eine enge und steile Schlucht läuft und er bei der Richtungsentscheidung nicht das Ende der Schlucht sehen kann. Ist diese

Schlucht eine Sackgasse, kann der Roboter nicht umdrehen – weil die Schlucht zu eng ist - und er kann nicht weit rückwärts laufen – weil die Sensoren nach vorn ausgerichtet sind. Für diese Situation konnte keine Lösung gefunden werden. Auch Kolter und Team [And09] fanden keine Lösung und versuchten dies über markante Landschaftsformationen vorherzusagen.
Das Verfahren hat aber auch starke Ungenauigkeiten. Beispielweise bei einem Pfad, der über 45° Winkelgrad geplant wird. Da nur die betrachteten Kacheln gezählt werden, ist die tatsächliche Strecke länger als in dieser Abstraktion. Diese Ungenauigkeit wirkt sich aber nicht stark aus, da der Roboter sich nach 3-4 Schrittfolgen auf die 45° Winkelgrad dreht und dann die Umwelt wieder rechtwinklig bewertet.
Eine Prüfung auf die Breite des Roboters ist nicht mehr nötig. Die Zehenspitzen des Roboters bilden die maximale Breite des Roboters. Die maximale Breite des Roboters entspricht der Pfadbreite. Dadurch kann kein zu schmaler Pfad ausgewählt werden[21]. Problematisch sind wieder schräge Pfade. Durch den Aliaseffekt des Pfads könnten Hindernisse in den Pfad ragen. Wie oben bereits erwähnt, schwächt sich dieses Problem mit der Drehung des Roboters ab.
Diese und andere Schwächen des Verfahrens werden aufgrund des guten Laufzeitverhaltens und der Parallelisierbarkeit in Kauf genommen. Das vorgestellte Verfahren dient nur der Vorselektion eines Pfades und ist die Vorstufe zur eigentlichen Trittpositionsplanung in Kapitel 4.3.3.2.

#### 4.3.3.2 Trittpositionsplanung

{Architekturnamen: *Step and Body Planner, Pitch and Roll Planner*}
In dieser Trittpositionsplanung werden *für jedes Bein* mechanisch erreichbare Trittpositionen ermittelt. Dafür wird wieder ein Bewertungsverfahren verwendet, das die „Güte“ der Kacheln ermittelt. Die Matrizen für dieses Bewertungsverfahren haben nur die Größe der Beinreichweite und sind vom selben Matrizentyp. Im Gegensatz zur Laufrichtungsplanung (4.3.3.1) wird hier mit einer höheren Kachelauflösung und Flächenerkennungsgenauigkeit (RANSAC-Plane Iterationen) gearbeitet. Die Ergebnisse dieses Verfahrens werden an die Stabilitätsplanung (4.3.3.3) übergeben.
Damit die vorherige Laufrichtungsplanung (4.3.3.1) möglichst schnell eine Richtungsentscheidung treffen kann, wird dort die komplette Umwelt mit einer groben Kachelauflösung (11 x 11 Kacheln á 40cm²) erfasst. Der verwendete RANSAC-Plane (4.3.2.2) Algorithmus wurde auf wenige Iterationen begrenzt, damit er schnell aber ungenau arbeitet[22].
Für die Trittpositionsplanung ist diese Auflösung zu grob und die Flächenerkennung noch zu ungenau, um real einen Roboterfuß auf diese Flächen zu setzten (Abbildung 4-12 ①). Weiterhin ist die Auflösung zu grob für die Stabilitätsplanung (4.3.3.3).
Aus diesen Gründen wird die Auflösung auf 20cm Kachelkantenlänge erhöht. Es werden aber nur noch

[21] Dies ist nicht allgemeingültig und gilt nur für diesen Roboter.
[22] Wäre die komplette Umwelt mit erhöhter Genauigkeit bearbeitet worden, würde ein Durchlauf mit dem RANSAC-Plane Algorithmus ca. 30-50 Sekunden dauern (parallelisierter aber nicht optimierter C# Code, auf Intel 4770T)

Kacheln betrachtet, die in der geplanten Laufrichtung (Resultat der Laufrichtungsplanung 4.3.3.1) liegen und die von den Roboterbeinen wirklich erreicht werden können.

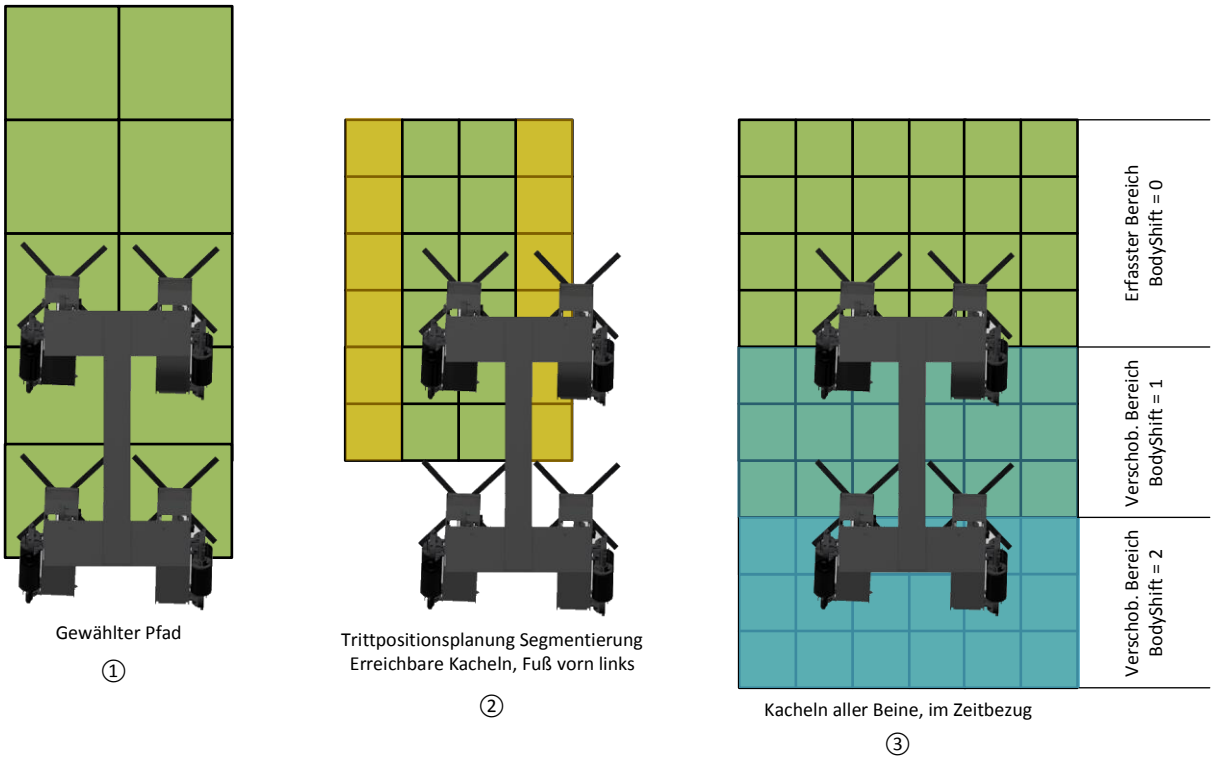


*Abbildung 4-12 Segmentierung und Zeitbezug*

In der Abbildung 4-13 ② wird die erhöhte Auflösung für den vorderen linken Fuß dargestellt. Die dargestellten Kacheln zeigen in die zuvor ermittelte Richtung und werden von 40 cm auf 20 cm halbiert. Um auch Ausfallschritte, die den Roboter abstützen können, zu ermöglichen wird der Kachelpfad rechts und links um jeweils eine Kachel erweitert. Dieses Modell wird für jedes Bein erstellt und technisch über Arrays realisiert.

Die Umwelt wurde schon für die Laufrichtungsplanung (4.3.3.1) in einer Punktwolke erfasst und in 40cm³ Subpunktwolken unterteilt. Für die genauere Flächenerkennung in diesem Verfahren werden nur Subpunktwolken verwendet, die um den Pfad liegen. Das heißt, es wird ein dreidimensionaler Korridor aus Subpunktwolken selektiert. Die selektierten 40 cm³ Subpunktwolken werden mittig in 20cm³ Würfel unterteilt. Diese kleineren Würfel aus der Punktwolke werden mit dem RANSAC-Plane[23] Verfahren mit erhöhter Genauigkeit untersucht. Bedingt durch den fest montierten Umweltsensor kann der Roboter AMEE nur die Umwelt vor dem Laufsystem erfassen. Dadurch müssen pro Schrittfolge nur 144 Subpunktwürfen á 20cm³ untersucht werden. In Abbildung 4-12 ③ werden die überlappenden Matrizen aller Beine dargestellt. Für die Vorderbeine sind 4 x 6 Kacheln in Laufrichtung

[23] Ca. 800-1000 Iterationen

erreichbar. Zudem wird die zu untersuchende Höhe auf 6 Subpunktwürfel (á 20cm³) begrenzt (6 x 0,2 = 1,2m -> nutzbare AMEE Höhe = max. 1,1m). Durch diese Einschränkungen kann mit dem RANSAC-Plane Algorithmus mit ca. 1000 Iterationen erheblich genauer und schnell nach Flächen gesucht werden.
Wie in der Laufrichtungsplanung (4.3.3.1) werden die gefundenen Flächen über Gewichtungsmatrizen bewertet und anschließend genauer ausgewertet.
**Umweltmatrix**: Es wird für jeden Fuß eine Umweltmatrix angelegt. Die Bewertung der Kacheln erfolgt wieder nach Güte der Erreichbarkeit in Prozent, wie im Kapitel 4.3.3.1. Auch hier werden Subwürfel, in denen *keine* Fläche gefunden wurde und Flächen über +/-20° Neigung, mit Null bewertet. Flächen, die zu hoch oder zu niedrig sind, um von den Roboterfuß erreicht zu werden, erhalten auch eine Bewertung mit Null. Die Bewertung bildet wieder eine Momentaufnahme der Umwelt ab und repräsentiert die Umwelt als eine Art Kostenkarte mit Flächen, auf die der Roboter treten kann.
Bedingt durch den Umweltsensor kann der Roboter nur die Umwelt erfassen, die vor ihm liegt. Es soll dem System aber auch möglich sein, hinter die aktuelle Fußposition zu treten. Es kann in kritischen Stabilitätssituationen vorteilhaft sein, den Schwerpunkt nach hinten abzustützen. Alle Kacheln - die hinter der Brust des Roboters liegen - sind Flächen, die in den vorherigen Schrittfolgen erfasst wurden. Dies wird auch in Abbildung 4-12 ③ dargestellt, wobei nach jeder Schrittfolge (Body Shift = n) die erfassten Flächen um den Betrag der Torsobewegung in Laufrichtung verschoben werden. Nur die Kacheln mit Body Shift = 0 werden zu diesem Zeitpunkt real erfasst. Für die Vorderbeine besteht die vordere Hälfe der Umweltmatrix aus realen Messwerten. Der hintere Teil der Umweltmatrix besteht aus den Daten der vorherigen Bewegung des Torsos und ist um den Bewegungsvektor verschoben. Dieser Bewegungsvektor des Torsos wird aus dem Reaktive-Layer (4.3.4) abgerufen. Bei den Hinterbeinen besteht die Umweltmatrix komplette aus „alten“ Daten, die vorher erfasst wurden und verschoben sind. Problematisch ist dabei das Rutschen der Füße auf dem Untergrund. In dieser Echtwicklungsstufe enthält das System noch keine echte Odometrie (2.2.5). Das bedeutet: Wenn der Roboter rutscht, ist die alte Umweltmatrix um diese Strecke verschoben und damit ungenau oder sogar unbrauchbar. In der nächsten Entwicklungsstufe soll der Kinect® V2 Sensor für die Odometrie verwendet werden.
Im initialen Startzustand des Roboters werden alle „alten“ Kacheln als ideal angenommen. Wird das System gestartet, können keine alten Umweltdaten verschoben werden, da nur aktuelle Daten vorliegen. Die Konsequenz dieser Designentscheidung ist, dass der Roboter auf relativ glattem Untergrund eingeschaltet bzw. initialisiert werden sollte. Ist dies nicht der Fall, muss der Reactive-Layer (4.3.4) über seine maximalen Torsoneigungswinkel eingreifen, um ein Kippen zu verhindern.
**Richtungsmatrix**: Auch die gewünschte Richtung wird äquivalent zur Laufrichtungsplanung modelliert. Nur wird hier ein Abweichen von der gewünschten Richtung schwächer (=schlechter) bewertet als zuvor. Diese Matrix hat eine weniger abstrakte Bedeutung. Eine Abweichung zur Seite stellt direkt eine horizontale Drehung des Beins aus der Laufrichtung dar. Der Roboter AMEE kann zwar jedes Bein seitlich drehen, aber verfügt über kein vertikal schwenkbares Schulterkugelgelenk. Bedingt dadurch sind nur kleine unterschiedliche Drehwinkel zwischen den Beinen möglich. Bei einer Bewegung des

Torsos würden die Beine am gleichen Fixpunkt mit unterschiedlichen Winkeln ziehen. Tests mit der realen Mechanik haben eine maximale Abweichung von ca. 15° ergeben, wobei der Fuß dann leicht auf der Auflagefläche rutscht. Aus diesem Grund werden seitliche Abweichungen vom Pfad, stärker „bestraft“ bzw. schwächer bewertet (Abbildung 4-13).

**Reichweitenmatrix**: Die Reichweitenmatrix veranlasst das System, möglichst große Schritte zu machen. Kacheln, die sich weiter entfernt von der aktuellen Fußposition und in der gewünschten Richtung liegen, werden höher bewertet. Die aktuelle Fußposition wird mit Null bewertet, da dies keinem Vorankommen entspricht. Das System wird damit „gierig“ nach einem schnellen Vorankommen.

**Resultatmatrix**: Die Resultatmatrix ist wie in der Laufrichtungsplanung (4.3.3.1) die jeweilige Multiplikation mit einem Skalar und die anschließende Addition der Matrizen. Anders als beim obigen Vorgehen wird der Wert Null aus der Umweltmatrix nicht verrechnet, sondern direkt in die Resultatmatrix übertragen und damit als nicht begehbar bewertet.

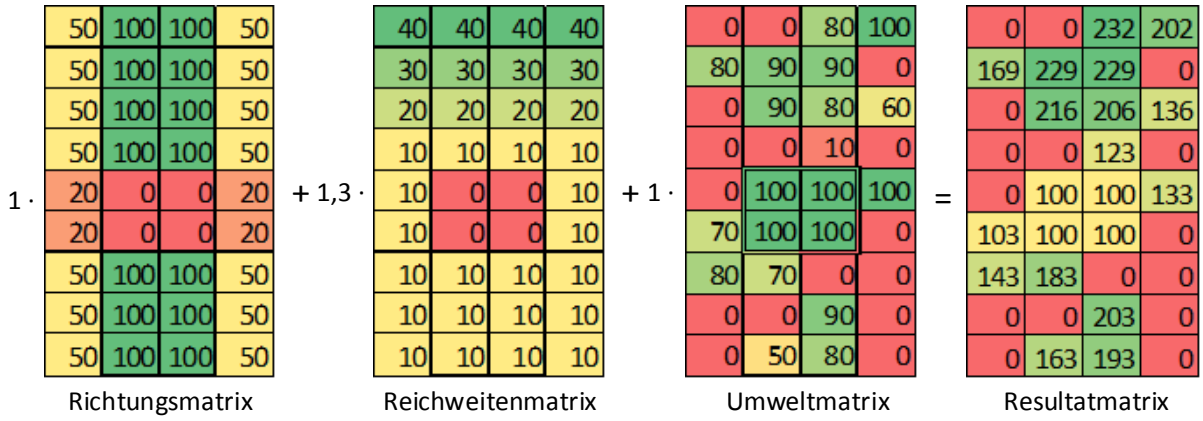


*Abbildung 4-13 Trittpositionsbewertung*

In der Abbildung 4-13 ist die aktuelle Fußposition in der Mitte der Matrizen. In der Richtungs- und Reichweitenmatrix sind es die mit Null bewerteten Felder.

**Trittpositionsmatrix**: Durch die neue Segmentierung der Umwelt entsprechen immer vier Kacheln der Fußauflagefläche. Die *Trittpositionsmatrix* ist nicht vom selben Typ. Sie bildet jeweils vier benachbarte Kacheln auf ein Element der *Trittpositionsmatrix* ab, welches der bewerteten Trittposition entspricht (Abbildung 4-14). Jeder Fuß erhält eine Trittpositionsmatrix und die Elemente werden aus der jeweiligen Resultatmatrix errechnet. Dazu wird jede mögliche überlappende 2x2 Submatrix der *Resultatmatrix* addiert und durch vier dividiert. Bei der Berechnung wird gegen einschränkende Regeln geprüft. Enthält eine Submatrix ein Element mit dem Wert Null, ist die Fußauflagefläche zu gering oder ein Hindernis befindet sich auf dieser Position. In diesen Fällen wird eine Null in die Trittpositionsmatrix eingetragen. Danach wird der Höhenunterschied der 2x2 Submatrix anhand der Umweltmatrix geprüft. Ist der Höhenunterschied so groß[24], dass er die Stabilität gefährden könnte - wird auch diese Trittposition mit Null bewertet.

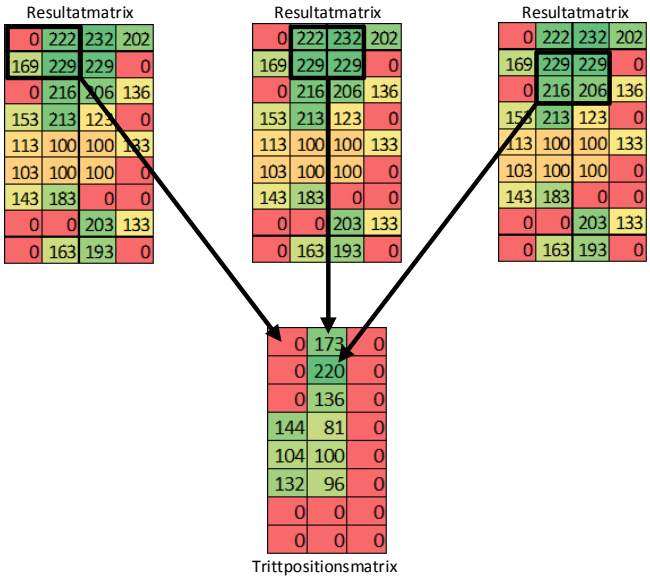


*Abbildung 4-14 Berechnung der Trittpositionsmatrix*

Der maximale Wert in der *Trittpositionsmatrix* repräsentiert die optimale Trittposition, ohne vorerst die Schwerpunktstabilität zu berücksichtigen. Die Trittpositionsmatrix wird an die Stabilitätsplanung (4.3.3.3) übergeben. Wird keine Trittposition gefunden, beginnt die Trittpositionsplanung erneut mit einem niedriger bewertetem Pfad aus der übergebenen Liste der Laufrichtungsplanung (4.3.3.1).

**Zusammenfassung**: Das Vorgehen in der vorherigen Laufrichtungsplanung (4.3.3.1) ist ein heuristisches Verfahren und abstrakt. Im Gegensatz dazu ist dieses Planungsverfahren zwar ähnlich aufgebaut, aber repräsentiert direkt mögliche Trittpositionen, die von der Mechanik angefahren werden können. Aus diesem Grund führt auch eine Bewertung mit Null zu einer nicht begehbaren Trittposition.

Für die Schwerpunktstabilität ist eine Zweipunktauflage des Fußes (durch die Zehen(3.3.4)) ausreichend. Das obige Verfahren verwirft aber schon eine Dreipunktauflage pro Fuß. Um zu entscheiden, welcher Zeh die Hauptlast tragen wird, muss der gemeinsame Masseschwerpunkt des Roboters bekannt sein. Diese Berechnung kann aber erst in der nächsten Stufe stattfinden. Wie in der Kapiteleinleitung erwähnt, soll dieses planende Modul möglichst schnell stabilitätsoptimale Trittpositionen für den reaktiven Teil bereithalten. Tritt ein stabilitätskritischer Zustand ein und die

[24] Experimentell wurden hier 3-5cm ermittelt.

laufzeitintensivere Stabilitätsberechnung ist noch nicht abgeschlossen, kann schon einer diese Positionen ausgewählt werden. Der Roboter würde wahrscheinlich nicht stabilitätsoptimal stehen, aber der Sturz würde stark verlangsamt und das System hätte Zeit die Stabilitätsberechnung zu beenden. Dies ist nur für den Notfall vorgesehen und streckt das experimentell ermittelte Zeitfenster von circa 500ms auf 2-3 Sekunden.

#### 4.3.3.3 Stabilitätsplanung

{Architekturnamen: *Gait Selection-, Leg Selection-, Body Shift Controller*} In den beiden Vorstufen wurden begehbare Trittpositionen gesucht. Bei dieser Suche wurde aber noch nicht die Stabilität des Roboters betrachtet oder bewertet.

Es werden die sortierten Trittpositionslisten aus der Trittpositionsplanung (4.3.3.2) verwendet. Durch die Kombination aller vorverarbeiteten möglichen Trittpositionen ergeben sich Posen (Stellungen der Glieder) für den Roboter. Diese Posen und ihre Bewegungsabläufe werden auf Schwerpunktstabilität getestet. Aufgrund dieser Posen wird auch entschieden, welches Laufmuster vorteilhaft wäre. Das Ergebnis ist eine bewertete, sortierte und geprüfte Liste mit mehreren Trittpositionen und Bewegungsabläufen, die dem Upstream-Reactive-Layer (4.3.4) übergeben wird.

**Schwerpunktmodell**: Der Roboter wird in diesem Modell durch die Verteilung von Massepunkten beschrieben. Die Massepunktpositionen werden über eine Unterteilung des Roboters in geometrische Grundkörper bestimmt (Abbildung 4-15 ②). Jeder Massepunkt ist der Massenschwerpunkt des jeweiligen Grundkörpers und in ein roboterglobales Koordinatensystem eingeordnet.

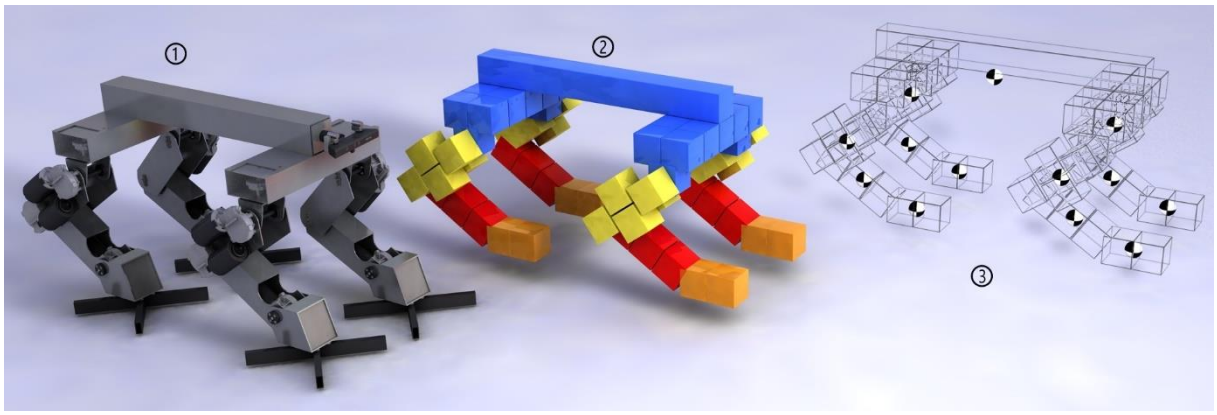


*Abbildung 4-15 Abstraktion des Schwerpunktmodells*

Der Roboter wurde als Sensorplattform konzipiert und kann in verschiedenen Konfigurationen betrieben werden. Eine Konfigurationsvariante wurde in Abbildung 4-15 visualisiert. Dieses Modell erlaubt ein einfaches Hinzufügen von Massepunkten, falls Bauteile verändert oder Sensoren hinzugefügt werden. In der Initialisierungsphase des Systems werden die einzelnen Massepunkte von starr verbundenen Bauteilen zu einem Massepunkt verrechnet. In dieser Abstraktion sind dies der Torso (Abbildung 4-15 ② blau) und die einzelnen Glieder der Beine (Oberschenkel gelb, Schienbein

rot, Fuß orange). Zur Laufzeit des Systems werden nur noch die zusammengefassten Massepunkte betrachtet ③, da sich der gemeinsame Massepunkt mit jeder Pose des Roboters verändert.
Die dreidimensionale Berechnung der Massepunkte erfolgt über diese Gleichungen [Mil06].

$$x_c = \frac{\sum(m_k \cdot x_k)}{\sum m_k}, y_c = \frac{\sum(m_k \cdot y_k)}{\sum m_k}, z_c = \frac{\sum(m_k \cdot z_k)}{\sum m_k}$$

$x_c, y_c, z_c$ sind die Koordinaten des gemeinsamen Massepunkts. Die Variablen $x_k, y_k, z_k$ stellen die Koordinaten der einzelnen Massepunkte dar und $m_k$ symbolisieren die Masse jedes Massepunktes.
**Stabilitätsermittlung**: Für jeden Fuß liegt eine Trittpositionsmatrix (4.3.3.2) aus der Trittpositionsplanung vor. Die Trittpositionsmatrizen sind sortierte Listen. Es werden iterativ alle Kombinationsmöglichkeiten zwischen den Trittpositionen der Beine betrachtet und die einzelnen Körperhaltungsposen auf Schwerpunktstabilität getestet. Zur Verdeutlichung des Ablaufs in diesem Modul wird die Nummerierung aus Abbildung 4-16 verwendet.
Das System beginnt im Stand des Laufsystems. Für ein dynamisches Laufmuster (4.1.5) ist es nötig, ein initiales Schrittmuster ⓪ als Startzustand für den ersten Schritt vorzugeben. Das System beginnt die Planung mit dem Fast-Walk (4.1.5) Schrittmuster, bei dem zwei diagonal gegenüberliegende Füße auf dem Boden verbleiben (n=2).

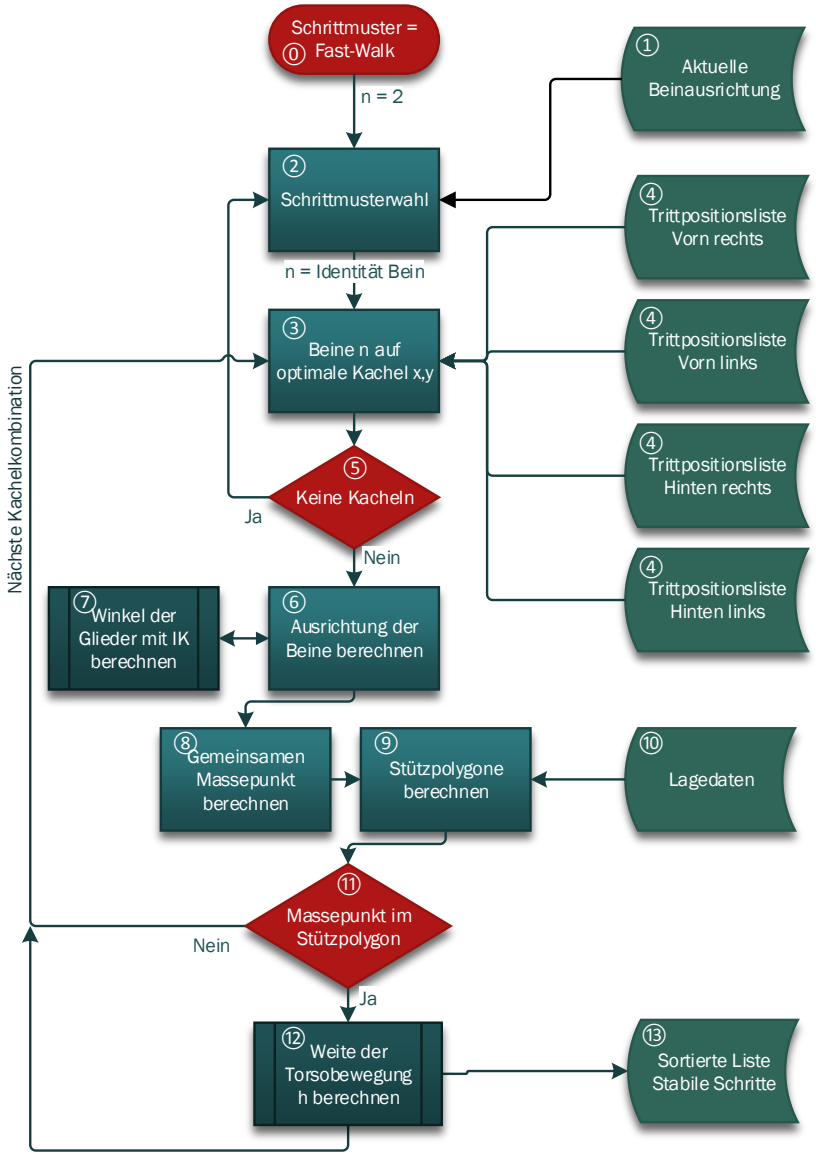


*Abbildung 4-16 Ablauf Stabilitätsplanung*

Die Schrittauswahl ② kann in dieser Entwicklungsstufe zwischen den zwei Schrittmustern (4.1.5) entscheiden oder auf neue Daten warten. Ist die initiale Phase abgeschlossen, wird hier geprüft, in welchem Schrittmuster sich das System befindet und welches für den nächsten Schritt des Roboters sinnvoll wäre. Wurden beispielweise alle Trittpositionskombinationen im Fast-Walk-Modus (zwei Kacheln) getestet ⑤ und keine schwerpunktstabile Trittposition gefunden, beginnt das System mit dem Save-Walk-Modus (vier Kacheln). Liegen dem System danach keine neuen Kacheln oder

Lagedaten des Torsos vor, geht das System in einen Wartezustand[25] (nicht dargestellt). Treffen veränderte Daten ein, beginnt der Vorgang erneut. Für jeden kompletten Durchlauf werden die aktuellen Beinpositionen aus dem *Upstream-Reactive-Layer* (4.3.4) abgerufen und für jedes Bein der noch mögliche Schrittradius ermittelt.
In der nächsten Stufe ③ werden die optimalen Kacheln (Ergebnisse der Trittpositionsplanung 4.3.3.2) aus den jeweiligen Trittpositionslisten④ absteigend gewählt. Je nach Schrittmuster sind dies zwei oder vier Kacheln. Die beteiligten Beine werden virtuell auf die Trittpositionen gesetzt ⑥. Befindet sich ein Bein schon in der maximalen Auslenkung und in der geplanten Laufrichtung, wird dieses Bein auf seiner Position belassen.
Da sich die Trittpositionsmatrizen überlappen, muss eine Kollision der Beine untereinander verhindert werden. Dafür wird eine virtuell belegte Kachel (mit einem Fuß) für die anderen Beine ausgeblendet. Zudem werden für die Vorderbeine die Kacheln ausgeblendet, die in Bewegungsrichtung vor der belegten Kachel liegen. Bei den Hinterbeinen werden die Kacheln hinter der belegten Kachel ausgeblendet.
Die dabei erzeugte Pose ist die Pose am Ende eines Bewegungsablaufs. Sie wird virtuell eingenommen. Die Winkel zwischen den Beingliedern und zum Robotertorso werden mit der beschleunigten inversen Kinematik [Ruh11] berechnet ⑦. Durch die nun bekannten Winkel der Beinglieder kann in der Stufe ⑧ ein gemeinsamer Massepunkt für den kompletten Roboter – für diese eine Pose – berechnet werden. Das Stützpolygon ⑨ (2.2.2) wird von den Füßen auf dem Boden aufgespannt und die Berechnung erfolgt mit einer optimierten Variante des *Graham Scan* [Rou98]. Damit wird die konvexe Hülle des Polygons (über die Auflagepunkte - Füße) bestimmt. Dies ist nötig, da die Zehen ein komplexes Polygon mit innenliegenden Punkten bilden. Danach wird der vorher berechnete gemeinsame Massepunkt auf die Ebene des Stützpolygons projiziert. Für diese Projektion wird der senkrechte Winkel (zur Erdanziehung) aus den aktuell anliegenden Lagedaten ⑩ des Robotertorsos verwendet.
Um zu bestimmen, ob die angenommene Pose schwerpunktstabil ist, wird geprüft, ob sich der projizierte gemeinsame Massepunkt im Stützpolygon befindet ⑪. Dies wird über einen optimierten *Umlaufszahl*[26] Algorithmus (*winding number*), aus der Arbeit von Hormann und Agathos [Hor01], erreicht. Befindet sich dieser Punkt nicht im Stützpolygon, ist die Pose nicht schwerpunktstabil und das gesamte Verfahren beginnt erneut (bei ③) mit der nächsten Kachelkombination. Ist der Punkt im Stützpolygon, wird ermittelt, wie weit der Torso schwerpunktstabil bewegt werden kann ⑫.[27]
Die maximale mögliche Bewegung des Torsos über die Füße und in Bewegungsrichtung wird heuristisch ermittelt ⑫. Dafür wird der Torso virtuell um jeweils eine Kachel in die Bewegungsrichtung

[25] Dieser wird durch den Heartbeat/Watchdog-Controller (4.3.5) überwacht.
[26] Das Verfahren ist für die jetzige Vorgehensweise überdimensioniert und berücksichtig viele Ausnahmefälle, die durch die Konstruktion nicht vorkommen können. Wird das System aber mit mehr als vier Zehen pro Fuß ausgestattet, kann beispielweise ein konvexes Polygon nicht mehr garantiert werden.
[27] Eine hilfreiche Zusammenfassung der physikalische Hintergründe bietet das „Handbook of Robotics [Kaj08]“ ab Kapitel 16.5 und die darin zitierten Veröffentlichungen.

solange verschoben, bis der gemeinsame Massepunkt nicht mehr im Stützpolygon liegt. Der letzte schwerpunktstabile Punkt ist die maximal mögliche Bewegung des Torsos.
Für den Fast-Walk (4.1.5) werden dieser Wert und Trittpositionskoordinaten in eine sortierte Liste ⑬ eingetragen. Die Sortierung erfolgt absteigend nach Länge der Torsobewegung. Die Liste wird für den Abruf durch den Upstream-Reaktive-Layer (4.3.4) bereitgehalten. Das gesamte Verfahren beginnt nun mit einer neuen Kachelkombination bei Punkt ③.
Der Save-Walk (4.1.5) muss an dieser Stelle weiter überprüft werden. (Nicht in Abbildung 4-16 dargestellt.) Bei diesem Laufmuster verändern sich die Auflagepunkte bzw. die Eckpunkte für das Stützpolygon mehrfach pro Schrittfolge. Die resultierende End-Pose wird über vier Einzelbewegungen eingenommen, da alle vier Beine bewegt werden und dann erst der Toroso bewegt wird. Diese Einzelbewegungen werden jeweils iterativ auf Schwerpunktstabilität getestet, wie zuvor im Fast-Walk. Die Reihenfolge der zu testenden Einzelbewegungen wird aufsteigend[28] nach möglicher Schrittweite gewählt. Wird dabei eine Einzelbewegung entdeckt, die nicht schwerpunktstabil ist, wird die Reihenfolge variiert. Bleiben alle Kombinationen erfolglos, wird die komplette Pose verworfen und das gesamte Verfahren beginnt mit der Auswahl neuer Kacheln bei Punkt ③. Wird eine Reihenfolge gefunden, bricht das Verfahren ab und trägt die Weite der möglichen Torosobewegung, die Trittpositionen und die Einzelschrittreihenfolge in die sortierte Liste ⑬ ein. Das gesamte Verfahren beginnt wieder mit einer neuen Kachelkombination bei Punkt ③.
*Anmerkung: Bei der Bewegungsüberprüfung wird die Bewegung auf dem Raster, der Kachelwelt, nur iterativ betrachtet. Es wird nur ein Massepunkt alle 20cm berechnet. Damit wird nicht der komplette Bewegungsspielraum des Toros ausgenutzt. Diese Designentscheidung ist durch mehrere Punkte begründet. Über 70% der Robotermasse liegt in den Beinen. Wird der Torso verschoben oder ein Bein bewegt, bewegen sich die Massepunkte der Beine nicht (mechanisch) linear zum Torsomassepunkt. Sind beispielweise die Beine eingeknickt und werden von einer vorgreifenden Position nach hinten bewegt, wandert der gesamte Massepunkt schneller als der Betrag der Bewegung selbst. Die gesuchte Länge vom Massepunkt zum Rand des Stützpolygons kann zwar errechnet werden. Aber der Massepunkt ist abhängig von vier kombinierten Kinematiken, wodurch die Komplexität des Gleichungssystems stark ansteigt. In dem oben vorgestellten Verfahren beträgt die Länge der maximalen Torsobewegung ca. 1,2m und entspricht in diesem Modell sechs Kacheln. D.h. für jede Ermittlung werden maximal sechs Iterationen benötigt. Dieses Vorgehen kann aber nicht verallgemeinert werden.*

[28] Es ist wahrscheinlicher, eine schwerpunktstabile Einzelbewegungsfolge zu finden, wenn mit den kürzesten Bewegungen begonnen wird. In der Theorie, nimmt die Fläche des Stützpolygons im Laufe der absteigenden Folge ab. Diese Strategie könnte schneller eine stabile Bewegungsreihenfolge finden, als in umgekehrter Reihenfolge.

**Zusammenfassung**: Das vorgestellte Verfahren ist für eine Multiprozessorplattform ausgelegt. Durch die Abstraktion zu einer „Kachelwelt“ können die Berechnungen für jede Kachelkombination parallel durchgeführt werden, da die unterschiedlichen Roboterposen nicht voneinander abhängig sind. Der obige Ablauf kann als Einzelprozess modelliert werden und benötigt nur eine Koordination bei der Auswahl der Kacheln③ pro Pose[29]. Es können damit mehrere Posen und Kachelkombinationen parallel geprüft werden. Damit skaliert dieser Ansatz bis zur Anzahl der möglichen Kachelkombinationen über die Anzahl der Prozessoren.

Das vollständige Überprüfen aller Kombinationen, für beide Laufmuster, kann eine hohe Laufzeit verursachen. Dieser Brute Force Ansatz ist erst durch die Vereinfachung der Umwelt vertretbar (4.1.3). Die Komplexität dieses Ansatzes wird durch die Gewichtungsverfahren in den Vorstufen reduziert. Es werden nur begehbare Kacheln untersucht. Nur eine absolut plane Umwelt würde eine vollständig belegte Trittpostionslisten (4.3.3.2) ergeben. Mithilfe der sortierten Listen werden schon im ersten Durchlauf die Positionen mit der höchsten Güte getestet. Mit hoher Wahrscheinlichkeit werden nach wenigen Durchläufen schwerpunktstabile Trittpositionen gefunden. Wird keine Trittposition im Fast-Walk Modus gefunden, kann davon ausgegangen werden, dass es sich um stark unebenes Gelände handelt. Damit wechselt das System in den Save-Walk Modus, wobei der Roboter auch langsamer geht. Die höhere Rechenzeit wird damit nicht sichtbar und beeinflusst wenig den realen Ablauf der Bewegung.

Das Verfahren kann an die verfügbare Rechenleistung angepasst werden. Für das Laufsystem ist es zwar vorteilhaft, so viele schwerpunktstabile Trittpositionen wie möglich bereitzuhalten. Aber es ist völlig ausreichend, die Überprüfung nach beispielweise vier schwerpunktstabilen Posen abzubrechen. Im Fall eines Fehltritts würden drei alternative Posen / Trittpositionen verbleiben.

Das Verfahren berücksichtigt noch keine dynamischen Schritte (2.2.2). Das pseudodynamische Verhalten kann über eine vergrößerte Hülle (in Laufrichtung) des Stützpolygons erreicht werden (Abbildung 2-1). Befindet sich eine Trittposition zwischen der erweiterten Hülle und dem Stützpolygon, handelt es sich um einen pseudodynamischen Schritt. Durch die implizite Richtungsangabe im Gewichtungsverfahren würde der Roboter dabei in die Laufrichtung kippen. Die Vergrößerung der erweiterten Hülle darf nur so groß sein, dass der Roboter in der Zeit des Kippens diese Position auch mechanisch erreichen kann. Dabei muss das System schwerpunktstabile Posen von dynamischen Posen unterscheiden können. Im Upstream-Reactive-Layer (4.3.4) würden dann dynamische Posen zum Laufen verwendet werden und in einer kritischen Situation kann das System über stabile Posen abgefangen werden. Hierfür ist nur ein Flag in der Posen-Liste der Stabilitätsplanung nötig. Diese Strategie wird nicht im Detail erläutert, um die obige Beschreibung nachvollziehbar zu halten.

[29] Leicht realisierbar mit .net 4.5 und dem parallel.for() Befehl in C# bzw. VC++11.

### 4.3.4 Upstream Reactive Layer

Der hier beschriebene Layer bildet den reaktiven Teil des Systems. Durch die Vorverarbeitung des *Deliberative-Layers* (4.3.3) werden geprüfte Posen aus einer nach „Güte“ sortierten Liste abgerufen und an die Mechanik bzw. die Beinkontroller zur Ausführung übergeben. Weiterhin wird das Laufsystem ständig auf Ausnahmezustände geprüft. Die Kontroller und Module im *Upstream-Reactive-Layer* werden periodisch vom zentralen *Heartbeat* (4.3.5) angestoßen. Die folgende Erläuterung verwendet wieder die Systemübersicht aus Abbildung 4-5 und wird als Ausschnitt in Abbildung 4-17 nochmal dargestellt.

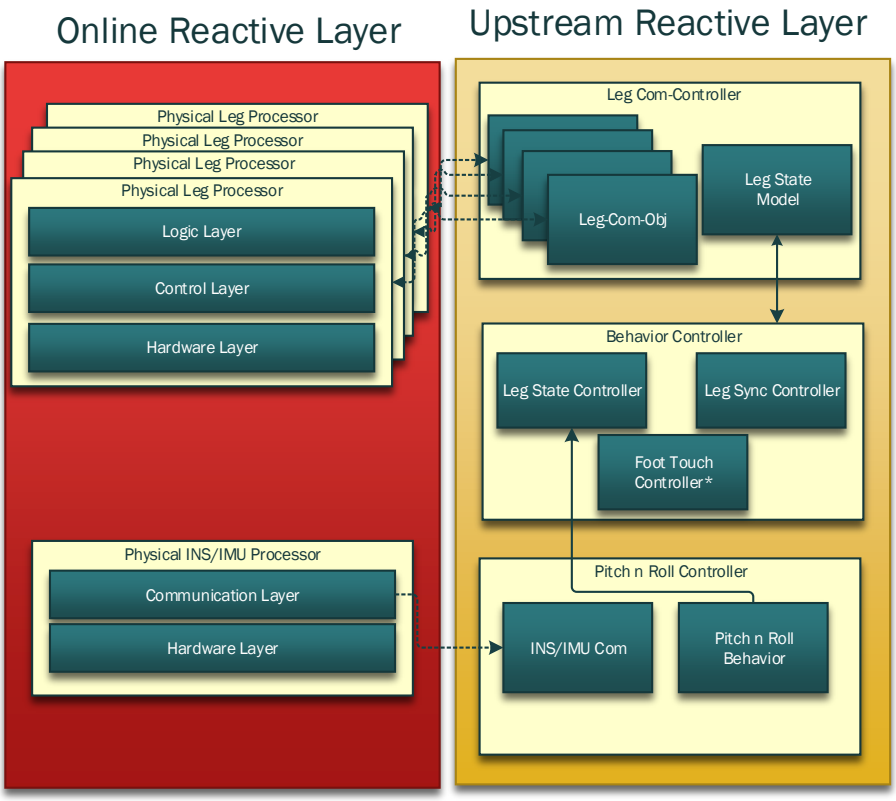


*Abbildung 4-17 Ausschnitt aus der Übersicht Abbildung 4-5*

**Leg-Com-Controller**: Das *Leg-Com-Obj* Modul besteht aus vier unabhängig laufenden Threads und seine Funktion ist **nicht** dem Heartbeat(4.3.5) unterworfen. Diese Module senden die abstrakten Befehle und empfangen die Antworten der Beine. Diese Kommunikation wird für das jeweilige Ziel aufbereitet. In der Initialisierungsphase wird ein permanenter Port auf der IP-Adresse des jeweiligen Beins geöffnet. Danach wird die Selbsttestfunktion der Beine ausgeführt, um sicherzustellen, dass alle Beinsensoren und Aktoren aktiv sind. Weiterhin wird nach jeder Aktion die aktuelle Lage der Glieder abgefragt und in das *Leg-State-Model* übertragen. Verliert ein *Leg-Com* Modul den Port zu seiner festen IP, wird versucht, diesen Port wieder zu öffnen. Bleibt der zweite Versuch erfolglos, wird das

gesamte System angehalten. Dies wird in das *Leg-State-Model* eingetragen und stellt einen kritischen Systemzustand dar. Antwortet eine Bein MCU in einem Zeitfenster von maximal 300ms nicht mehr, kann von einem Defekt ausgegangen werden. Ein weiterer Betrieb des Systems ist damit nicht sicher und das komplette System wird vollständig neugestartet.
Getrieben durch den *Heartbeat,* werden Befehle vom *Leg-Com-Obj* aus dem *Leg-State-Model* abgerufen und an die Beine zur Ausführung übergeben. Erfolgt eine Antwort, wird auch diese dort eingetragen. Bleibt eine Rückmeldung aus (Timeout 5s), wird ein Selbsttest des Beinprozessors erzwungen und im Fehlerfall mit einem Neustart reagiert. Der *Heartbeat* wird hier hauptsächlich genutzt, um inkonsistente Zustände zwischen den Beinen auszuschließen, da jedes Modul über das *Leg-State-Model* und den *Heartbeat* synchronisiert wird.
Im *Leg-State-Model* werden alle Zustände, Informationen und geplante Aktionen gespeichert.
**Behavior-Controller**: Das Modul *Leg-State-Controller* besteht aus mehreren hierarchischen Automaten, die die Abfolge der Beinbewegungen steuern. Bedingt durch die physikalischen Beinkontroller im *Online-Reactive-Layer(4.2.3)* werden hier nur noch rudimentäre Funktionen benötigt. Die Trittpositionen und die Reihenfolge der Schritte werden aus dem *Leg-State-Model* abgerufen, die dort von Deliberative-Layer (4.3.3) hinterlegt wurden. Der *Leg-State-Controller* ruft eine geplante Aktion (Pose) ab und löst diese in Beinanweisungen auf. Dabei koordiniert er auch die Abfolge der Beinanweisungen. Jede Beinanweisung wird wieder in das *Leg-State-Model,* als auszuführende Aktion, zurückgeschrieben. Beim nächsten Heartbeat sendet das betreffende *Leg-Com-Obj* diese Aktion an die Beinkontroller. Danach wartet der *Leg-State-Controller* indirekt auf die Antwort des Beins und leitet die nächste Aktion ein. Das indirekte Warten wird wieder über einen Eintrag im *Leg-State-Model* geregelt. Zudem prüft der *Leg-State-Controller* auch die Lage des Systems im Raum gegen extreme Werte. Die eigentliche Prüfung erfolgt im *Pitch and Roll Controller*. Der *Leg-State-Controller* koordiniert nur die fest modellierten Verhaltensweisen des *Pitch and Roll Behavior* Moduls.
**Leg-Sync-Controller:** Führen die Beine synchrone Aktionen aus, melden die Beinkontroller den Fortschritt der realen mechanischen Bewegung in Prozent zurück [Ruh11]. Durch die Rückgabe in Prozent kann der *Leg-Sync-Controller* einfach und robust gehalten werden, da er nicht die real zurückgelegte Entfernung errechnen[30] muss. Er sammelt die Fortschrittsrückmeldungen und bremst oder beschleunigt die einzelnen Beine, wenn die Rückmeldungen zu stark abweichen. Der Controller greift erst ein, wenn die Nachrichten einen Versatz von über 20% haben[31]. Dazu nutzt er auch das *Leg-State-Model* und kommuniziert darüber indirekt mit den Beinen. Der *Leg-Sync-Controller* überwacht nur den realen Fortschritt der Aktionen und nicht die Positionen oder die Zustände in den Schrittfolgen.

[30] Beim Laufen um Kurven, mit unterschiedlich langen aber synchronen Beinbewegungen, ergibt sich eine relativ komplexe Berechnung.
[31] Am realen Roboter konnte dies kaum beobachtet werden. Bei einer Abweichung scheinen sich die Drehmomente gegenseitig zu hemmen, bis sie wieder synchron arbeiten.

**Foot-Touch-Controller**: Der *Foot-Touch-Controller* behandelt die Rückmeldungen der Drucksensoren in den Füßen des Roboters (3.3.4). Hat der Fuß Bodenkontakt, sendet der Beinkontroller eine Meldung über die Druckverteilung auf die einzelnen Zehen (3.3.4). In Kombination mit den Zuständen des Beins können einige Rückschlüsse daraus gezogen werden. Wird eine Aktion ausgeführt, bei der der Fuß auf den Boden aufsetzen soll, und die Bodenkontaktmeldung geht erheblich früher ein als die Aktions**ende**meldung (Rückmeldung der Beinkontroller), wird eine Fehlerbehandlung eingeleitet. Es kann daraus geschlossen werden, dass die geplante Trittposition in der Realität höher liegt. Würde dann die Aktion stumpf ausgeführt werden, könnte dies den Roboter umwerfen. Der *Foot-Touch-Controller* bricht die laufende Aktion der Beinkontroller ab. Im Detail wird im *Leg-State-Model* die geplante Trittposition mit der aktuellen Fußposition überschreiben. Damit wird die Aktion als erfolgreich beendet und es ist kein aufwendiger Eingriff in den *Leg-State-Controlle*r nötig.
Die gegenteilige Situation tritt ein, wenn die Aktions**ende**meldung ohne Bodenkontaktmeldung eingeht. In der realen Umwelt wäre dann der Boden weiter entfernt als geplant. Dieser Fall hätte ohne den *Foot-Touch-Controller* auch verzögerte Auswirkungen und wird von der Lagekontrolle (*Pitch and Roll Controller*) erst bemerkt, wenn der Schwerpunkt verlagert wird. Wurde festgestellt, dass noch kein Bodenkontakt hergestellt wurde, wird die Trittposition sukzessive nach unten verschoben bis der Bodenkontakt hergestellt wird. Eine Höhenverschiebung ist aber auf maximal 10cm begrenzt. Weicht die Planung derart stark von der realen Umwelt ab, handelt es sich entweder um einen Messfehler (z.B. reflektierte Messpunkte) oder die Umwelt hat sich schlagartig verändert (nachgebender Boden). Das System reagiert mit dem Ausführen einer alternativen Trittposition aus dem *Deliberative-Layer*(4.3.3).
Zudem kann bei einem Body Shift (ziehen des Robotertorsos über die Beine) ermittelt werden, ob der Boden unter dem Fuß nachgibt. In diesem Fall wird das System dazu veranlasst, eine andere Trittposition auszuwählen. Konkret wird im *Leg-State-Model* eine schlechter bewertete Pose ausgewählt, die vom *Deliberative-Layer* hinterlegt wurde. Diese Verhaltensweisen sind ein zusätzliches Sicherheitssystem und beschleunigen die Erkennung eines Fehltrittes. Ein Umkippen des Roboters wird zwar durch den *Pitch and Roll Controller* verhindert, aber erst bei einer starken Abweichung von der geplanten Lage. In einer zukünftigen Entwicklungsstufe soll ein Machine-Learning Verfahren angebunden werden. Dafür könnte die Druckverteilung auf Zehen als Feedback genutzt werden, um die Tragfähigkeit einer Bodenformation zu lernen. Dadurch könnte auch das Stützpolygon (4.3.3.3) dynamisch über die Druckverteilung der Zehen angepasst werden. Mit dieser Erweiterung würde der planende Layer mit feingranulareren Ausgansparametern arbeiten.
**Pitch and Roll Controller:** Der *Pitch and Roll Controller* besteht aus zwei Modulen. Das *INS/IMU-Com* Modul ist ein unabhängiger Thread, der nur den offenen Port der IP-Verbindung zum *Physical-INS/IMU-Processor* überwacht und die Nachrichten von diesem gepuffert weiterleitet. In das System wird über das *Pitch and Roll Behavior* Modul mit einem Regelwerk eingegriffen. Seine Hauptfunktion ist die Überwachung der Lage des Roboters im Raum. Werden experimentell ermittelte

Neigungswerte[32] des Torsos überschritten, wird ein definiertes Verhalten im *Leg-State-Controller* ausgelöst. Das Verhalten ist in zwei Verhaltensweisen unterteilt. Das erste Verhalten versetzt die Höhe der aktuellen Trittpositionen so, dass die Schräglage des Torsos ausgeglichen wird. Dabei werden anteilig die Höhen (Auslenkung der Beine) korrigiert, die in die Richtung des Neigungsvektors zeigen *oder* die dem Neigungsvektor gegenüberliegen. Die Entscheidung, welche Aktion ausgeführt wird, hängt von der noch erreichbaren Höhe des Torsos ab. Priorisiert ist aber eine Korrektur in die Höhe, damit der Roboter genügend Bodenfreiheit behält. Die Berechnung der Kinematik wurde in der vorausgegangen Bachelorarbeit [Ruh11] in Abschnitt 5.3.4.4.9 vorgestellt.
Das zweite Verhalten ist zeitgesteuert. Ist nach einer Sekunde die Schräglage nicht korrigiert, wird eine alternative Pose aus dem *Deliberative-Layer* eingenommen. D.h. die Füße werden an eine andere Trittposition gesetzt. Der Grund für das Auslösen der zeitgesteuerten Verhaltensweisen ist entweder eine Fehlplanung im *Deliberative-Layer* oder ein Rutschen des Fußes bzw. das Nachgeben des Untergrunds. In dieser Situation wird das Laufsystem angehalten und eine komplett neue Planung im *Deliberative-Layer* angestoßen. Damit wird auch die Umwelt neu erfasst und es kommt zu einer kleinen Verzögerung in der Fortbewegung des Roboters.
**Zusammenfassung**: Der reaktive Layer arbeitet mit geplanten Trittpositionen und Körperhaltungsposen aus dem *Deliberative-Layer(4.3.3)*. Aktiv werden nur fehlerhafte Trittpositionen behandelt. Es gibt aber Situationen, die nicht explizit im *Upstream-Reactive-Layer* bearbeitet werden. Die Mehrzahl von unvorhersehbaren Situationen können über den Eingriff des Watchdogs (4.3.5) gelöst werden. Beispiel: Trotz des überdimensionierten Drehmoments der Antriebe haben einige Test mit dem Roboter AMEE gezeigt, dass Situationen eintreten, in denen ein Motor mehr als das maximale Drehmoment aufbringen müsste. Ohne eine zeitliche Überwachung würde der Roboter in dieser Pose „hängen bleiben“, da das System endlos auf eine Rückmeldung warten würde. Bei dem hier vorgestellten System greifen drei Komponenten ein, die alle durch Watchdogs ausgelöst werden. Wird in der obigen Situation vom Beinkontroller (*Online-Reactive-Layer*) kein Fortschritt der Bewegung erkannt, erhöht dieser nach 500ms die Drehzahl des entsprechenden Antriebs [Ruh11], um das sog. Losbrechmoment zu überwinden (Dies wurde in der Bachelorarbeit diskutiert). Zudem sind in den Beinkontrollern experimentell ermittelte maximale Zeiten für Einzelbewegungen hinterlegt. Werden diese überschritten, erfolgt eine Fehlermeldung an den *Upstream-Reaktive-Layer*. Ein weiterer Watchdog liegt im Hauptkontroller und überwacht komplexere Bewegungsabläufe. Wird beispielweise ein Schritt ausgeführt und dafür mehr als 10 Sekunden benötigt, werden neue Trittpositionen bzw. Posen aus dem Deliberative-Layer abgerufen.
Im Worst-Case Fall (in extrem unebenem Gelände mit wenig Halt) kippt der Roboter von einer Pose in die nächste. Das reaktive System ruft dabei in schneller Folge alternative Trittpositionen und Posen aus der nach Güte absteigend sortierten Liste ab. Als Folge würden die Liste vollständig abgearbeitet werden, da der *Deliberative-Layer* eine höhere Laufzeit aufweist. Bedingt durch die hohe Gewichtung der Schrittweite im Bewertungsverfahren (4.3.3.2) des *Deliberative-Layer* werden in dieser Situation

[32] Ca. 30° Pitch und ca. 10° Roll

die Schritte immer kürzer. Schlussendlich bleibt der Roboter in einer geplanten schwerpunktstabilen Pose stehen, bis eine neue Planung vorliegt. Dieses Verhalten wirkt fast natürlich und vorsichtig, hat aber eine Schwachstelle, die auch bei anderen QRTRs beobachtet werden kann. Ist die letzte geplante Pose fehlerhaft, fällt der Roboter um. In dieser Situation – starke Torsoneigung und keine alternativen Trittpositionen mehr - kann nur noch ein Notfallverhalten eingeleitet werden. Der Roboter AMEE zieht die Beine an den Torso und lässt sich fallen. Dieses Verhalten kann auch beim LS3 von Boston Dynamics® beobachtet werden. *Anmerkung: Diese Verhaltensweise wurde NICHT am realen Roboter AMEE getestet, da ein mechanischer Schaden die Projektgelder überschreiten würde. Zudem fehlt auch das Umfeld für solche Tests.*

### 4.3.5 Heartbeat / Watchdog Controller

Der Heartbeat (Timer) und Watchdog Controller (nicht in Abbildung 4-5 dargestellt) stößt fast jeden Controller in diesem System an. Ausgenommen sind die Kommunikationsmodule, da diese jederzeit auf Nachrichten reagieren müssen. Jeder Controller im Hauptkontroller arbeitet wie ein Funktionsaufruf und beendet sich nach seinem sequentiellen Ablauf. Zudem sind im Watchdog-Teil maximale Laufzeiten, der einzelnen Kontroller bzw. des betreffenden mechanischen Ablaufs, hinterlegt. Er kann Controller beenden, Controller gezielt neu starten und das komplette System neu starten.

**Heartbeat**: Die Designentscheidung für einen System-Heartbeat wurde getroffen, da im ganzen System ein (bzw. mehrere) Zeitverhalten modelliert wurden. Für dieses System wurden aber keine Regelalgorithmen verwendet, sondern ein Verhaltensmodell. Das Zeitverhalten wird über den Heartbeat modelliert, um eine Überreaktion und ein Aufschwingen des Systems zu verhindern. Dies ist bedingt durch die Interaktion mit der Mechanik und deren verzögerte Reaktion bzw. Sichtbarkeit in den Sensoren.

Alle Module werden als simple Automaten modelliert, ohne vorerst Rücksicht auf das Zeitverhalten zu nehmen. Jedes Modul führt nur eine Aktion pro Zustandswechsel aus. Der Heartbeat löst die Zustandsübergange aus und bildet damit das Zeitverhalten im System ab.

**Heartbeat in anderen QRTRs:** In einigen Arbeiten werden Zeiten für die Kontroller angegeben. Für den dynamischen Läufer BigDog (3.2.3) wird die Wiederholungsfrequenz der reaktiven Softwaremodule mit 200Hz angegeben. Der Planning-Layer (hier Deliberative-Layer) wird mit 2Hz getaktet. Dies geht aus einem Software- Architekturplan des BigDog (7.4) hervor. Auch die Arbeit von Rebula [Reb08] gibt genaue Zeiten für seine Implementierung und den LittleDog (2.7.1) an. Der Reaktive Layer (in dieser Thesis Upstream-Reactive-Layer) wird mit 50Hz getaktet und der low-level Controller (Reactive-Layer in den Beinkontrollern) mit 500Hz. Natürlich sind diese Zeiten je nach Modellierung und Mechanik unterschiedlich. Interessant ist aber, dass trotz unterschiedlicher Implementierungen die meisten Arbeiten auf ein ähnliches Timingverhältnis kommen. Dies könnte über die Masse des Roboters erklärbar sein, da die Massenträgheit sich direkt auf das Timing auswirkt. Eine spekulative Schlussfolgerung wäre, dass es einen „natürlichen“ Takt für alle Laufsysteme gibt.

Für den Roboter AMEE wird der Reactive-Layer (Beinkontroller 4.2.3) mit 400Hz getaktet. Der Upstream-Reactive-Layer (4.3.4) wird mit 50Hz getaktet. Der Deliberative-Layer (4.3.3) wird mit jeder neuen Schrittfolge angestoßen. Diese Werte sind experimentell ermittelt worden und können noch optimiert werden.

**Watchdog:** Der Heartbeat-Controller wird mit wenig Aufwand um einen Watchdog erweitert. Wird das System von realen Ereignissen getrieben (abgesehen von Timing Events), ist eine Fehlererkennung über das Zeitverhalten schwieriger. Es muss oft das komplette physikalische Verhalten berücksichtigt werden. Beispielsweise müsste eine Beinsteuerung wahrscheinlich auch die Trägheitsmomente berechnen, um zu entscheiden, wann eine Aktion als Fehlerhaft erkannt wird. Es bietet zwar die Vorteile einer allgemeingültigen Lösung, aber der Rechen- und Modellierungsaufwand wäre erheblich höher. Die einfachste Form ist die Observer-Strategie, indem der Watchdog nur die Zustandsübergänge beobachtet und experimentell ermittelte Maximalzeiten vergleicht. Der *Heartbeat-Watchdog-Controller* prüft nach einer bestimmten Anzahl von Heartbeat-Zyklen, ob der Zustandsübergang stattgefunden hat.

**Deadlock-Auflösung:** Da die im Hauptkontroller vorgestellten Verfahren eine noch nicht ausgiebig getestete Eigenentwicklung sind und in ihrer Abfolge ein komplexes Verhalten haben, kann eine Dead- oder Livelockfreiheit nicht garantiert werden. Eine genauere Untersuchung würde den Umfang dieser Thesis in seine Gesamtheit überschreiten. Aber durch die Verwendung des Watchdogs ist auch ein Dead- und Livelockerkennung möglich. Die Kontroller und Module bestehen aus hierarchischen Automaten. Da alle Zustandsübergänge überwacht werden, betrifft dies auch die höhere Ebene des Automaten. Tritt ein Deadlock oder ein Livelock in einem Modul/Automaten auf, findet im übergeordneten Automaten kein Zustandswechsel satt. Dies wird vom Watchdog durch Zeitüberschreitung erkannt und der Automat wird wieder in den Startzustand versetzt. Dadurch, dass alle Informationen und Zustände nicht im Automaten gehalten werden, sondern im Leg-State-Model (4.3) – können die Automaten ihre Zustände wiederherstellen. Da bis zur Erkennung und dem Neustart des Automaten einige Zeit vergeht, haben sich auch die Parameter (Sensorwerte etc.) geändert und die fehlerhafte Zustandskombination wird meist überwunden. Diese Strategie ist zwar *keine* Fehlerauflösung, aber der Roboter bleibt handlungsfähig. Das obige Verfahren konnte auch schon in den Beinkontrollern [Ruh11] erfolgreich gezeigt werden. Problematisch bleibt aber ein Livelock in der obersten Schicht der Hierarchie.

Dieses Prinzip wurde erfolgreich in der vorangegangen Bachelorarbeit [Ruh11] gezeigt. Viele Arbeiten aus dem Learning Locomotion Programms (z.B. Rebula und Team [Reb08]) verwenden dieses Design (ohne Watchdog). Zudem geht aus dem Architekturplan des QRTRs BigDog® (7.4) dieses Design (ohne Watchdog) hervor und hat diese Designentscheidung maßgeblich beeinflusst.

**Weiterentwicklung**: In dieser Entwicklungsstufe ist der *Heartbeat/Watchdog-Controller* noch ein hochpriorisierter Thread des Betriebssystems (Windows 8.1 Embedded®). Aus Sicherheitsgründen sollte dieser Kontroller in eine externe MCU integriert werden. Hier bieten sich MCUs mit erweitertem Temperaturbereich aus dem Fahrzeugbau an. Die Implementierung dieser MCU stößt dann den Takt

der Module extern an und überwacht diese per Ethernet. Der Watchdog würde im extremen Fehlerfall den Hardware-Reset des Hauptrechners (Hauptkontroller) auslösen und so das komplette System neu starten. Zudem bieten die meisten MCUs einen zusätzlichen Hardware-Watchdog an, der wiederum den implementierten Software-Watchdog überwacht.

## 4.4 Zusammenfassung

Das vorgestellte hybride Systemkonzept wurde noch nicht vollständig getestet. Erste Versuche und Simulationen sind aber vielversprechend. Das System abstrahiert die Umwelt von einer Punktwolke zu einer Kachelwelt (4.3.2.2). In der ersten Stufe erfolgt dies noch mit geringer Auflösung und Genauigkeit. Der Roboter „sieht sich um, wohin er gehen kann“. Das System bewertet die Umwelt und sucht eine Laufrichtung (4.3.3.1). Dabei hält es mehrere alternative Richtungen vor. Ist eine optimale Richtung gefunden, wird die Umwelt im Nahbereich noch einmal genauer Ausgewertet (4.3.3.2). Dabei „sieht“ sich das System nur die Trittpositionen an, die es auch mit einem realen Schritt erreichen kann. Auch dieser Nahbereich wird bewertet und sortiert. Danach spielt das System alle möglichen Schrittkombinationen virtuell durch und prüft dabei, ob die virtuell eingenommene Gliederpose schwerpunktstabil sind (4.3.3.3). In stark unebenem Gelände testet das System zusätzlich die Einzelbewegungen, um eine eigene Schrittfolge zu finden, mit der die Erhebung / das Hindernis überwunden werden kann. Dabei entstehen wieder optimale und suboptimale Posen und Schritte, die für die reale Ausführung bereitgehalten werden.
Ist diese Planung abgeschlossen, führt das System die beste Trittposition real aus (4.3.4) - der Roboter macht einen Schritt. Tritt dabei ein Fehler auf – der Roboter kippt trotz Planung – wird eine weniger optimale Trittposition ausgewählt und ausgeführt.
Das vorgestellte System hat noch viele Punkte für Erweiterungen und Verbesserungen, sollte aber eine solide Ausgangsbasis für weitere Experimente mit dem Roboter AMEE-XW2 zur Verfügung stellen. Dabei wird die komplexe Abstimmung der Einzelkomponenten im System kaum dargestellt.

# 5 Implementierung und Ergebnisse

Dieses Kapitel ist in zwei Unterkapitel geteilt. Als erstes werden einige Implementierungsdetails (5.1) vorgestellt. Der zweite Teil (5.2) diskutiert die erreichten Ergebnisse und setzt sich dabei kritisch mit ihnen auseinander.

## 5.1 Implementierung

In diesem Unterkapitel werden einige Implementierungsdetails und gelöste Probleme erläutert. Die Entwicklung ist experimentell getrieben - ähnlich dem Test-Driven-Development (TDD), wobei hier gegen den realen Roboter getestet wird. Das TDD ist zwar zeitintensiv, wird aber durch das hohe Unfallrisiko bevorzugt. Erläutert wird die Implementierung der Beinkontroller (5.1.1), der Lagesensor (5.1.2) und der Hauptkontroller (5.1.3).

### 5.1.1 Beinkontroller

Die Beinkontroller bilden in diesem Konzept den sog. *Online-Reactive-Layer* (4.3.1) und bündeln alle zeitkritischen Aufgaben in einem Laufsystem (4.1.4). Sie stellen die ausführende Schicht im System dar. Aus diesem Grund sind die Beinkontroller mit besonderer Sorgfalt und erhöhtem Testaufwand implementiert. Die Beinkontroller-Software wurde in der vorausgegangen Bachelorarbeit [Ruh11] für einen ATMEL® AT-Mega 128 entwickelt. Die Beinkontroller sind für AMEE-XW2 auf leistungsstärkere MCUs (AT32UC3C0512C (UC3C)) implementiert. Die Gründe für den MCU Wechsel sind in Kapitel 3.4.1 erläutert.

Verwendet wird dafür das Entwicklungskit ATMEL Studio 6.1® mit dem ASF® 3.11.0.792 (Atmel Software Framework) unter Windows® 8 und die Programmiersprache C-11. Das ASF soll den Entwicklungsaufwand durch eine Hardwareabstraktion verringern. Es kapselt nur die direkten Registerzugriffe durch abstrakteren Code. Die Kapselung umfasst aber nicht die spezifische Funktionsweise der Hardware wie andere SDK´s (z.B. Arduino). Für die Implementierung wird kein weiteres Framework verwendet. Das System läuft in einem Superloop und wird durch einen Heartbeat-Hardware-Timer getrieben.

*Anmerkung: Obwohl die vorherige Implementierung die Hardware sauber von der Logik trennt, ist die Migration erheblich aufwendiger als geplant. Bei der leistungsstärkeren MCU handelt es sich um eine völlig andere Prozessorarchitektur. Auch die Anbindung und Funktionsweise der Peripherie unterscheidet sich stark vom AT-Mega 128 und ist vollkommen anderes zu implementieren. Durch diese veränderte Anbindung wird teilweise die Logik verändert. Von den ursprünglichen 2.500 Zeilen C-Code, sind ca. 500 Zeilen angepasst und ca. 1.000 Zeilen neu implementiert.*

**Eventkontroller**: Die alte Implementierung der Beinkontroller wird an kritischen Stellen durch Interrupts getrieben (2011 State-Of-The-Art) und nicht durch den implementierten Heartbeat (ähnlich 4.3.5). Der Prozessorhersteller empfiehlt (2014) für die UC3C MCU die Verwendung von Events, die durch das ASF angeboten werden. Für diese Events hat die MCU eine nebenläufige Einheit

(Eventkontroller), die die Register der I/Os pollt[33]. In der angepassten Implementierung der Beinkontroller sind alle Interrupts durch Events ersetzt, was die Implementierung stark verändert. Die Beinkontroller sind nun vollständig durch den Heartbeat getrieben und der Eventkontroller wird nur zu seinem Heartbeat befragt.

**DMA**: Die neue MCU verwendet einen Hardware-DMA-Kontroller, der direkt in den verwendeten Adressbereich der MCU schreibt. Viele Schnittstellen sind vollständig in Hardware abgebildet und arbeiten unabhängig vom Kern der MCU. In den Beinkontrollern wird dies beispielweise für die Anbindung der seriellen Schnittstelle genutzt, was vorher durch Interrupts gelöst wurde.

**32-Bit**: Die alte MCU arbeitet mit einem 8-Bit Kern und High/Low Adressen, um 16Bit Adressen bzw. Register anzusprechen. Die neue MCU hat einen 32-Bit Kern und Bussystem. Die Logik der alten Implementierung ist teilweise auf eine 8-Bit Logik programmiert (z.B. Bit-Shift, eigene Schattenregister für die Automaten). Diese Logik ist zwar auf einer 32-Bit Prozessorarchitektur uneingeschränkt lauffähig, aber es sollen die Vorteile der neuen MCU voll ausgenutzt werden. Zudem ist auf der alten MCU eine Abfragen von 8-Bit-Werten über die Adresse (16-Bit Pointer = min. zwei Takte) langsamer als ein call-by-value. Diese Randbedingungen sind auf der neuen MCU nicht nötig und sind beseitigt.

Durch diese Änderungen ist die Implementierung in den neuen Beinkontrollern vollständig von einem eigenen Heartbeat-Kontroller getrieben und realisiert die Vision aus der vorausgegangen Bachelorarbeit [Ruh11].

### 5.1.2 Lagesensor

Um den Lagesensor-Chip (3.4.3.2) an das Ethernet im Roboter anzubinden und das interne Netz gegen eine Datenflut zu schützen, ist eine MCU (ATMEL® ATMega 328) integriert. Die Implementierung fragt den Lagesensor alle 10ms ab und vergleicht die Messwerte mit dem vorherigen Werten. Weichen die neuen Neigungswerte um mehr als 1° Winkelgrad von den alten Wert ab, erfolgt eine Meldung an den Hauptkontroller per Ethernet. Der Kompass (bzw. drei Kompasse) wird gesondert behandelt, da er trotz Sensorfusion (DMP 3.4.3.2) stark durch die Antrieb beeinflusst wird. Die Kompassdaten sind nur bedingt brauchbar und es erfolgt keine Meldung, wenn der Roboter um die senkrechte Achse gedreht wird. Die Daten des Kompasses werden gemittelt und nur mit einer Neigungsänderung mitgesendet. Erfolgt eine ständige Änderung der Lagedaten, wird ein Timer gestartet, der das Senden der Lagedaten auf einmal pro 200ms begrenzt.

Für die Implementierung wird die sog. MPU9150Lib von Pansenti[34] verwendet. Diese abstrahiert die Initialisierung und Kommunikation mit dem Lagesensor-Chip.

[33] Laut Hersteller ist es schneller, den Event-Kontroller zu befragen, als bei jedem Interrupt alle Register zu sichern und wiederherzustellen. Dies wurde nicht überprüft.

[34] www.pansenti.com, info@pansenti.com. Der Code ist dort seit Ende des Jahres 2013 nicht mehr abrufbar. Er kann aber auf wechselnden Internetplattformen (unter dem Suchwort „Atmel MPU9150“) gefunden werden.

### 5.1.3 Hauptkontroller

Der Hauptkontroller umfasst den Upstream-Reactive-Layer(4.3.4), Deliberative-Layer(4.3.3) und die Umwelterfassung im Higher-Logic-Layer (4.3.2.2). Durch den Umfang der Implementierung ist der Upstream-Reactive-Layer nur teilweise in das Robotersystem implementiert. Der Deliberative-Layer liegt nur in einer nicht optimierten und simulierten Form vor. Für eine vollständige Integration in das Robotersystem mit Abstimmung der Parameter werden noch weitere 4 Monate benötigt. Aus diesem Grund sind hier nur einige besondere Details aufgeführt.

Für den Hauptkontroller wird *.net 4.5.1* von Microsoft® unter *Windows 8.1 Embedded Industrial Pro®* verwendet. Für erste Tests und Simulationen wird das AForge.NET[35] und Accord.NET Framework stark modifiziert genutzt. Anstatt VC++11 zu verwenden, wird als Programmiersprache C# verwendet. Diese Designentscheidung soll den Aspekt der weichen Zeitanforderungen (4.1.4) im Hauptkontroller unterstreichen.

**Grundsystem**: Alle Layer bzw. fast alle Module werden periodisch per Heartbeat(4.3.5) angestoßen. Der Heartbeat ist ein gesonderter System-Thread und löst alle 100ms einen Takt aus, der wiederum den eigentlichen Heartbeat-Kontroller (Automat) periodisch anspricht. Die Initialisierung des Systems erzeugt persistente Objekte (in eigenen Threads) und persistente Datenobjekte, die die meiste Zeit untätig im Hauptspeicher auf einen Aufruf durch den Heartbeat warten. Zudem sind sie so lose gekoppelt, das sie keine Referenzen aufeinander halten. Um die Garbage Collection nicht abschalten zu müssen und nicht alle Methoden static zu implementieren, wird ein Referenzobjekt (Leg-State-Modell 4.3.4) erzeugt. In diesem Objekt werden die Referenzen von Objekten und Datenobjekten gehalten, die periodisch verwendet werden. Dieses Referenzobjekt wird vom Heartbeat periodisch angesprochen und führt eine Dummymethode aus, um ein Löschen durch die Garbage Collection zu verhindern (Absatz Probleme). Auch die Referenz der erfasste Punktwolke wird vom Referenzobjekt gehalten, da sie zeitlich versetzt von zwei unabhängigen Modulen (4.3.3.1.1, 4.3.3.2) genutzt wird und sonst in der Zwischenzeit von keinem anderen Thread oder Objekt gehalten wird.

Die Objekte kommunizieren nicht über Events (wie bei C# sonst üblich), sondern es erfolgt immer ein Methodenaufruf, der wiederum eher wie ein Funktionsaufruf gehandhabt wird.

Mit diesen Maßnahmen wird ein softes Echtzeitverhalten erreicht und das erfolgreiche Konzept in den Beinkontroller wird auch im Hauptkontroller vorgesetzt.

**Parallelität**: Ein Vorteil der Umweltabstraktion zur Kachelwelt (4.1.3) ist die einfach zu implementierende Parallelisierung der Verfahren. Dies wird an einigen Beispielen erläutert. Eine erfasste Punktwolke wird in Subpunktwolken (4.3.2.2) segmentiert. Jeder Punkt aus der Punktwolke wird iterativ einer Subpunktwolke zugeordnet. Das Verfahren ist ohne Rücksicht auf die Parallelität implementiert (wie ein sequenzieller Ablauf). Die Iteration ist ein Parallel.For-Loop. Das .net Framework regelt automatisch die optimale Lastverteilung auf die noch freien Prozessoren, mit einem auf das System angepassten Thread-Pool. Auch die Synchronisation der parallelen Bearbeitung wird automatisch vom System geregelt.

[35] http://www.aforgenet.com

Für die Bewertung der Kacheln (4.3.3.1, 4.3.3.2), die Bewertung von Pfaden (4.3.3.1.1) bis hin zur Stabilitätsberechnung (4.3.3.3) wird dieses Vorgehen verwendet, da die Kacheln, Pfade und Gliederposen unabhängig voneinander sind. Durch die Verwendung von Threadsafe sortierten Listen wird während der laufenden Berechnung auf die Liste zugegriffen. Dadurch kann beispielweise der Upstream-Reactive-Layer schon eine nicht optimale Pose abrufen, obwohl die Stabilitätsplanung noch nicht vollständig abgeschlossen ist.

***Probleme**: Nach der Initialisierung des Hauptsystems bleibt die Main-Methode in einer leeren Schleife, da der Hauptkontroller sich nicht selbst beenden können soll. Dies erweist sich als problematisch. Systeme mit Windows 8.1 oder Windows Server 2012R2 erkennen einen leeren Loop und beenden diesen spontan (ca. 65 Minuten). Es konnte noch nicht geklärt werden, wodurch dieses Verhalten ausgelöst wird. Erst wenn eine erfüllbare Abbruchbedingung für das komplette Programm implementiert wird, läuft der Timer wie erwartet endlos. Die Abbruchbedingung darf auch nicht in einem toten Zweig enden. Das System scheint zu erkennen, dass eine statische (nicht erfüllbare) Definition der Abbruchbedingung vorliegt.*

*Aus diesem Grund besteht auch eine leichte Verunsicherung bei statischen Objekten und die oben erwähnte Lösung (Referenzobjekt) wurde implementiert.*

## 5.2 Ergebnisse

Bedingt durch die hohe Abhängigkeit zwischen der Mechanik und der Implementierung werden in diesem Kapitel die Einzelergebnisse in drei Punkten zusammengefasst und kritisch untersucht. Dieses Unterkapitel ist in drei Schwerpunkte unterteilt: Mechanik (5.2.1), Beinkontroller (5.2.2) und Hauptkontroller (5.2.3).

### 5.2.1 Mechanik

Es wurden erste Laufversuche und Belastungstest mit dem Roboter durchgeführt. Der Roboter erweist sich als agil und kräftig. Die Realisierung der Mechanik ist erfolgreich und bedarf nur noch kleiner Korrekturen (5.2.1.1).

#### 5.2.1.1 Kritik

Das Laufverhalten konnte nur rudimentär getestet werden. Dies ist bedingt durch zwei Faktoren. Erstens: Die Möglichkeiten im Testumfeld sind nicht ideal. Das Stützgestell (3.3.6) hat die maximale Größe, um es im Testumfeld zu handhaben. Es ist aber zu klein für den Roboter AMEE-XW2. Wird der Roboter im Stützgestell (3.3.6) beweglich aufgehängt, fängt er beim Laufen leicht an zu schwingen und tritt auf die seitlichen Kanten des Gestells. Dadurch ist es relativ schwierig, mehrere Schrittfolgen zu testen. Aus diesem Grund wird auch das Kurvenlaufen vernachlässigt. Für weitere Tests müsste der Roboter im Gelände oder an großvolumigen Geländeformationen getestet werden. Diese Möglichkeiten sind im Testumfeld schwer zu realisieren. Zweitens: Der Zeitfaktor für die Realisierung wurde unterschätzt. Bei der Produktion kam es mehrfach zu Verzögerungen durch

Lieferschwierigkeiten bei einigen Kleinteilen. Dies wurde nicht in der Zeitplanung berücksichtigt.
Durch ein fehlendes Werkzeug konnte die sog. Nut in den Stahlachsen (3.3.3) nicht richtig gefertigt werden. Als Lösung wurde die Nut zu klein gefertigt und in die Antriebe eine kleinere Passfeder eingesetzt. Dadurch haben die Gelenke in Drehrichtung ein Spiel von einem Millimeter, was sich über alle Achsen auf fast 10mm Spiel aussummiert. Erfolgt bei einem Schritt ein Lastwechsel durch die Verlagerung des Schwerpunkts, fällt der Roboter schlagartig um 10mm in die Bewegungsrichtung. Dadurch wirkt das Laufverhalten zu „energisch“ und nicht vorsichtig, wie geplant. Zudem ist ein leichter Zero-Spin (7.5) zu erkennen, der sich durch ein abruptes Abbremsen des Roboters beim Auftreten äußert. Diese Probleme wurden unterschätzt.
Diese Probleme sind relativ leicht zu beheben. Der Roboter AMEE bietet viel Potential für Weiterentwicklungen, da modular konstruiert wurde.

### 5.2.2 Beinkontroller

Für diesen Roboter werden die Beinkontroller auf einer leistungsfähigeren Plattform betrieben. Die dabei entstandene Beinkontrollersoftware (5.1.1) nutzt die technischen Möglichkeiten der neuen Hardware aus. Im Vergleich zur ursprünglichen Implementierung in der vorausgegangen Bachelorarbeit ist die Reaktionszeit der Beinkontroller geringer, was ein Ziel des Plattformwechsels war. Die Beinkontroller sind erfolgreich realisiert. Der Zeitaufwand wurde aber auch hier unterschätzt, da die neue MCU-Architektur komplexer ist, als geplant war.
**Reaktionszeiten**: Die neuen Beinkontroller haben eine bessere Reaktionszeit gegenüber der alten Implementierung und Plattform. Die ermittelten Reaktionszeiten beziehen sich auf die Beinkontroller in der vorausgegangen Bachelorarbeit [Ruh11] und auf die neuen Beinkontroller aus dieser Thesis.
Für die Messung wird das JTAG-Interface der MCU (JTAGICEmkII®) genutzt und ein Breakpoint an den Eintritt des Super-Loops gelegt. Im Debug-Modus des ATMEL Studios wird damit die reale Laufzeit der MCU ermittelt. Die Betrachtung der Reaktionszeit findet unter zwei Bedingungen statt. Erstens die Reaktionszeit, wenn das System nur die Antriebe regelt. Zweitens die Reaktionszeit, wenn zu den Steuer-/Regelzeiten die Rechenzeit der inversen Kinematik hinzukommt. Die gemessenen Zeiten geben eine Auskunft darüber, wie schnell das System beispielsweise auf einen Stop-Befehl reagiert. Ermöglicht wird diese Betrachtung durch ein Implementierungsdetail, das zwei unterschiedliche Bewegungsarten ermöglicht.
Die eine Bewegungsart wird als *Move-2-Point* [Ruh11] bezeichnet. Dabei werden nur einmal die Zielwinkel von der inversen Kinematik berechnet und die Zielwinkel werden direkt angefahren. Die Messung enthält nur die Zeit, die pro Super-Loop für die Regelung benötigt wird. In der alten Implementierung der Beinkontroller werden 633,98µs für eine durchlauf der *Move-2-Point* Anweisung benötigt. Die neue Implementierung mit der neuen MCU Plattform benötigt ca. 320µs für den *Move-2-Point* pro durchlauf.
Die zweite Bewegungsart wird als *Linemover* bezeichnet und das System berechnet bei jedem Super-Loop Durchlauf neue Winkel für die Glieder. Hier werden die Regelzeiten plus die Rechenzeit der inversen Kinematik gemessen. Der alte Beinkontroller benötigte 1332,8µs für jede Berechnung

inklusive Regelaufgaben. Der neue Beinkontroller benötigt noch ca. 530µs.
Die Leistungssteigerung wird durch die real vorhandene FPU[36] und die 32-Bit Busbreite erreicht. Die meiste Rechenzeit wird durch die inverse Kinematik (bzw. den Winkelfunktionen) verursacht. Die Leistungssteigerung steht aber nicht im Verhältnis zur Taktratensteigerung. Der alte ATMega 128 wurde mit 18 MHz getaktet. Die neue UC3C MCU wird mit 66MHz getaktet (Taktsteigerungsfaktor 3,66 / Leistungssteigerungsfaktor 2,51). Dies könnte durch den Architekturwechsel von RISC® auf ARM® bedingt sein.
Zum Abschluss der Geschwindigkeitstests sollten auch die Zeiten der Interruptroutine mit den Bearbeitungszeiten für eine Anfrage beim Eventkontroller verglichen werden. Die Zeit für eine Abfrage des Eventkontrollers konnte trotz der Verwendung eines JTAG-Interfaces nicht ermittelt werden. Der Eventkontroller ist direkt mit dem sog. High-Speed-Bus (66MHz Bus-Takt) verbunden und benötigt *vermutlich* einen Takt pro Abfrage.

#### 5.2.2.1 Kritik

Wie auch in der vorausgegangen Bachelorarbeit [Ruh12] kommt es bei langsamen Bewegungen zu einem kurzen Vollausschlag in den AD-Wandlern, die mit den Winkelsensoren verbunden sind. Dieser Umstand konnte noch nicht genau geklärt werden. Eine Vermutung ist, dass die Steuerimpulse (PWM) der Treiberelektronik in die Messleitungen induziert wird. Dies kann auch nicht mit abgeschirmten Messleitungen verhindert werden. Dieses Problem wirkt sich nicht so stark aus wie in der Vorarbeit. Das Problem wird zurzeit mit einem gleitenden Mittelwert der Messwerte abgeschwächt. Weiterhin wird der Sensorwert von der Beinkontrollerlogik beim Anlaufen der Antriebe kurzzeitig ignoriert. Bedingt durch diesen Workaround muss dies bei einer Positionsanfrage an die Beinkontroller vom Hauptkontroller berücksichtigt werden. Eine nicht realisierte Lösung wäre es, die AD-Wandler von kleinen MCUs (z.B. ATMEL® AT-Tiny 12[37]) direkt an jeden Sensor anzuschließen. Die kleineren „Mess-MCUs“ würden dann ihre Messwerte per 1-Wire an den Beinkontroller übermitteln. Die vorhandenen Messleitungen fungieren dann als Bus- und Stromversorgungsleitungen zwischen Beinkontroller und Mess-MCUs.
Die Drucksensoren sind nur provisorisch angeschlossen, da sie Lötkontakte haben, die direkt auf den leitenden Kunststoff übergehen. Der Kunststoff ist nur bis ca. 80°C stabil und es konnte keine befriedigende Klemmlösung gefunden werden.

### 5.2.3 Hauptkontroller

Der Hauptkontroller ist ein hybrides System mit einem reaktiven (Reactive-Layer 4.3.4) und einem planenden Teil (Deliberative-Layer 4.3.3). In dieser Thesis ist der Reactive-Layer teilweise in das Robotersystem implementiert. Der Deliberative-Layer befindet sich noch in der Testphase und ist in rudimentären Simulationsmodulen implementiert. Aus diesem Grund wird das ganze System auch als

[36] Die meisten MCUs haben zusätzliche Pipelinestufen und Operatoren, aber meist keine eigenständige FPU
[37] Diese MCUs sind ohne externe Beschaltung lauffähig. Sie könnten direkt in den Kabelbaum integriert werden.

Konzept bezeichnet.
Der Deliberative-Layer bewertet die Umwelt durch den Umweltsensor (4.3.3.1). Dafür wird die Welt zu Trittpositionen abstrahiert und besteht für den Roboter nur noch aus Kacheln (4.3.2.2). Die Umwelt wird grob untersucht, um eine optimale Laufrichtung zu finden (4.3.3.1.1). Durch eine Bewertung nach erreichbaren Höhenunterschieden der Kacheln wird eine Kostenkarte der Umwelt erstellt. Danach werden geradlinige Pfade auf Begehbarkeit geprüft. Diese Prüfung unterscheidet zwischen übersteigbaren und unüberwindbaren Hindernissen. Um das Verfahren einfach zu halten, wird der Sichtschatten von großen Hindernissen aktiv als Erkennungsmerkmal genutzt. Eine weitere Prüfung untersucht die Kopffreiheit (4.3.3.1.2) und unterscheidet dabei kleine verdrängbare Hindernisse von massiven Hindernissen. Sind diese Prüfungen abgeschlossen, werden die Ergebnisse verrechnet und eine bewertete Liste mit Laufrichtungen an die nächste Stufe übergeben.
Die nächste Stufe (4.3.3.2) untersucht nur noch die Umwelt, die von den Beinen direkt erreicht werden kann, in einer höheren Kachelauflösung. Dafür wird wieder eine Kostenkarte nach Erreichbarkeitskriterien erstellt. In dieser Stufe stellen die Kacheln direkt die Trittpositionen dar und der nächsten Stufe wird die Kostenkarte als sortierte Liste übergeben. Die dritte und letzte Planungsstufe (4.3.3.3) im Deliberative-Layer untersucht die von der Vorstufe ermittelten Kacheln auf Schwerpunktstabilität des Roboters. Dafür werden die Füße virtuell auf die vorher ermittelten Kacheln bzw. Trittpositionen gesetzt. Die daraus entstandene Gliederpose des Roboters wird gegen sein resultierendes Stabilitätspolygon getestet. Dabei werden die zwei Schrittmuster unterschieden. Findet das System keine schwerpunktstabile Pose mit nur zwei beteiligten Füßen (Fast-Walk), wechselt das System das Schrittmuster und verwendet drei Auflagepunkte (Save-Walk). Um eine Pose mit drei Auflagepunkten / Füße zu testen wird auch die Reihenfolge der Einzelbewegungen getestet. Diese Stufe füllt eine Liste mit Trittpositionen, Gliederposen und Bewegungsabläufen. Aufgrund des Bewertungsverfahrens ist diese nach bestimmten Kriterien sortiert und enthält auch alternative Posen. Diese sortierte Liste wird für den Reactive-Layer bereitgehalten.
Die Bewertungsverfahren in den einzelnen Stufen sind über Matrizen modelliert (4.3.3.1). Jeder wünschenswerte Aspekt für ein Laufsystem wird durch eine Matrix abgebildet, wie beispielweise die gewünschte Laufrichtung und die Reichweite. Das System kann mit einem Anfügen einer weiteren Matrix einfach erweitert werden. Um das Verhalten des Laufsystems zu beeinflussen, können Skalare an den Matrizen verändert werden. Damit kann die Gewichtung eines einzelnen Aspekts beeinflusst werden (4.3.3.1.4).
Der Upstream-Reactive-Layer (4.3.4) überwacht die Torsoneigung und die Beinkontroller. Dieser Layer synchronisiert komplexe Bewegungen der Beine und erteilt ihnen Befehle. Seine Hauptaufgabe ist es die Planung aus dem Upstream-Deliberative-Layer auszuführen und Fehltritte zu erkennen. Der Upstream-Reactive-Layer greift auf die vom Upstream-Deliberative-Layer erstellte sortierte Liste zu und führt die als optimale bewertete Aktion aus. Dabei löst der Upstream-Reactive-Layer die Anweisungen aus dieser Liste in Trittpositionen auf und übermittelt dies an die entsprechenden Beinkontroller (Online-Reactive-Layer). Die Beinkontroller (4.2.3) lösen die Trittpositionen in Einzelbewegungen auf.

Tritt dabei ein Fehler durch eine Fehltritt auf, was durch die Torsoneigung (3.4.3.2, 5.1.2) und die Drucksensoren (3.4.3.4) in den Füßen erkannt wird, ruft der Upstream-Reactive-Layer eine alternative Pose aus der sortierten Liste des Deliberative-Layers ab. Der Vorteil ist dabei, dass das System im Fall eines Fehltrittes ohne Zeitverzögerung reagieren kann. Der Upstream-Reactive-Layer kann sofort alternative Trittpositionen an die Beinkontroller senden.
Eine noch nicht ausgiebig getestete Erweiterung dieses Systems ist die Anwendung von dynamischen Schritten (4.3.3.3), wobei der Roboter in Laufrichtung gekippt wird und das statische Verhalten verlässt. Der Vorteil wäre ein schnelleres Laufen im Fast-Walk Modus des Laufsystems.
Das Gesamtkonzept muss noch unter realen Bedingungen getestet werden, was zurzeit nicht ohne hohen Zeitaufwand möglich ist (5.2.1.1). Das simulierte Verhalten des Gesamtsystems ist aber vielversprechend und die vorgestellten Verfahren bieten noch viele Möglichkeiten für Erweiterungen. Die Verfahren sind auch noch nicht optimiert und können weiter in der Performance gesteigert werden.
**Benchmarks**: Interessant sind die benötigten Zeiten für die Verfahren im Deliberative-Layer. Auch wenn diese noch nicht auf Geschwindigkeit optimiert sind und zurzeit eher ein proof-of-concept darstellen, wurden einige Zeiten als Anhaltspunkte ermittelt.
Die Segmentierung einer Punktwolke mit 210.000 Punkten in 1.000 Subpunktwolken benötigt ca. 46 ms. Die Ausgangspunktwolke hat eine Größe von 4m³ und wird in 0,4m³ große Subpunktwolken aufgeteilt. Dieser Benchmark bezieht sich auf die Umweltabstraktion in der Laufrichtungsplanung (4.3.3.1)
Die Flächenerkennung über 1.000 Subpunktwolken mit maximal 13.000 Punkten pro Subpunktwolke und 300 Iterationen des RANSAC Verfahrens benötigte 11**s** (andere Implementierungen benötigen ca. 500-900ms für ähnliche Datenmengen 5.2.3.1). Für diese Verfahren wird der RANSAC-Plane Algorithmus des AForge.net Framework optimiert und um einen Zugriff auf die Speicheradressen der Punktwolke erweitert. Dieser Benchmark bezieht sich auch auf die Laufrichtungsplanung.
Der Brute Force Ansatz in der Stabilitätsplanung[38] (4.3.3.3) benötigt für 331.776 Trittpositionsmöglichkeiten (4 x 6 Kacheln hoch 4 Beine) 9 ms. Dies ist der Worse-Case Fall, wenn alle Kacheln sich als Trittposition eignen. Zudem sind nach 0,03 ms die ersten drei Posen getestet und das Verfahren könnte abgebrochen werden.
Diese Tests sind noch synthetisch[39], da sie mit simulierten Daten durchgeführt wurden. Als Testplattform kam ein Intel Core 2 Quad® mit 3,2GHz zum Einsatz. Der im Roboter verwendete Hauptrechner (3.4.2) (Intel i7 4770T®) ist im Durchschnitt 25% bis 45% schneller als die verwendete Testplattform. Alle angegeben Zeit sind der Mittelwert von 10 Testläufen.

[38] Optimierter Code inkl. optimierter inversen Kinematik [Ruh11]
[39] Die Ergebnisse und das Verhalten (Trittpositionen, Pfade) wurden visualisiert und visuell überprüft.

#### 5.2.3.1 Kritik

Eine der größten Schwachstellen dieses Konzepts ist der ausgebliebene Beweis seiner Funktionsfähigkeit im Feldversuch. Es wurden zwar wissenschaftliche Arbeiten gefunden, die ein ähnliches Verfahren nutzen (2.6.7), aber auch in einigen Punkten abweichen. Deshalb bleibt es in dieser Thesis bei einem Konzept.

Ein weiterer Punkt ist die hohe Laufzeit des RANSAC-Plane Algorithmus aus dem Accord.net Framework (4.1.3). Dieses Verfahren muss für den Roboter AMEE-XW2 optimiert werden. Andere Implementierungen wie beispielweise die Pointcloud Library haben eine um den Faktor 10 kleinere Laufzeit. Dies liegt aber nicht an der verwendeten Programmiersprache C#[40]. Ein zweiter Ansatz wäre es, den RANSAC-Plane Algorithmus in die GPU zu verlagern.

In Kapitel 4.3.3.1.4 wird das Problem einer engen Sackgasse aufgezeigt. Das Problem verstärkt sich durch die geringe Reichweite (4,5m) des verwendeten Sensors. Um dieses Problem abzuschwächen, sollte der Roboter mit einer Stereokamera erweitert werden. In der Arbeit von Fathi und Brilakis [Fat11] wird ein einfaches Verfahren zur Generierung von Punktwolken mit einfachen Videokameras vorgestellt. Die Punktwolke ist zwar ungenau, aber eignet sich für die Laufrichtungsplanung (4.3.3.1) in dieser Thesis. Damit würde der Roboter AMEE-XW2 eine ungenaue Punktwolke (bis ca. 10m) für die Richtungsentscheidung nutzen und für die Trittpositionsplanung die genaue Punktewolke des Kinect V2 Sensors.

Die vorgestellten Verfahren im Deliberative-Layer müssen intensiver auf Schwachstellen getestet werden. In den Simulationen konnte keine Situation gefunden werden, die das System nicht bewältigen konnte (abgesehen von Sackgassen). Es ist aber wahrscheinlich, dass es diese Situationen gibt. Eine theoretische Untersuchung aller möglichen Zustände ist noch nicht erfolgt.

[40] Beispielweise http://www.codeproject.com/Articles/212856/Head-to-head-benchmark-Csharp-vs-NET

# 6 Schluss

In diesem Kapitel wird das vorgestellte Gesamtkonzept eines QRTRs zusammengefasst. Im Kapitel 6.2 wird eine Vision und einige Punkte für die Weiterentwicklung, anhand von Arbeitspaketen, vorgestellt.

## 6.1 Zusammenfassung

Diese Thesis behandelt das Gesamtkonzept eines Quadruped Rough Terrain Robots (QRTR) mit dem Schwerpunkt des autonomen Laufens auf unebenem Untergrund. Dazu wurden die physikalischen Hintergründe und die grundlegenden Strategien untersucht. Um ein eigenes Gesamtkonzept aufzustellen wurden die Arbeiten aus dem *Learning Locomotion Program* der DARPA genauer evaluiert und daraus wurden einige Vorarbeiten erstellt.
Durch die Erkenntnisse aus der Grundlagenforschung zu vierbeinigen Robotern konnte ein eigener pseudodynamischer Läufer konstruiert und realisiert werden und dient als Versuchsaufbau. Basierend auf den mechanischen Möglichkeiten dieses Roboters wurden eine Kontrollerarchitekur und die dafür benötigten Verfahren entwickelt. Das Kontrollerkonzept wurde als hybrides System ausgelegt und um eine Schicht erweitert. Diese Erweiterung spaltet den reaktiven Teil nach harten und weichen Zeitanforderungen. Der reaktive Teil mit harten Zeitanforderungen wurde in eine angepasste externe Hardware gekapselt.
Für die planende Schicht wurde die Umwelt zu einer „Kachelwelt“ abstrahiert. Dafür wurde ein Bewertungsverfahren entwickelt, das alle Trittpositionen nach unterschiedlichen Aspekten bewertet. Diese Aspekte umfassen auch eine Schwerpunktstabilitätsplanung. Zudem kann über das Bewertungsverfahren das Laufverhalten variabel eingestellt werden. Für die reaktiven Verfahren wurden Erkennungsmerkmale und Verhaltensweise zur Behandlung von Fehltritte des Roboters entwickelt. Zudem synchronisiert der reaktive Teil des Kontrollerkonzepts die Beine und sorgt für eine schwerpunktstabile Lage des Roboters. Das hybride System wurde so ausgelegt, dass es bei einer Fehlplanung sofort auf andere vorausberechnete Trittpositionen zugreifen kann.
Das Kontrollerkonzept wurde teilweise Implementiert und getestet. Für eine vollständige Beurteilung des Gesamtsystems stehen aber noch reale Feldversuche aus. Der vorgestellte Roboter und das Kontrollerkonzept können durch ihren modularen Aufbau leicht weiterentwickelt werden.

## 6.2 Vision / Weiterentwicklung

Die Forschung an laufenden Robotern ist bei weitem noch nicht abgeschlossen. Jedes gelöste Problem wirft neue Fragestellungen auf, wie wahrscheinlich in allen Naturwissenschaften. Das vorgestellte Gesamtkonzept kann sich nicht mit anderen Entwicklungen aus diesem Bereich messen. Die Erfolge der letzten Jahre vermitteln aber einen Eindruck darüber, was technisch machbar ist und sein wird. Es zeigt sich aber immer mehr, dass selbst für das Laufen eine gewisse Intelligenz nötig ist. In der Zukunft soll sich die Arbeit des Autors in diesen Bereich verlagern.

Die kurzfristige Vision ist die Realisierung aller Projektziele im Projekt AMEE (2.4). Der Roboter AMEE-XW2 soll autonom in unebenem Gelände und abseits von Wegen laufen können. Um dieses Ziel zu erreichen fehlen noch einige Komponenten im Gesamtsystem AMEE-XW2. Für die Weiterentwicklung des Systems werden hier einige Arbeitspakete vorgestellt. Diese Arbeitspakete sind direkt auf den Roboter AMEE-XW2 zugeschnitten.

### 6.2.1 Odometrie

Eine genaue Odometrie (2.2.5) ist für einen laufenden Roboter besonders wichtig. Rutscht ein Fuß, kann dies der Roboter nicht detektieren. Das Verfahren der Trittpositionsplanung (4.3.3.2) ist in diesem Konzept besonders davon abhängig, da vorher erfasste Kacheln unter dem Roboter verschoben werden. Wieweit und wohin diese Kacheln verschoben werden, hängt momentan von der aktiven Bewegungsrichtung ab.
Hier bietet sich ein Verfahren an, indem die Verschiebung der Punktwolke des Kinect V2 Sensors beobachtet wird. Aufgrund der Genauigkeit der Punktwolken müsste diese Odometrie genauer sein als andere optische Verfahren. Für den Kinect V1 Sensor war diese Möglichkeit durch das Kinect Fusion® Paket schon gegeben.

### 6.2.2 Pfadplanung

Die Arbeit „Path Planning for Robotic Demining: Robust Sensor-based Coverage of Unstructured Environments and Probabilistic Methods" [Aca03] stellt ein Verfahren zur Pfadplanung für Mienensuchroboter vor. Dieses Verfahren könnte sich auch für allgemeine Suchaufgaben eignen. Der Roboter AMEE-XW2 könnte damit ähnliche Aufträge wie ein Suchhund übernehmen.

### 6.2.3 Drehen auf der Stelle

Das Wenden des Roboter AMEE-XW2 auf der Stelle ist komplex. Ein Thema der Arbeit "Omnidirectional Static Walking of a Quadruped Robot on a Slope" [Zha051] beschäftigt sich damit. Vielleicht ist eine Integration in das vorhandene System möglich.

### 6.2.4 Adaptive Gewichtung

Ein großer Entwicklungsschwerpunkt könnte es sein, eine KI an die Skalare der Bewertungsmatrizen in der Laufrichtung- (4.3.3.1) und Trittpositionsplanung (4.3.3.2) anzubinden. Zudem könnten auch noch zusätzliche Matrizen an die Bewertung angehängt werden, die von einem Learning-Verfahren gefüllt werden. Hierfür eignet sich auch das Verfahren aus der parallel erstellten Master-Thesis aus dem Projekt AMEE von Bettzüche [Bet14].

### 6.2.5 Personenerkennung

Der Roboter AMEE-XW2 überrennt kleine Hindernisse, um durchsetzungsfähig gegenüber seiner Umwelt zu bleiben. Das vorgestellte Verfahren (4.3.3.1.2) versucht auch einen menschlichen Arm oder ein Bein zu überrennen, wenn nicht genügend Messpunkte des Sensors auf dem Körperteil liegen. Als

ersten Ansatz bietet sich hier die API der Kinect V2 an, die eine relativ sichere Personenerkennung anbietet. Die API gibt die Positionsdaten der Gelenke von bis zu vier Personen zurück. Damit könnten die Bewertungsmatrizen beeinflusst werden und eine Person könnte wie beispielsweise eine Wand umgangen werden.

### 6.2.6 Userinterface und Visualisierung

Bei AMEE handelt es sich zwar um einen autonomen Roboter, aber eine Online-Visualisierung der Umweltsensordaten würde das System sinnvoll erweitern. Da der Sensor Punktwolken generiert, bietet sich eine dreidimensionale Visualisierung in einer Third-Person Ansicht an. D.h. der Operator hätte eine frei drehbare und „schwebende“ Ansicht der erfassten Sensordaten. Diese Art der Visualisierung und ggf. Steuerung des Roboter macht die Einschätzung beispielweise einer Gefahrensituation für den Operator leichter.
Problematisch ist hier nur die zu übertragende Datenmenge, die durch Punktwolken entsteht. Es gibt aber Bestrebungen ein Verfahren zur Punktwolkenkomprimierung zu finden.

# 7 Anhang

## 7.1 Roboter AMEE-XW2


*Abbildung 7-1 (oben links) Testaufbau, (unten links) Größenvergleich*

## 7.2 Nacht des Wissens 2013

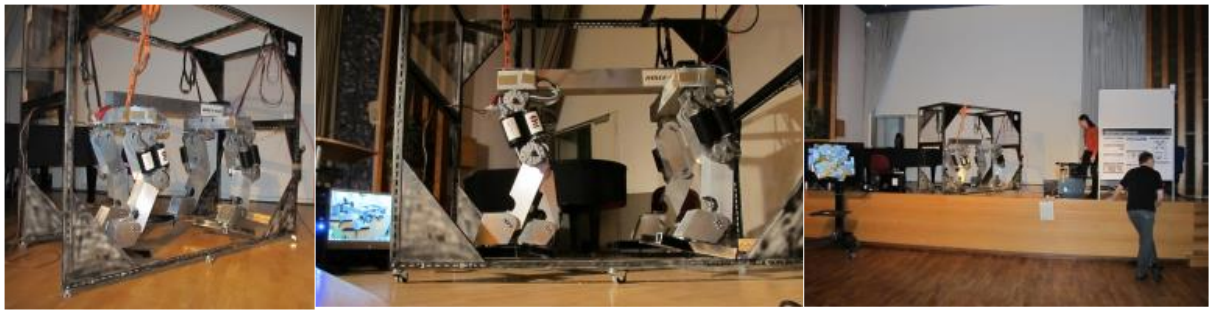

*Abbildung 7-2 Roboter AMEE-XW2 auf der Nacht des Wissens 2013*

## 7.3 Montage

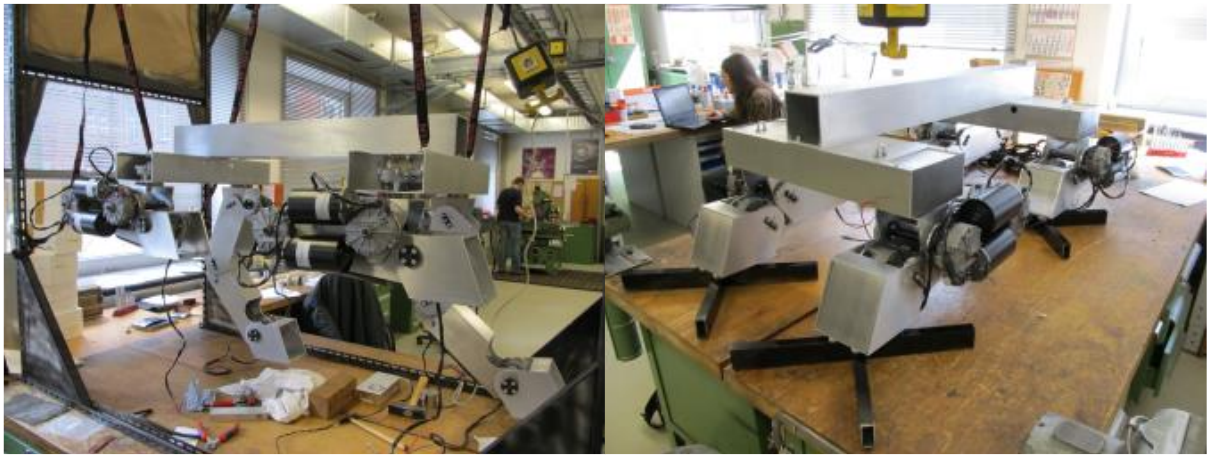

*Abbildung 7-3 (links) Erster Bewegungstest, (rechts) Parkposition der Beine*

## 7.4 Boston Dynamic® BigDog Softwarearchitektur

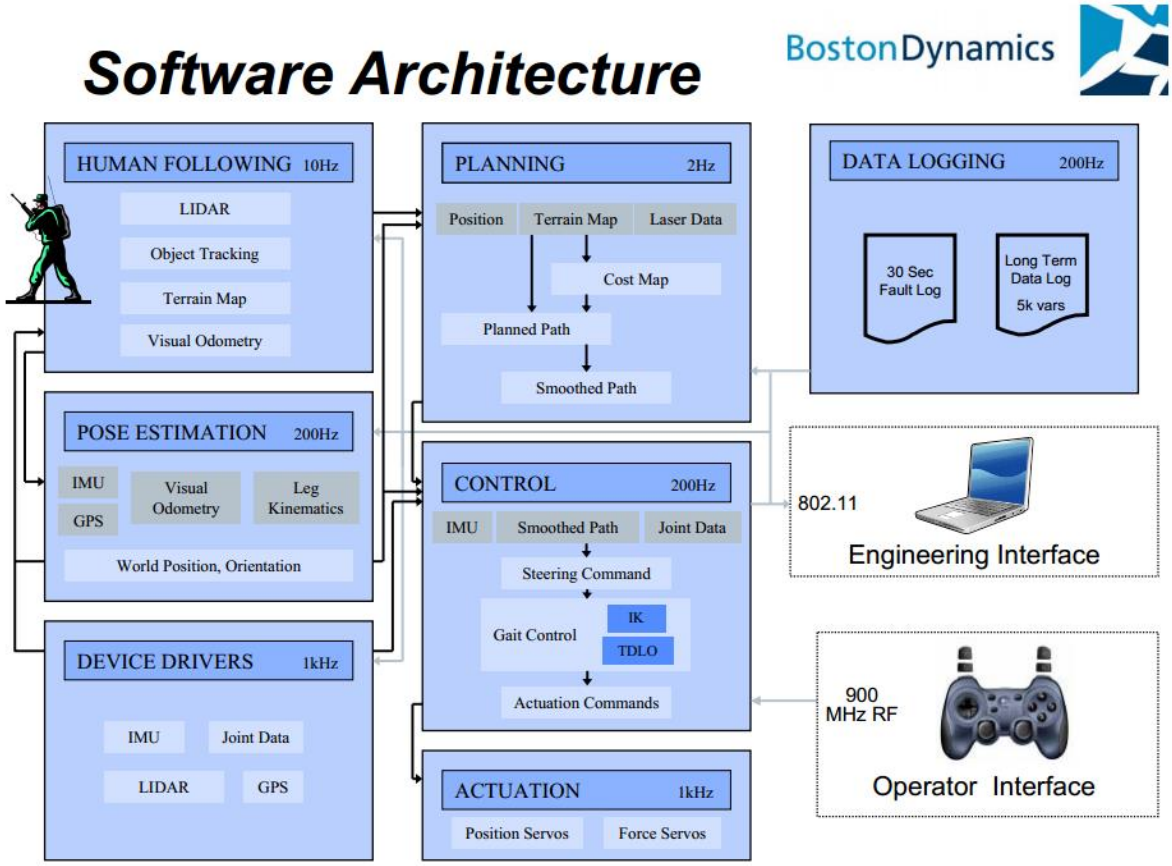


*Abbildung 7-4 Softwarearchitektur BigDog von Boston Dynamics [Bos10]*

## 7.5 Zero Spin Control

Laufende Maschinen zeigen oft einen Drehimpuls beim Aufsetzen der Füße, der sich als Trägheitsmoment auf den Massenschwerpunkt des Roboters äußert. Untersuchungen am MIT-Leg-Lab haben gezeigt, dass laufende Menschen einen unerwartet geringen Drehimpuls auf ihren Massenschwerpunkt ausüben [Pop05]. Die Schlussfolgerung dieser Arbeit war, dass es einen dezentralen „biologischen“ sog. *Zero Spin Controller* im zentralen Nervensystem geben muss. Mark Raibert (MIT / Boston Dynamics) spekulierte schon 1986 in seinem Buch „Legged Robots That Balance“ über mögliche Auswirkungen auf einen dynamischen Läufer. Erst im Jahr 2003 konnte Kajita [Kajita et al. (2003; 2004)] am Roboter HRP-2 beweisen, dass die Implementierung einer Spin-Control bei einem Zweibeiner den Roboter ruhiger und natürlicher laufen lässt. Für den Roboter AMEE ist es wahrscheinlich nicht nötig, eine Zero Spin Control zu verwenden. Der Zero Spin wirkt sich stärker aus,

je höher der Massenschwerpunkt liegt. Der Massenschwerpunkt liegt bei dem Roboter AMEE sehr niedrig (4.3.3.3). Die Abstraktion zum Zero Moment Spin Controller umfasst eine relativ komplexe Herleitung und wird hier nicht genauer vorgestellt. Das Zero Moment Point (ZMP) Modell wird um mehrere abstrakte Punkte erweitert. Stark vereinfacht ausgedrückt, wird ein theoretischer Kreuzungspunkt betrachtet. Er wird über den ZMP und Kraftvektor des Fußes gebildet und auf den Boden projiziert. Durch die Lage dieses Punktes im Verhältnis zum Stützpolygon kann die auszuübende Kraft errechnet werden, um den Roboter besser zu stabilisieren. Die Arbeit „Ground Reference Points in Legged Locomotion: Definitions, Biological, Trajectories and Control Implications“ [Pop05] beschreibt das Verfahren und das Gleichungssystem genauer. Die Forschung daran ist noch nicht vollständig abgeschlossen und ist für das Verständnis dieser Thesis nicht nötig.

# Literaturverzeichnis


[Par12] **Parness, Aaron . 2012.** Gripping Foot Mechanisms for Anchoring and Mobility in Microgravity and Extreme Terrain. *NASA Jet Propulsion Laboratory .* [Online] 09 30, 2012. [Cited: 07 08, 2014.] http://www-robotics.jpl.nasa.gov/tasks/showTask.cfm?FuseAction=ShowTask&TaskID=206&tdaID=700015.

[Aca03] **Acar, Ercan U., et al. 2003.** *Path Planning for Robotic Demining: Robust Sensor-based Coverage of Unstructured Environments and Probabilistic Methods.* Carnegie Mellon University, Pittsburgh, PA 15213 USA : The International Journal of Robotics Research, 2003. Vol. 22, No. 7–8, July–August 2003, pp. 441-466.

[Ale11] **Alexander Shkolnik, Michael Levashov, Ian R. Manchester and Russ Tedrake. 2011.** *Bounding on Rough Terrain with the LittleDog Robot.* [PDF] MASSACHUSETTS,USA : MASSACHUSETTS INST OF TECH, 2011. DOI: 10.1177/0278364910388315.

[Ani09] **Anish Adukuzhiyil, Harshit Singh, Pavani Vantimitta. 2009.** *Robot Motion for Obstacle Negotiation.* [PDF] USA : Stanford University , 2009.

[Bet14] **Bettzüche, Björn. 2014.** *Lernverfahren für die Wahl sicherer Schrittpositionen .* HAW Hamburg : HAW Hamburg, 2014. Master-Thesis.

[Bet10] **—. 2010.** *Machbarkeitsprüfung zur Entwicklung von SW-Anwendungen mit MS-Robotics Developer Studio für das Robocup Rescue Szenario.* [PDF] s.l., Hamburg : HAW Hamburg, Technische Informatik, Juli 2010.

[Bos131] **Boston Dynamics . 2013.** Boston Dynamics Home Page. *CHEETAH - Fastest Legged Robot.* [Online] Boston Dynamics , 2013. [Cited: April 13, 2014.] http://www.bostondynamics.com/robot_cheetah.html.

[Bos13] **Boston Dynamics. 2013.** Boston Dynamics, The LS3. [Online] Boston Dynamics, 02 09, 2013. [Cited: 02 09, 2013.] http://www.boston-dynamics.com/robot_ls3.html.

[Bos10] **Boston Dynamics Inc. 2008, 2010.** *BigDog Overview.* [Internet] 78 Fourth Avenue, Waltham, MA, 02451-7507, US : Boston Dynamics Inc., 2008, 2010. http://www.bostondynamics.com.

[Bra95] **Branicky, Michael . 1995.** *Studies in Hybrid Systems: Modeling, Analysis, and Control.* Cambridge : MIT Press, LIDS , 1995. ASIN: B001A3GWDK.

[Dan12] **Daniel Kuehn, et al. 2012.** *Additional DOFs and Sensors for Bio-inspired Locomotion: Towards Active Spine, Ankle Joints, and Feet for a Quadruped Robot.* Turin : In Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2012), 2012. iSAIRAS-12.

[Def14] **Defense Advanced Research Projects Agency . 2014.** Defense Advanced Research Projects Agency . *Legged Squad Support System (LS3).* [Online] Defense Advanced Research Projects Agency , 2014. [Cited: 07 19, 2014.] http://www.darpa.mil/Our_Work/TTO/Programs/Legged_Squad_Support_System_(LS3).aspx.

[Def05] **Defense Advanced Research Projects Agency. 2005.** Learning Locomotion. USA : Goverment USA, 2005.

[Fai11] **Faisal , Mufti, Robert , Mahony and Jochen , Heinzmann. 2011.** *Robust estimation of planar surfaces using spatio-temporal RANSAC for applications in autonomous vehicle navigation.* School of Engineering, College of Engineering and Computer Science, Australian National University, Canberra, ACT 0200, Australia : ELSEVIER Robotics and Autonomous Systems, 2011. Robotics and Autonomous Systems 60 (2012) 16–28.

[Fat11] **Fathi, Habib and Brilakis, Ioannis . 2011.** *Automated sparse 3D point cloud generation of infrastructure using its distinctive visual features.* Atlanta, GA 30332, USA : 2011 Elsevier Ltd, 2011. doi:10.1016/j.aei.2011.06.001.

[Gal10] **Gallo, Orazio, Manduchi, Roberto and Rafii, Abbas. 2010.** *CC-RANSAC: Fitting planes in the presence of multiple surfaces in range data.* University of California, Santa Cruz und Canesta, Inc., United States USA : ELEVIER Pattern Recognition Letters 32, 2010. Pattern Recognition Letters 32 (2011) 403–410.

[HLe06] **H. Lee, et al. 2006.** *Quadruped robot obstacle negotiation via reinforcement learning.* s.l. : IEEE International Conference on Robotics and Automation, 2006.

[Hon071] **Honda. 2007.** *ASIMO Technical Information.* [PDF] Japan / Germany : Honda Motor Co., Ltd., Public Relations Division, 2007.

[Hor01] **Hormann, Kai and Agathos, Alexander . 2001.** *The point in polygon problem for arbitrary polygons.* Erlangen, Germany : Elsevier Science B.V., 2001. Computational Geometry 20 (2001) 131–144.

[Jan10] **Janssen, Herbert, Principal Scientist. 2010.** Personal Communication Honda ASIMO. [EMail/Phone]. Honda Research Institute Europe GmbH, Carl-Legien-Strasse 30, 63073 Offenbach/Main, Germany : s.n., 2010.

[Jea10] **Jean-Emmanuel , Deschaud and Francois , Goulette. 2010.** *A Fast and Accurate Plane Detection Algorithm for Large Noisy Point Clouds Using Filtered Normals and Voxel Growing.* Mines ParisTech, CAOR-Centre de Robotique, Mathematiques et Systemes 60 Boulevard Saint-Michel 75272 Paris Cedex 06 : s.n., 2010.

[Kaj08] **Kajita, Shuuji and Espiau, Bernard . 2008.** *Handbook of Robotics.* Heidelberg, Germany : Springer-Verlag, 2008. 978-3-540-23957-4.

[Mic14] **Kaku, Michio. 2014.** *THE FUTURE OF THE MIND.* s.l. : Contents, 2014. 978-0385530828.

[Köc05] **Köckritz, Oliver . 2005.** *Visuomotorische Bewegungskoordination für mobile Roboter.* HAW-Hamburg : HAW-Hamburg, 2005.

[And09] **Kolter, J. Zico, Andrew, Y. and Youngjun, Kim. 2009.** *Stereo Vision and Terrain Modeling for Quadruped Robots.* [PDF] CA 94305, USA : Stanford University, Stanford, 2009.

[Kol09] **Kolter, J. Zico, Rodgers, Mike P. and Andrew, Y. 2009.** *A Control Architecture for Quadruped Locomotion Over Rough Terrain.* [PDF] CA 94305 USA : Computer Science Department, Stanford University, Stanford, 2009.

[Mil06] **Millard F., Beatty. 2006.** *Principles of Engineering Mechanics, Volume 2 Dynamics -- The Analysis of Motion.* s.l. : Springer-Verlag GmbH, 2006. ISBN 978-0-387-31255-2.

[Mur08] **Murphy, Robin R., et al. 2008.** Search and Rescue Robotics. [book auth.] Bruno Siciliano and Khatib Oussama. *Springer Handbook of Robotics.* Berlin : Springer, 2008, pp. 1151-1173.

[Nac10] **Nachtigall, Werner. 2010.** *Bionik als Wissenschaft.* Heidelberg : Springer-Verlag, 2010. ISBN 978-3-642-10319-3 .

[Pfe07] **Pfeifer, Rolf , Bongard, Josh and Grand, Simon. 2007.** *How the body shapes the way we think : a new view of intelligence.* Cambridge, Mass. : MIT Press, 2007.

[Pop05] **Popovic, Marko B., Goswami, Ambarish and Herr, Hugh. 2005.** *Ground Reference Points in Legged Locomotion: Definitions, Biological, Trajectories and Control Implications.* Cambridge, MA 02139-4307, USA, Mountain View, CA 94041, USA : SAGE Publications, The International Journal of Robotics Research Vol. 24, No. 12, December 2005, pp. 1013-1032, 2005. DOI: 10.1177/0278364905058363.

[Pop04] **Popovic, Marko B., Herr, Hugh M. and Hofmann, Andreas G. . 2004.** *Angular Momentum Regulation during Human Walking: Biomechanics and Control.* Cambridge, MA 02139-4307, USA : ICRA 2004, 2004. dx.doi.org/10.1109/ROBOT.2004.1307421.

[Reb08] **Rebula, John R., et al. 2008.** *A Controller for the LittleDog Quadruped Walking on Rough Terrain.* [PDF] Florida 32502, USA : Florida Institute for Human and Machine Cognition, 2008. jrebula@alum.mit.edu.

[Rod86] **Rodney A., Brooks. 1986.** *A robust layered control system for a mobile robot.* Massachusetts, USA : Journal of IEEE, 1986. ISSN: 0882-4967.

[Rou98] **Rourke , Joseph O'. 1998.** *Computational Geometry in C (Second Edition).* Cambridge : Cambridge University Press, 1998. ISBN 0521640105.

[Ruh121] **Ruhnke, Jan. 2012.** *A Walksystem for a Quadruped Rough Terrain Robot, Controller Concepts AW1.* [PDF] Hamburg, Germany : HAW-Hamburg, 2012. AW1.

[Ruh11] **—. 2011.** *Entwicklung und Realisierung eines vierbeinigen USAR-Roboter-Laufsystems.* [PDF] Hamburg, Germany : HAW-Hamburg Dep. Technische Informatik, Juni 02, 2011. Bachelor Arbeit.

[Ruh123] **—. 2012.** *Projekt 1, Project AMEE a Quadruped Rough Terrain Robot.* Hamburg, Germany : HAW-Hamburg, 2012.

[Ruh12] **—. 2012.** *Simulation der Drehmomente in einem Roboter Bein, Ausarbeitung Modellierung Technische Systeme .* HAW-Hamburg : s.n., 2012.

[Sch96] **Schneider, Eberhard and Seilmeier, Gerhard. 1996.** Körperbau Seite 439. *Jagd Lexikon.* München : BLV Verlagsgesellschft mbH, 1996.

[Spe05] **Spenneberg, Dirk, et al. 2005.** *ARAMIES: A FOUR-LEGGED CLIMBING AND WALKING ROBOT.* [PDF] D-28359 Bremen, Germany : University of Bremen, Robotics Lab, Faculty of Mathematics and Computer Science, 2005.

[Spr09] **Sproewitz, A., et al. 2009.** *Compliant Leg Design for a Quadruped Robot.* Switzerland, Germany : Biologically Inspired Robotics Group, EPFL, Switzerland, TU Ilmenau, Germany, 2009.

[Sut98] **Sutton , Richard S. and Barto, Andrew G. . 1998.** *Reinforcement Learning: An Introduction.* Cambridge, Massachusetts : MIT Press, 1998.

[Tan11] **Tanaka, Shogo. 2011.** *The notion of embodied knowledge.* [PDF] Japan : Captus University Publications, 2011.

*Visual Odometry, Part II: Matching, Robustness, Optimization, and Applications.* [Fra12] **Fraundorfer, Friedrich and Scaramuzza, Davide. 2012.** JUNE 2012, s.l. : IEEE ROBOTICS & AUTOMATION MAGAZINE, 2012. 10.1109/MRA.2012.2182810.

[Wet95] **Wettergreen, David, Pangels, Henning and Bares, John. August 1995.** *Behavior-based Gait Execution for the Dante II Walking Robot.* [PDF] Pittsburgh, PA 15213-3891, USA : Carnegie Mellon University, August 1995.

[Zha05] **Zhang, Lei , et al. 2005.** *Omnidirectional Static Walking of a Quadruped Robot on a Slope.* Hitachi-Shi 316-8511, Japan : Journal of Robotics and Mechatronics Vol.18 No.1, 2006, 2005.

[Zha051] **Zhang, Lei , et al. 2005.** *Omnidirectional Static Walking of a Quadruped Robot on a Slope.* Department of Systems Engineering, Faculty of Engineering, Ibaraki University, Japan : Journal of Robotics and Mechatronics, 2005. Vol.18 No.1, 2006.

# Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23.07.2014

Ort, Datum

Unterschrift