Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Björn Bettzüche

Lernverfahren für die Wahl sicherer Schrittpositionen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Björn Bettzüche

Lernverfahren für die Wahl sicherer Schrittpositionen

Abschlussarbeit zum Erlangen des akademischen Grades Master of Science
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am 10.10.2014

Björn Bettzüche

Thema der Master Thesis

Lernverfahren für die Wahl sicherer Schrittpositionen

Stichworte

Maschinelles Lernen, Reinforcement Learning, Roboter, vierbeinige Roboter, unebenes Gelände

Kurzzusammenfassung

Diese Arbeit stellt ein Verfahren zum Erlernen sicherer Schrittpositionen vor, mit dem Ziel, dass sich der vierbeinige Roboter AMEE sicher in unebenem Gelände fortbewegen kann. Die autonom zu bewältigende Aufgabe ist es, den Boden der näheren Umgebung zu erfassen und Eigenschaften zu ermitteln. Ob eine Trittposition gut ist, wird durch ausprobieren und Analyse der Tritteigenschaft erlernt, sodass künftig Bodenabschnitte bezüglich ihrer Laufqualität bewertet werden können. Reinforcement Learning dient als lernalgorithmisches Gerüst.

Title of the paper

Learning method for safer foothold selection

Keywords

Machine learning, Reinforcement Learning, robotics, quadruped robot, rough terrain

Abstract

This work presents an approach for learning secure step positions, with the objective that the four-legged robot AMEE can safely walk in rough terrain. The autonomous task to be mastered is to identify features of the surrounding area. The suitability of a step position will be learned by doing and analyzing the step characteristic so that floor sections can be evaluated for their running quality. Reinforcement Learning serves as framework of the machine learning method.

Inhaltsverzeichnis

Vorwort

Groß war die Aufregung seinerzeit, als 2007 Studiengebühren von erst 500,- und dann reduzierten 350,- Euro pro Semester eingeführt wurden. Doch der neue Hamburger Senat von 2011 war sicherlich froh darüber, denn schließlich konnte er sich durch die Abschaffung als Wohltäter profilieren. Doch auch ich muss mich wohl im Nachhinein als dankbar für die Gebühren und vor allem dankbar der Hochschule für ihre Verwendung zeigen. Denn ohne die daraus finanzierten Gelder für studentische Projekte, so Bescheiden sie auch sind, wäre es meinem Projektpartner und mir nicht möglich gewesen, die Entwicklung eines vierbeinigen, autonomen Roboters in Angriff zu nehmen und unsere Visionen zu verwirklichen. In dem Zusammenhang auch ein Dank an unsere Sponsoren Hoffmann + Krippner GmbH, Ott Antriebe GmbH und Microsoft USA Ltd.

Beim Leser muss ich mich im Voraus allerding für das Farbdesign dieser Thesis vielleicht entschuldigen. Es hat etwas Corporate Identity zu tun, denn es werden die offiziellen Farben der HAW Hamburg und ihrer Fakultäten verwendet, überwiegend das grün der Fakultät Technik und Informatik.

1 Einleitung

„Ein Spaziergang am Elbufer[1] in Hamburg-Altona. Er beginnt mit dem Abstieg Richtung Ufer über eine Jahrhunderte alte, enge, steile und abgelaufene Treppe. Dann der weiche und feine Sand, der in die Schuhe rieselt und die Schritte schwer werden lässt bis zur Uferpromenade. Die verrotteten Betonplatten scheinen beinahe chaotisch, wie dahin geworfen, verlegt zu sein aber man kommt wieder gut voran, vorausgesetzt entgegenkommende Spaziergänger zwingen einen nicht erneut auf den Strand. Nach wenigen hundert Metern ist der befestigte Weg auch schon zu ende. Es ist an der Zeit, den Füßen eine Abkühlung im Wasser zu gönnen. Vorbei an einigen Büschen, über Treibgut hier und da, geht es auf dem deutlich festeren weil feuchten Sandstrand in die Elbe, bis die Knöchel vom Wasser bedeckt sind. Dann immer weiter entlang der Uferlinie. Jede Welle lässt das Wasser auf- und abfließen und reißt dabei ein wenig des Bodens unter den Füßen weg, sodass man immer wieder ein wenig einsackt und ins Wanken gerät. Nach einer Weile ändert sich der Untergrund, der zunehmend von Kieselsteinen und Muscheln bedeckt ist, bis sich vor einem eine flache Mauer aus Findlingen auftut, an denen sich die sonst sanften Wellen brechen. Der Weg rauf zur Strandpromenade ist durch dichtes und dorniges Gestrüpp versperrt. Es bleibt einem die Wahl entweder ein ganzes Stück wieder zurück zu gehen oder über die glitschigen Felsbrocken zu steigen. Mühsam und auf jeden Schritt bedacht kämpft man sich also über die Wellenbrecher. Immer wieder rutscht man ab und fängt sich nur mit Mühe, aber ein Schritt nach dem anderen ist auch diese ungemütliche Passage überwunden.“

Ein harmlos anmutender Spaziergang kann schon aufzeigen, wie herausforderungsvoll das Laufen sein kann. Wir gehen intuitiv mit unterschiedlichsten Bodenbedingungen um, stoßen aber auch an unsere Grenzen, sobald die Welt rauer und schroffer wird, wie bei der Felsmauer. Zunächst eine grundsätzliche Abwägung darüber, welcher Weg der bessere ist. Drüber weg oder zurück und außen vorbei? Dann werden die Schritte zunehmend mit Bedacht gewählt. Bietet der Stein vor mir genug halt oder laufe ich Gefahr, abzurutschen? Je riskanter das Gelände wird und je weniger Erfahrung man damit hat, desto mehr wird überlegt, kalkuliert, abgewogen.

In der Robotik wird intensiv an der laufenden Fortbewegung geforscht, da sie auch die Erschließung unstrukturierter Umgebungen ermöglicht, wo radgetriebene Fahrzeuge an ihre Grenzen stoßen. Hier ist das studentische Projekt AMEE[2] mit dem vierbeinigen, autonomen Roboter AMEE-XW2 angesiedelt, der für Such- und Erkundungseinsätze in rauen, unebenen, unstrukturierten Umgebungen ausgelegt ist. Ein schreitender Roboter, der selbständig über Stock und Stein, in Geröllwüsten wie dichten

[1] https://www.google.de/maps/@53.5439672,9.9064968,15z
[2] **A**utonomous **M**apping **E**xploration and **E**vasion

Wäldern oder sumpfigen Überschwemmungsgebieten wandern soll, muss sich mit genau der Problematik des Strandspaziergangs auseinander setzen: Wo setze ich den nächsten Schritt hin?

> „Festzustellen, was eine gute und sichere Schrittposition ist, ist aufgrund der Vielzahl an Untergründen und Einflüssen eine Frage der Erfahrung. Diese Erfahrung lässt sich für eine Maschine am besten durch bestärkendes Lernen erwerben."

1.1 Überblick

Diese Arbeit stellt ein Verfahren zum Erlernen sicherer Schrittpositionen vor, mit dem Ziel, dass sich der vierbeinige Roboter AMEE sicher in unebenem Gelände fortbewegen kann. Die autonom zu bewältigende Aufgabe ist es, den Boden der näheren Umgebung zu erfassen und Eigenschaften wie Steigung und Unebenheit zu ermitteln. Ob eine Trittposition gut ist, wird durch ausprobieren erlernt. Dabei wird der ausgeführte Schritt beispielsweise anhand der Ausführbarkeit, ob er rutschfest war und wie die Kraftaufnahme und Drehmomententwicklung war, also anhand dessen, wie sich der Schritt anfühlt bewertet.

Abgrenzung und Einordnung:

- Das Hauptaugenmerk liegt in der prinzipiellen Methodik des erfahrungsbasierten Lernens für die Aufgabe der sichern Schrittwahl.
- Die Analyse der Bodenbeschaffenheit oder Schrittwahrnehmung wird nur als theoretisch angenommene Analyseergebnisse simuliert. Es wird keine Methode zur Analyse vorgestellt.
- Es erfolgt keine Pfadplanung, sondern es wird nur die unmittelbare Umgebung bewertet. Die Pfadplanung oder zumindest eine Zielrichtung wird als gegeben vorausgesetzt.
- Es erfolgt keine Steuerung der Beine, sondern nur eine Empfehlung in Form einer Art Kostenkarte für Schrittpositionen. Die letztliche Auswahl der konkreten Position sowie deren Durchführung erfolgt durch AMEEs Lauf-Kontroller.
- Auch wenn die Motivation bei einem vierbeinigen, geländegängigen Roboter liegt, sind die methodischen Analysen der betrachteten Lernaufgabe und des Lernverfahrens auch auf Zwei- oder Sechsbeiner oder auf Fahrzeuge, ob Kette oder Räder anwendbar.

Aufbau dieser Arbeit:

① **Einleitung**
Die lesen Sie gerade.

② **Grundlagen**
- Überblick der zu verwendenden Technologien und vergleichende Ansätze.

③ **Randkonzepte**
- LittleAMEE: Ein Softwaretechnisches Konzept für eine verteilte Roboteranwendung mit Echtzeitanteilen in der Beinsteuerung und ereignisgesteuerten Modulen der höheren Logik im Roboterverhalten.
- Ein Robot Vision Ansatz zur visuellen Erfassung der näheren Umgebung und Vorverarbeitung im Kontext der Lernaufgabe.
- Reinforcement Learning: Aufbau eines modularen, flexiblen Software Frameworks.

④ **Modellierung der Lernaufgabe**
- Beschreibung des Modells und der Datenstrukturen, zum Erlernen einer Trittpositionswahlstrategie mit Reinforcement Learning.
- Umgang mit kontinuierlichen Wertebereichen.
- Modellierungsvarianten zur Verkleinerung des Zustandsraums und Beschleunigung des Lernfortschritts.

⑤ **Versuchsaufbau**
- Modellierung eines Test-Szenarios.
- Definition des Tests und der Kennzahlen.

⑥ **Versuchsergebnisse und Analyse**
- Schrittweise Analyse von Parametern und Modellierungsvarianten zur Verbesserung des Lernverfahrens.

⑦ **Ausblick**
- Weiterführende Ansätze
- Offene Herausforderungen

1.2 Motivation

Roboter, die außerhalb der Fabrik oder Laborumgebung agieren sollen, haben potentiell eine sehr viel höhere Varianz bei der Durchführung einer bestimmten Aufgabe. So sind während der Fortbewegung

beispielsweise Bodenprofil und -eigenschaften oft unbekannt und über die Zeit veränderlich. Obwohl es eine Vielzahl von Beweggründen gibt, gehende mechanische Geräte zu studieren, liegt der Schwerpunkt dieser Arbeit auf der Erreichung robuster Fortbewegung für einen Roboter, insbesondere der Fortbewegung auf Beinen, die einige Vorteile gegenüber Rädern haben kann, vor allem dann, wenn der Pfad abseits ebener Wege führt. Konkret geht es im Forschungsprojekt AMEE um die Entwicklung eines vierbeinigen Roboters, vornehmlich für Erkundungseinsätze in Katastrophenszenarien.

Roboter mit Beinen bieten die Möglichkeit, in sehr anspruchsvollem Gelände zu navigieren, und es hat in letzter Zeit große Fortschritte in diesem Bereich gegeben. Es wurde jedoch ein großer Teil dieser Arbeit unter der Annahme betrieben, dass entweder der Roboter vollständiges Wissen über seine Umwelt besitze oder seine Umgebung nur im angemessenen Rahmen uneben sei, so dass mit nur geringen Wahrnehmungen navigiert werden kann. Eine unrealistische Annahme in vielen wirklichen Domänen. Ziel dieser Arbeit ist eine integrierte Wahrnehmung und Steuerung für einen vierbeinigen Roboter, die es ihm ermöglicht, bisher unbekanntes, zerklüftetes Gelände, welches große, unregelmäßige Hindernisse beinhalten kann, zu erkennen und zu durchqueren (vgl. auch [Kol09]).

1.3 Stand der Technik

Obwohl in den letzten zwanzig Jahren eine Vielzahl mechanischer Konzeptstudien zur schreitenden Roboterfortbewegung durchgeführt wurden und Honda bereits Mitte der 90‘er Jahre medienwirksam den Zweibeiner ASIMO vorstellte [Hon07], ist das autonome Gehverhalten prinzipiell in der Regel noch nicht zufriedenstellend umgesetzt und weiterhin Gegenstand der aktuellen Forschung. Vergleichend zum Projektrahmen dieser Arbeit, AMEE, werden folgend einige Roboter vorgestellt, die den aktuellen Stand der Technik repräsentieren.

BigDog

BigDog oder sein Nachfolger AlphaDog ist ein großer, dynamischer von Boston Dynamics hergestellter Vierbeiner. Er ist bis heute bei weitem der beeindruckendste Roboter, der mit unwegsamem Gelände umgehen kann, wobei die meisten Details über Gestaltung und Steuerung unveröffentlicht bleiben. Einige der bei BigDog verwendeten Kontrollerprinzipien basieren wahrscheinlich auf der Arbeit aus den 1980er und 90er Jahren von Marc Raibert und seinen Schüler am MIT. Der Kontroll-Ansatz für BigDog stützt sich offensichtlich auf die Regelung der Kräfte und Haltung des Roboters. BigDog ist sehr agil und hat bewiesen, dass er durch bewusste Platzierung der Füße und Körper-Lageregelung mit erheblichen Störungen umgehen kann. Allerdings ist BigDog noch weitgehend blind für kommendes Gelände und reagiert scheinbar weitgehend reflexartig, um dynamisch das Gleichgewicht zu halten [Rai08].

LittleDog

Der ebenfalls von Boston Dynamics stammende LittleDog ist ein vierbeiniger Roboter, entwickelt für die Erforschung des Erlernens von Fortbewegung („Learning Locomotion“). Wissenschaftler an führenden Institutionen nutzen LittleDog, um die grundlegenden Beziehungen zwischen motorischem Lernen, dynamischer Kontrolle, Wahrnehmung der Umwelt und geländegängige Fortbewegung zu sondieren. LittleDog wird am MIT, Stanford, Carnegie Mellon, USC, Uni. Pennsylvania und IHMC als Teil eines aus DARPA-Mitteln finanzierten Programms für zukunftsweisende Robotik verwendet. LittleDog hat vier Beine, jeweils von drei Elektromotoren angetrieben. Der integrierte Computer erledigt Sensorik, Aktor-Steuerung und Kommunikation. Die Sensoren des LittleDog messen Gelenkwinkel, Motorströme, Körperorientierung und Fuß- / Bodenkontakt. Steuerungsprogramme greifen auf den Roboter durch die Boston Dynamics Robot API zu. [Kal11] [Reb08]

ARAMIES / iStruct

Das ARAMIES-Projekt vom Bremer DFKI [Hil06] beschäftigte sich mit der Entwicklung eines Laufroboters, der in schwierigem Gelände autonom operieren kann. Einsatzbereiche sollten z.B. Canyons und Krater auf dem Mond oder Mars sein. Das Projekt „ARAMIES“ verwendete ein Laufsystem mit elektrischen Antrieben und versuchte einen „biologischen Kontroller“ umzusetzen. Es wurden aber feste Laufmuster (Walking Gaits) verwendet, die nach Erfahrungen aus dem DARPA-Programm „Learning Locomotion“ [Reb08] [Est05] nicht immer vorteilhaft sind aber im Vergleich zur üblichen modellbasierten Robotersteuerung den nötigen Rechenaufwand zugunsten hoher Energieeffizienz stark minimiert. Das Projekt ARAMIES wurde 2007 eingestellt. Im Jahr 2014 stellte das DFKI den iStruct „Charlie“ [iSt13] vor, eine biologisch inspirierte Konstruktion, die an einen Affen erinnert und als Forschungsplattform für Roboter Algorithmen dient. Ein wesentliches Merkmal ist die aktuierte Wirbelsäule.

1.4 Zielsetzung

Ein Kernpunkt zur Handhabung der oben beschriebenen Aufgabe, Laufen durch unwegsames Gelände, ist die Auswahl einer sicheren Fußpositionierung, insbesondere da der Roboter AMEE als (semi-) statischer Läufer umgesetzt werden soll, [Ruh11] [Ruh12]. Der Charakter dieser Herausforderung ist mit dem menschlichen Klettern vergleichbar, wobei die Fuß- und Hand-Halte-Selektion einer der wichtigsten Aspekte ist. In dieser Arbeit werden statische, bzw. semistatische Läufer betrachtet, bei denen der Masseschwerpunkt (Center of Gravity COG) bzw. der Zero Moment Point ZMP immer im Stabilitätspolygon gehalten wird [Ruh11], [Byl08]. Unter diesem Aspekt gewinnt die Fußpositionierung an zusätzlicher Bedeutung. Frühere Arbeiten in diesem Bereich umfassen in der Regel die Definition einer Belohnungsfunktion für Fußstellungen als eine gewichtete Linear-kombination von Eigen-schaften des Geländes wie Steilheit und Unebenheit. Allerdings muss ein erheblicher Aufwand in die

Entwicklung dieser Merkmale gesteckt werden, um komplexere Entscheidungsfunktionen zu modellieren und ein manuelles Abstimmen ihrer Gewichte ist keine triviale Aufgabe, insbesondere da sie direkten Einfluss auf die metaphysische Bewertung darüber haben, was eine gute Trittposition ist (z.B. Rutschsicherheit vs. Bahnfortschritt) und diese Ziele oftmals kontextabhängig im Konflikt zueinander stehen. Alternative Ansätze verfolgen überwachte Lernmethoden, um den Boden zu klassifizieren und die Gangart darauf anzupassen [Kim10] oder anhand von erlernten Geländevorlagen eine Bewertung der Trittposition vorzunehmen [Kal11].

Ziel dieser Arbeit ist es, ein fortwährend lernendes Modul zu entwickeln, welches die unmittelbare Umgebung wahrnimmt und eine Auswahl guter Trittpositionen unter Aspekten der Robustheit, Erreichbarkeit, Kollisionsfreiheit und des Bewegungsfortschritts vornimmt, wobei das Reinforcement Learning , welches ein Lernen auf Basis von Erfahrungen modelliert, als lernalgorithmisches Gerüst dient. Anhand der visuellen Wahrnehmung der Umgebung (Exterozeption) wird der Untergrund bezüglich seiner Tritteigenschaft eingeschätzt, woraufhin in der Regel die vorteilhafteste Position selektiert wird. Während des Ausführens des Schritts geben dann propriozeptive Daten der Beinsensorik eine Rückmeldung über die tatsächlichen Laufeigenschaften und werden zur Verbesserung künftiger Schätzungen herangezogen.

1.5 Vorgehensweise

Das angestrebte Ziel der eigenen Forschung ist es, eine a priori Bodenanalyse unter Aspekten der Standfestigkeit und Laufeigenschaft vorzunehmen. Dies soll für möglichst vielfältige und im Idealfall auch für unbekannte Terrainarten möglich sein. Die Beurteilung soll das Pilotsystem dahingehend unterstützen, dass die einzelnen Schrittpunkte oder der Laufstil den Gegebenheiten angepasst werden können. Da im Rough-Terrain-Szenario mit unterschiedlichsten Bodentypen zu rechnen ist, sollte das System fortwährend aus den Erfahrungen dazulernen und somit die Bewertungsfähigkeit über die Zeit ausbauen und verbessern.

Nicht Gegenstand dieser Arbeit ist die Pfadplanung oder die Regelung der Beinmechanik, letzteres ist durch frühere Arbeiten gegeben [Ruh11]. Die gesamte Schrittpositionsauswahl ist architektonisch zwischen diesen beiden, in **Abb. 1.1** grau dargestellten, Komponenten anzusiedeln und beinhaltet ihrerseits mehrere Komponenten unterschiedlicher Aufgaben. Die blauen Module „Rewarded Map Generator“, „Foot Planner“ und „Motion Controller“ entstehen in Zusammenarbeit mit Projektmitgliedern und sind ebenfalls nicht direkter Bestandteil dieser Arbeit. Das Augenmerk des vorgestellten Verfahrens liegt verallgemeinert in der Analyse der Laufeigenschaften aufgrund derer eine günstige Trittposition gelernt werden soll, gegeben durch die Module „Visual Analysis“, „Locomotion Analysis“ und „Learn Estimation“ mit einem Fokus auf das Lernverhalten. Für die Modellierung und

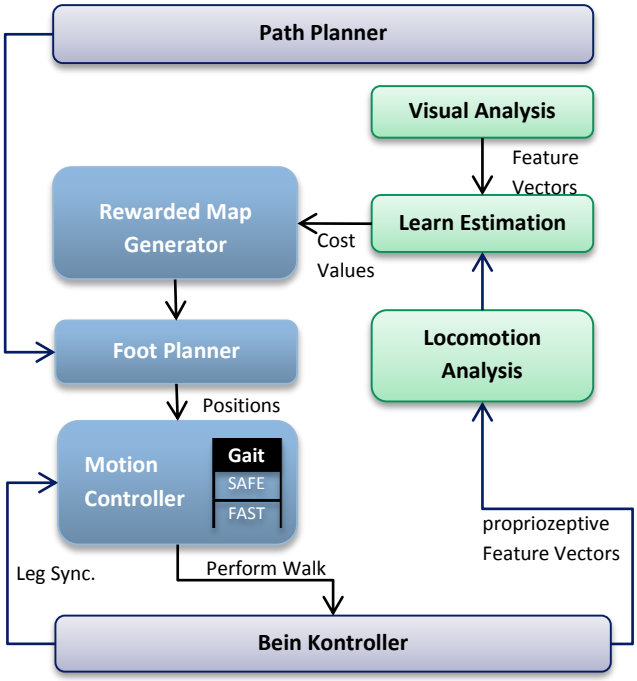


Abb. 1.1: Systemarchitektur. Dunkelblaue Module: Bewegungsumsetzung. Grüne Module: Foothold Selection.

Untersuchung des Lernverfahrens werden die visuelle und propriozeptäre Datenvorverarbeitung hier nur simuliert.

Der „Path Planner“ gibt einen groben Weg mit zu erreichenden Zwischenzielen vor, welche beispielweise durch einen Fernsteuerungsbefehl wie „geradeaus“ oder durch komplexere Pfadalgorithmen zur vollständigen Erkundung eines Terrains erzeugt werden. Mit Hilfe einer von „Rewarded Map Generator“ erzeugten Kostenkarte und den gegeben Pfadzielen wählt das Modul „Foot Planner“ die nächsten Schrittpositionen für einen vollständigen Schrittzyklus, für den Vierbeiner AMEE folglich vier Positionen, aus, welche schließlich vom „Motion Controller“ in Zusammenarbeit mit dem darunterliegenden „Bein Kontroller“ in eine mechanische Bewegung umgesetzt wird.

Lernverfahren

Diese Arbeit setzt die Erstellung der Kostenwerte durch ein lernalgorithmisches Verfahren im Modul „Learn Estimation“ um. Es bedient sich der durch „Visual Analysis“ gelieferten Umgebungsmerkmale, um eine Schätzung über die Kosten (Vorteilhaftigkeit) potentieller Trittpositionen aufzustellen und durch propriozeptive Merkmale der „Locomtion Analysis“ zu verifizieren, letztlich also „Erfahrungen“ für künftige Schätzungen zu sammeln. Es nutzt dazu das Reinforcement Learning [Sut12], ein erfahrungsbasierendes online Lernverfahren, welches in **2.3** in groben Grundzügen erläutert wird. Die eigentliche Herausforderung besteht weniger beim Lernalgorithmus, sondern viel mehr bei der Wahl, Ermittlung und Auswertung geeigneter Merkmalsvektoren. Sie ist Bestandteil der grundsätzlichen Modellierung der Lernaufgabe, der sich **Abschnitt 4** widmet.

Haptische Merkmalsextraktion

In erster Linie sollen Sensordaten, simulierte und reale, über Drehmomente der Gelenke und Kraftaufnahme an der Trittfläche als Kennlinien erfasst werden. Über das Zeitintervall eines Schrittes können daraus Merkmale wie Varianz, Wölbung und Verzerrung gewonnen werden, die zusammen mit odometrischen Daten wie Geschwindigkeit und zurückgelegter Strecke als Eingangsmerkmalvektor für den Lernalgorithmus dienen, um einen Schritt im Nachhinein zu bewerten. Aber auch so einfache Informationen von der Beinsteuerung wie „Position nicht erreicht“ oder „etwas angestoßen“ können schon als Rückmeldung verwendet werden. Der Grundlagenabschnitt **2.1.2** stellt Möglichkeiten der Propriozeption – wie sich ein Schritt „anfühlt“ – vor, die im Detail aber nicht weiter ausgearbeitet wird sondern nur simuliert in die Modellierung der Lernaufgabe einfließt (**4.1.2**). Eine mögliche erste rudimentäre Umsetzung im Roboter AMEE zeigt der Ausblick in **7.3** auf.

Visuelle Umgebungsmerkmale

Um einen Bodenabschnitt zu beurteilen, muss definiert werden, anhand welcher Merkmale dies festgemacht werden kann und wie diese zustande kommen. Einige Möglichkeiten der Datenerfassung und -vorverarbeitung durch das Modul „Visual Analysis“ stellt der Grundlagenabschnitt **2.1.1** vor. In **3.2** wird ein prototypischer erster Ansatz mit dem Kinect® V2 Sensor demonstriert, der auch in der parallelen Arbeit zum Hauptkontroller der Laufsteuerung von Ruhnke [Ruh14] verwendet wird. Darüber hinaus fließen die Betrachtungen abstrahiert in die Modellierung der Lernaufgabe ein **4.1.1**.

Tests

Für grundsätzliche Untersuchungen und Feinabstimmungen des Lernverfahrens, die klären sollen, wie gut das Verfahren zum Erlernen sicherer Schrittpositionen geeignet ist und unter welchen Bedingungen Schwierigkeiten zu erwarten sind, werden in **Abschnitt 5** ein Versuchsmodell und ein Testszenario entwickelt. Eine Analyse der Experimente erfolgt in **Kapitel 6**.

1.6 Zusammenfassung

Wie das DARPA-Programm „Learning Locomotion“ oder die Arbeiten von Boston Dynamics beweisen, ist die Erforschung vierbeiniger autonomer Roboter von hohem Interesse und erfordert weitere innovative Ansätze. Die potentiellen Vorteile bebeinter Fortbewegung sind gerade in schwierigem Gelände unumstritten. Darüber hinaus bietet die Forschung an Robotern gerade im Rescue-Szenario

vielfältige Herausforderung in beispielweise Computer Vision, künstlicher Intelligenz oder Informationssystemen. Diese Arbeit führt die bisherigen Forschungen zum vierbeinigen Roboter AMEE konsequent fort, indem ein erfahrungsbasiertes Auswahlverfahren für Trittpositionen unter Aspekten der Robustheit, Sicherheit und des Voranschreitens erforscht wird.

Bisherige Ansätze verwenden in der Regel feste, geregelt Gangmuster für allenfalls mäßig unebenes Gelände und für bekanntes Terrain. Es werden meist per offline Lernverfahren eine Handvoll gängiger Bodenarten erlernt und die Klassifizierungsmethoden dahingehend optimiert. Zwar haben die Forscher der Stanford University mit ihren Expert Templates dahingehend einen guten, generalisierten Ansatz, jedoch verwenden auch sie hochauflösende offline Scans des Testgeländes. Der BigDog von Boston Dynamics ist beeindruckend, jedoch als dynamischer Läufer nur bedingt mit AMEE vergleichbar (siehe [Ruh12]). Es wird augenscheinlich auf eine rein dynamische Regelung ohne vorausschauende Erfahrung gesetzt.

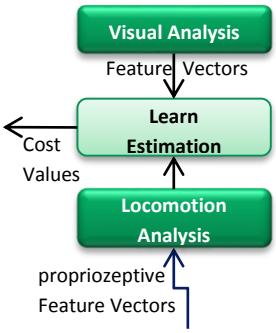
Diese Arbeit verfolgt einen ganzheitlichen Ansatz, indem Exterozeption und Propriozeption in einem online Lernverfahren kombiniert werden und dem System ermöglichen, auch ohne detailliertes Vorwissen in unbekanntem Terrain sicher voranzuschreiten, da es sich um ein fortwährend selbstlernendes System handelt. Die Architektur der Schrittpositionsauswahl ist modular und weitgehend generisch konzipiert, sodass Analysemethoden und -daten entsprechend der Roboter-konfiguration teilweise austauschbar sind.

2 Grundlagen & Vergleichende Arbeiten

Das in dieser Arbeit vorgestellte Verfahren beruht auf maschinellem Lernen und tangiert Bereiche der Bild- und Sensordatenverarbeitung. Die Grundlagen zu Kern- und Randaspekten auf denen hier aufgebaut wird, sollen nun kurz angerissen werden.

2.1 Datenvorverarbeitung

Das zu erlernende Verhalten – die Auswahl sicherer Schrittposition – beinhalten im Gesamtkonzept Umwelt- und Eigenwahrnehmung als Basis der Entscheidungsfindung des Lernmoduls. Das Erfassen und Verarbeiten solcher Eingangsdaten liegt nicht im unmittelbaren Fokus dieser Arbeit. Stattdessen werden hier einige vielversprechende Ansätze aus der Forschung vorgestellt, die in Folgearbeiten aufgegriffen werden können. An dieser Stelle bieten die vergleichenden Arbeitenden eine Vorstellung über und Ausblick auf die Randkomponenten des Lernmoduls *Learn Estimation*, auch als Grundlage für die Modellierung der Lernaufgabe.



2.1.1 Stereo Vision

Seit 2007 erforscht unter anderen die Stanford University im Rahmen eines DARPA-Projekts für zukunftweisende Fortbewegung mit Hilfe des von Boston Dynamics hergestellten Roboters LittleDog Algorithmen für die vierbeinige Fortbewegung. Während erste Arbeiten auf einem durch ein offline 3D-Scan entstandenes Geländemodell zurückgriffen, entwickelten Kolter et al. [Kol09] ein System, dass dem Roboter die Fortbewegung mit on-board Perzeption ermöglicht. Das System besteht aus drei Komponenten:

- Wahrnehmung
  Baut ein Modell des Geländes vor dem Roboter mit Stereo-Vision.
- Geländemodellierung
  Ausfüllen fehlender / verdeckter Teile des Geländes.
- Planung/Steuerung
  Schritte durch umliegendes Gelände planen und Gelenke bewegen, um diese Schritte zu erreichen.

Abb. 2.1: LittleDog robot ausgestattet mit Tyzx DeepSea Stereo Kamera [Kol09].

Zunächst nutzt die Wahrnehmungskomponente Stereo-Vision und Iterative Closest Point (ICP), ein Punkt-Wolke-Matching-Algorithmus, um ein kohärentes Modell des aktuellen Geländes zu erzeugen und auszurichten, während gleichzeitig der Roboter darin lokalisiert wird. Der ICP-Algorithmus ist eine Methode zur Suche nach einer vollen 6-DOF-Transformation, die zwei sich überlappende Netzmodelle angleicht. Kurz gesagt, funktioniert der ICP-Algorithmus, indem er zunächst eine Reihe von korrespondierenden Punkten auf den beiden Maschen auswählt und dann eine 6-DOF Transformation anwendet, um einige metrische Fehler zu reduzieren und iteriert diesen Prozess bis zur Konvergenz. Die zweite Komponente füllt der Kamera verborgene Geländeteile durch eine Schätzung ihrer Struktur, basierend auf Textursynthesemethoden, aus, um eine robustere Planung zu erzielen (siehe auch [Efr99]). Die dritte Komponente, das Planungssystem, erstellt eine Kostenkarte bezüglich der Vorteilhaftigkeit der verschiedenen Geländepunkte und plant eine Reihe von Low-Cost Schrittpositionen für die nähere, sichtbare Umgebung. Als nächstes werden Trajektorien für das COG des Roboters und für die Füße geplant, um die Schritte zu erreichen und gleichzeitig statische Stabilität des Roboters zu wahren. Die zugrundeliegende Kostenkarte wird direkt aus der Höhenkarte erstellt und beinhaltet zu jedem Punkt lokale Merkmale wie Steigung, Höhenunterschied und Sanftheit, berücksichtigt also auch die umliegenden Punkte.

Im Rahmen desselben Projekts haben Kalakrishnan et. al. [Kal11] das erzeugte Terrainmodell in Kombination mit den gespeicherten Tritt-positionen eines Testlaufs genutzt, um die Schrittpositionswahl der Laufmaschine zu optimieren. Ein menschlicher Experte gab zu allen Fehltritten eine Position an, die besser gewesen wäre. Passend zu den Expertenpositionen wird aus dem Bodenmodell ein Ausschnitt als Vorlage extrapoliert. Bei weiteren Läufen auf anderen Untergründen sucht die Maschine dann nach vergleichbaren Bodenausprägungen und wählt die in der Vorlage enthaltene Schrittposition aus.

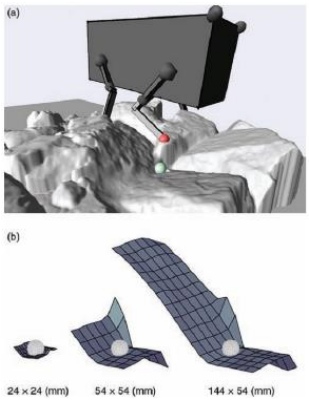
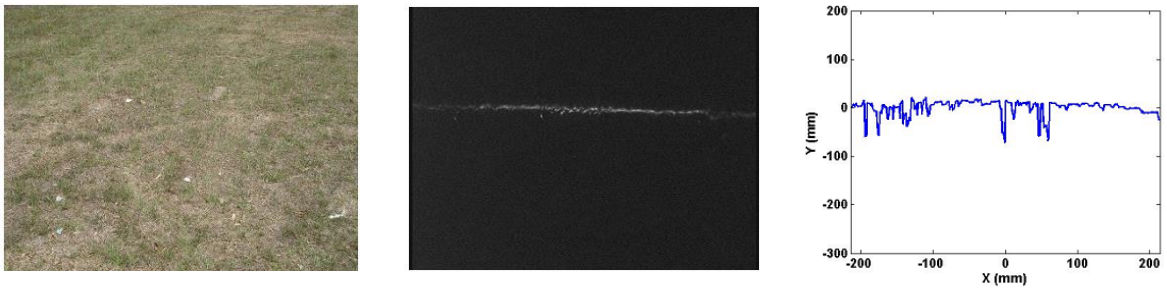


Abb. 2.2: Templates durch Experten. [Kal11]

Anstatt mit 3D-Umweltdaten haben Lu et. al [LuL09] anhand eines Laserstreifenscans und einer Frequenzganganalyse über ein probabilistisches Neuronales Netz gelernt, gängige Bodentypen wie Gras, Sand oder Asphalt zu klassifizieren. Tatsächlich ist der projizierte Laserstreifen charakteristisch genug, um solch grundsätzliche Unterscheidungen vornehmen zu können.



(a) (b) (c)

Abb. 2.3: (a) Grasfläche. (b) Rohdaten des Laserstreifenscans. (c) Nachbearbeitete charakteristische Kennlinie anhand derer eine Frequenzganganalyse zur Bodenklassifizierung erfolgt [LuL09]

2.1.2 Propriozeption

Propriozeption ist eine Eigenwahrnehmung der Körperpose und -bewegung im Raum, letzteres auch als Odometrie bekannt. Umgangssprachlich beurteilt man bei der propriozeptiven Analyse, wie sich ein Schritt „anfühlt“. Typische Kenndaten bei einem laufenden System sind Druck am Endeffektor sowie Drehmomente und Stromaufnahme an den Gelenkantrieben über die Zeit. Die Kraftverteilung variiert je nach Bodenbeschaffenheit charakteristisch, insbesondere die Varianz, Kurtosis und Verzerrung dieser Daten über ein Zeitintervall. Ein Seouler Forscherteam [Kim10] demonstrierte für ein einzelnes Roboterbein, wie die Analyse der Propriozeption zur Bodenklassifizierung genutzt werden kann.

Abb. 2.4: Versuchsaufbau – Propriozeption eines Beins. [Kim10]

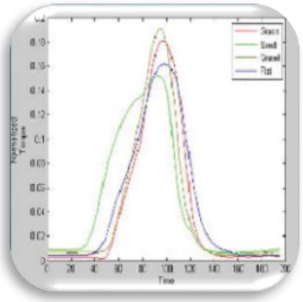


Abb. 2.5: gelernte Kraftaufnahme-Kennlinien für die Böden Gras, Sand, Schotter, Eben. [Kim10]

Die Beinkontroller des Roboters AMEE liefern darüber hinaus noch Informationen über eine eventuelle Kollision oder Abweichung zur beauftragten Fußposition [Ruh11]. Gegebenenfalls wären

Beschleunigungs- und Neigungsdaten eines Gyroskops ebenfalls eine Bereicherung für die propriozeptive Analyse.

2.2 Maschinelles Lernen

Lernfähigkeit und Intelligenz werden meist in unmittelbaren Zusammenhang gesetzt und so ist maschinelles Lernen auch eines der ältesten Themen der KI. Grundsätzlich ist Lernen wohl die Fähigkeit des Systems, aufgrund von Erfahrungen die Leistung zu verbessern oder das Wissen zu mehren. In dieser Arbeit geht es konkret um die Fähigkeit, die in unmittelbarer Umgebung liegende Bodenbeschaffenheit nach Aspekten der sicheren Mobilität zu Beurteilen und diese Fähigkeit kontinuierlich weiter zu entwickeln. Eine weit verbreitete Methode für Klassifizierungsaufgaben ist das statistische Lernen von Modellen anhand von Beispieldaten. Neuronale Netzwerke (NN) sind in diesem Bereich eine etablierte Technik, auch wenn die Erstellung eines guten Netzes eher auf Erfahrung und Probieren basiert, als auf Formalien oder Paradigmen. Bei wahrscheinlichkeitsverteilten Darstellungen bietet sich die Bayes'sche Kombinatoik an. Ein weiteres, in den letzten Jahren an Beliebtheit stets zugenommenes, Lernverfahren ist das Reinforcement Learning, welches auf Markov-Entscheidungsprozesse (eng.-kurz: MDP) basiert. Durch ein verstärkendes Signal (Belohnung / Bestrafung) auf eine Aktion, lernt das System in welchem Zustand welche Aktion die vielversprechendste ist.

Ziel dieser Arbeit ist es, ein fortwährend lernendes Modul zu entwickeln, welches die unmittelbare Umgebung wahrnimmt und eine Auswahl guter Trittpositionen unter Aspekten der Robustheit, Erreichbarkeit, Kollisionsfreiheit und des Bewegungsfortschritts vornimmt, wobei das Reinforcement Learning, welches ein Lernen auf Basis von Erfahrungen modelliert, als lernalgorithmisches Gerüst dient. Es bietet gegenüber anderen Methoden den Vorteil, dass es dem menschlichen Lernen sehr Nahe kommt und eben durch Interaktion mit der Umwelt lernt, indem gute Handlungen belohnt und schlechte bestraft werden. Dadurch ist dies auch eine potentiell höchst anpassungsfähige Methodik die mit veränderlichen Umweltbedingungen zurechtkommt. Unter den verschiedenen Ausprägungen des Reinforcement Learnings ist Sarsa(λ) eines der fortschrittlichsten.

2.3 Reinforcement Learning

Das Grunddesign von Reinforcement Learning (RL) basiert stets darauf, dass der Agent eine Aktion auf/in der Umwelt ausführt und daraufhin eine Belohnung für diese Aktion sowie den Folgezustand mitgeteilt bekommt. Im konkretisierten Anwendungsszenario sind folglich die ersten Modellierungsaufgaben festzulegen, welche Aktionen und Zustände es gibt, mit welcher Datenstruktur sie abgebildet

werden und vor allem, durch welche (Software-/Hardware-) Komponenten die Umwelt repräsentiert wird.

Abb. 2.6: Basiskomponentenmodell des Reinforcement-Learning

2.3.1 Sarsa-λ

Sarsa(λ) ist ein *On-Policy **T**emporal **D**ifference Learning* Verfahren ([Sut12], Kap.7). TD-Verfahren sind eine leistungsstake, in vielen simulierten Domänen bewährte Methodik des maschinellen Lernens, beruhend auf einer etablierten theoretischen Grundlage. Die zentrale Idee beim TD-Lernen betrifft die Strategieverbesserung (*Policy Iteration*), mit der Aktualisierung der **Schätzwerte Q(s,a)** für die zu erwartende Gesamtbelohnung im **Zustand s** für die **Aktion a**: Addiere die Differenz aus tatsächlicher und erwarteter Gesamtbelohnung (skaliert durch **Lernrate α**).

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot [R_t - Q_t(s,a)] \qquad \textbf{Glg. 2.1}$$

Während eine Lernepisode läuft, ist die letztliche **Gesamtbelohnung R_t** noch nicht bekannt, sondern wird durch die Belohnung der aktuellen Aktion und der Schätzung für den **Folgezustand s^+** und der **nächsten Aktion a^+** abgeschätzt:

$$R_t = r_{t+1} + \gamma \cdot Q_t(s^+, a^+) \qquad \textbf{Glg. 2.2}$$

Mit dem **Bewertungsfehler** $\boldsymbol{\delta_t} \coloneqq r_{t+1} + \gamma \cdot Q_t(s^+, a^+) - Q_t(s,a)$ ergibt sich die Aktualisierungsgleichung dann zu:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot \delta_t \qquad \textbf{Glg. 2.3}$$

Sarsa(λ) berücksichtigt zusätzlich frühere Entscheidungen mittels ***Eligibility* (e-Funktion)**, einer Belohnungsberechtigung früherer (s,a)-Tupel der Episode an der aktuellen Belohnung. Ein Update erweitert sich dann zu:

$$\boldsymbol{Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot \delta_t \cdot e_t(s,a) \quad | \text{ü}ber\ alle\ (s,a)\ der\ Episode} \qquad \textbf{Glg. 2.4}$$

Q_t ist folglich eine rekursive Abbildung $Q \times E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, beziehungsweise ein Folgeglied der zu erlernenden optimalen Schätzfunktion, die sich letztlich aus der Reihe $Q^\pi = \lim_{t \to \infty}(Q_t)$ ergibt.

Für die e-Funktion gibt es verschiedene Ausprägungen. Eine mit „ersetzenden Spuren“ ist:

$$e_t(s,a) = \begin{cases} 1, & f\ddot{u}r\ aktuelles\ (s,a) \\ \gamma \cdot \lambda \cdot e_{t-1}(s,a), & sonst \end{cases} \qquad \textbf{Glg. 2.5}$$

Vorausgesetzt, Zustands- und Aktionsraum sind diskret, lassen sich Q- und e-Funktion in der praktischen Umsetzung als Tabellen beziehungsweise $|\mathcal{S}| \times |\mathcal{A}|$ Matrizen $\boldsymbol{W}$ und $\boldsymbol{E}$ über alle Zustände und Aktionen realisieren, die bei jedem Iterationsschritt aktualisiert werden. Die für den Algorithmus wesentlichen Gleichungen lauten dann:

Bewertungsmatrix $\boldsymbol{W} = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \cdots & \cdots & \cdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix}$ mit $Q(s_i, a_j) = w_{ij}$

Lohnberechtigungsmatrix $\boldsymbol{E} = \begin{pmatrix} e_{11} & \cdots & e_{1n} \\ \cdots & \cdots & \cdots \\ e_{m1} & \cdots & e_{mn} \end{pmatrix}$ mit $e(s_i, a_j) = e_{ij}$

Aktualisierung der Lohnberechtigungsmatrix: 1) $\boldsymbol{E}_t = \gamma\lambda \cdot \boldsymbol{E}_{t-1}$
2) $e_{ij} = 1\ f\ddot{u}r\ (s = s_i, a = a_j)$

Aktualisierung der Bewertungsmatrix $\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \alpha\delta \cdot \boldsymbol{E}_t$

Initialisiere Bewertungsmatrix **W** (Q-Funktion) beliebig und Lohnberechtigungsmatrix **E** = **0.** Setze Lernrate α, Diskontierungsfaktor γ, Spurzerfallsfaktor λ.
Wiederhole (für jede Episode):
  Initialisiere Zustand s, Aktion a
  Wiederhole (für jeden Schritt der Episode):
    Führe Aktion a aus, beobachte Lohn r und Folgezustand s^+
    Wähle nächste Aktion a^+ entsprechend der Strategie (z.B. ε-greedy)
    $\delta \leftarrow r + \gamma Q(s^+, a^+) - Q(s,a) = r + \gamma w_{s^+a^+} - w_{sa}$
    $\boldsymbol{E} \leftarrow \gamma\lambda\boldsymbol{E}$
    $e_{sa} \leftarrow 1;\ e_{s\bar{a}} \leftarrow 0$ (ersetzende Spuren; $\bar{a}$ = nicht a)
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\delta\boldsymbol{E}$
    $s \leftarrow s^+; a \leftarrow a^+$
  bis s Endzustand ist

Abb. 2.7: Sarsa(λ) Algorithmus aus dem Sutton [Sut98]. *Projekt AMEE* Interpretation

2.3.2 RL Parameter

Die formale Bedeutung und Auswirkung der Lernparameter kann im Detail der Fachliteratur entnommen werden. Die folgende Zusammenfassung stellt eine informelle Interpretation dar. Alle Parameter werden mit einem Wert zwischen Null und Eins belegt.

- **Lernrate α**
  Bei der Aktualisierung der Belohnungsschätzfunktion (**Glg. 2.1**) gibt dieser Faktor an, zu welchem Anteil die aktuelle Erfahrung in die künftige Schätzung einfließt. Eine niedrige Lernrate bevorzugt das bereits erlernte Wissen, ein hohes Alpha die neuen Erfahrungen. Typischerweise liegt der Faktor im Bereich 0,5 bis 0,7.

- **Diskontierungsfaktor γ**
  Im Wesentlichen sorgt dieser Faktor für nicht episodische Lernaufgaben dafür, dass die Schätzfunktion konvergiert. Darüber hinaus kann er als Voraussicht des Agenten interpretiert werden, da Gamma die zukünftige Belohnung für die aktuelle Schätzung gewichtet (**Glg. 2.2**). Ein üblicher Wert liegt um die 0,9.

- **Spurzerfallsfaktor λ**
  Bei der Belohnungsberechtigung (**Glg. 2.5**, *eligibility*) zuvor besuchter Zustände an der aktuellen Belohnung regelt der Parameter Lambda, wie weit die Berechtigung zurückreicht und in die Vergangenheit hin abgeschwächt wird. Es kann als Rückwärtssicht betrachtet werden und liegt oft im Bereich 0,8 bis 0,95.

- **Explorationsrate ε**
  Die Explorationsrate gehört zur häufig genutzten ε-Greedy Erkundungsstrategie (**Abb. 2.7**; [Sut12]) und gibt den Anteil der Aktionen an, die nicht der jeweils erlernten besten Aktion entsprechen, sondern zufällig gewählt werden. Der Agent soll hier hin und wieder auch mal etwas *Neues* ausprobieren.

2.3.3 Lineare Gradientenabstiegsmethode

Beim Standard Sarsa(λ) wird die Wert- beziehungsweise Zustands-Wert-Funktion tabellarisch dargestellt und die einzelnen Tabellenwerte sukzessive hin zur optimalen Bewertungsfunktion angepasst. Die tabellarische Repräsentation ist nur anwendbar, wenn man es mit endlich vielen, abzählbaren Zuständen und Aktionen zu tun hat. Für sehr große Zustandsräume und insbesondere in einem kontinuierlichen Zustandsraum ist dies nicht möglich. Hier muss eine Näherungsfunktion gefunden werden, die die Schätzunktion approximiert. Es werden verschiedene Funktionsnäherungs-verfahren für Reinforcement Learning vorgeschlagen, wie einfache Diskretisierung, radiale Basis-funktionen oder Neuronale Netze, bei denen es stets gilt ein ausgewogenes Verhältnis zwischen Genauigkeit, Rechenaufwand und Handhabung zu finden.

Eine naheliegende Herangehensweise für das Problem im kontinuierlichen Werten, ist eine Generalisierung durch Partitionierung des Zustandsraums. Statt einen konkreten Zustand zu bewerten, werden stattdessen ein oder mehrere Bereiche, in denen ein Zustand einzuordnen ist, bewertet. Mehr zur Partitionierung in Bereiche, die man hier als Merkmale eines Zustands bezeichnet, folgt in **2.3.4** und **4.2**. Ein etabliertes Approximationsverfahren für Sarsa(λ) ist die Gradientenabstiegsmethode, die auch hier weiter verfolgt wird. Die grundlegende Neuheit dabei ist, dass die Wertfunktion V_t beziehungsweise Q_t durch eine lineare Funktion dargestellt wird, bestehend aus den Parametern $\vec{w}$ und einem zustands- und aktionsspezifischen Merkmalsvektor $\vec{f}_{sa}$.

2.3.3.1 Approximation durch lineare Basisfunktionen

Ausgangsidee für die lineare Gradientenabstiegsmethode ist, dass sich die gesuchte Funktion q annähernd als Linearkombination mehrerer Basisfunktionen darstellen lässt: $q = \sum_{i=0}^{n} w_i \cdot f_i(\vec{x})$. Hierbei bezeichnet $\vec{x}$ den Eingangsvektor und w_i das Gewicht der i'ten Basisfunktion f_i. Ist q beispielsweise ein Polynom zweiten Grades, also $q = w_0 + w_1 \cdot x + w_2 \cdot x^2$, besteht der Eingangs-vektor aus einem Element x und die Basisfunktionen sind $f_0(x) = 1$, $f_1(x) = x$, $f_2(x) = x^2$.

2.3.3.2 Iterative Parameterbestimmung

Das Funktionsmodell in Form der Basisfunktionen wird der Aufgabe entsprechend als gegeben vorausgesetzt. Ziel der Funktionsapproximation ist es, die Gewichte $\vec{w}$ anhand einer Reihe von Test- oder Messdaten so anzupassen, dass die geschätzte Funktion $\widehat{\boldsymbol{q}}$ möglichst nah an der optimalen (realen) Funktion $\boldsymbol{q}$ liegt. Eine Möglichkeit der Anpassung ist die **Gradientenabstiegsmethode**, bei der die Abweichung δ des Ist-Ergebnis $\widehat{\boldsymbol{q}}$ vom Soll-Ergebnis $\boldsymbol{q}$ durch Korrektur der Gewichte in die Richtung

der größten Fehlerreduzierung minimiert wird. Die Richtung wird dabei durch Ableiten des quadratischen Fehlers[3] $E = \frac{1}{2}(\hat{q} - q)^2 = \frac{1}{2}\delta^2$ ermittelt.

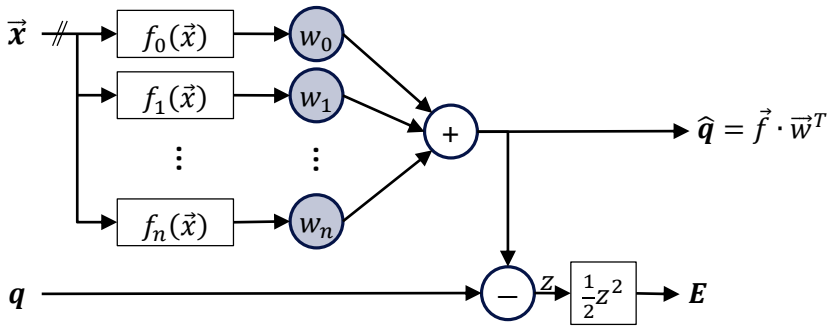


Abb. 2.8: funktionale Graphdarstellung einer Linearkombination von Basisfunktionen mit quadratischem Fehler

Der Gradient $\nabla E = \left(\frac{\partial E}{\partial x_0} \; \frac{\partial E}{\partial x_1} \; \cdots \; \frac{\partial E}{\partial x_n}\right)^T$ zeigt die Richtung des steilsten Fehleraufstiegs. Dabei ist das Ergebnis der Ableitung[4] $\nabla E = \delta \cdot \vec{f}$. Die Korrektur der Gewichte erfolgt iterativ mit jedem Eingangsvektor und zugehörigem Soll-Wert durch Subtraktion des Fehlergradienten, zusätzlich skaliert durch eine Schrittweite α:

$$\vec{w}_{t+1} = \vec{w}_t - \alpha \cdot \nabla E = \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix}_t - \alpha\delta_t \cdot \begin{pmatrix} f_0(\vec{x}_t) \\ \vdots \\ f_n(\vec{x}_t) \end{pmatrix}$$

Eine Herleitung und Diskussion zum Verfahren sowie Alternativen der linearen Regression sind in der gängigen Lehrliteratur zu finden (z.B. [Bis06]). Der hier vorgestellte kurze Überblick dient nur dem Verständnis der Anwendung im Zusammenhang mit Sarsa(λ).

2.3.3.3 Linear Gradient-Descent-SARSA(λ)

Die lineare Gradientenabstiegsmethode lässt sich auch auf die Bewertungsfunktion im Reinforcement Learning anwenden. Voraussetzung ist, dass sich jeder Zustand auf einen Merkmalsvektor abbilden lässt. Im Zusammenhang mit der Idee, den kontinuierlichen Zustandsraum zu partitionieren, heißt dass beispielsweise, dass ein Zustand s das Merkmal f_i hat, wenn s in der Partition P_i liegt. Allgemein kann die Merkmalszuordnung aber auch beliebige andere Formen annehmen, zum Beispiel das Verhältnis

[3] Quadrieren ist eine einfache Methode, absolute Werte zu erhalten. Der Faktor ½ vereinfacht die Ableitungsfunktion.

[4] Anwendung der Kettenregel auf jede partielle Ableitung nach x_i

zweier Eingangsparameter angeben, wie $f_i(\vec{x}) = \frac{x_i}{x_{i-1}}$. Bezogen auf die zuvor beschriebenen linearen Basisfunktionen, sind solche hier quasi Merkmalsidentifikatoren f_{sa}, die einem Zustands-Aktions-Paar eine bestimmte Eigenschaft zuordnen, gewichtet durch Parameter $\vec{w}$. Im von Sutton beschriebenen ***Linear Gradient-Descent-SARSA(λ)***-Algorithmus ergibt sich die Bewertung eines Zustand-Aktions-Paars dann zu

$$\boldsymbol{Q_t(s,a) = \vec{w}_t^T \cdot \vec{f}_{sa} = \sum_{i=1}^{n} w_{t,i} \cdot f_{sa,i}} \qquad \textbf{Glg. 2.6}$$

In Sarsa(λ) wird die Abweichung einer Schätzung durch den Parameter δ errechnet (vgl. **Glg. 2.3**) und der Gradient der quadratischen Fehlerfunktion ist auch hier wieder $\nabla E = \delta \cdot \vec{f}$. Nach der linearen Gradientenabstiegsmethode wird der Fehler durch sukzessive Anpassung des Parametervektors $\vec{w}$ reduziert und Q nähert sich der optimalen Bewertungsfunktion:

$$\boldsymbol{\vec{w}_{t+1} = \vec{w}_t + \alpha \delta_t \cdot \vec{e}_t} \qquad \textbf{Glg. 2.7}$$

Der Merkmalsvektor $\vec{f}$ ist hier durch einen Eligibility-Vektor $\vec{e}$ ersetzt. Dies ändert die Eigenschaft der Funktionsapproximation an sich nicht, sondern erweitert lediglich den Informationsgehalt, durch Angabe einer Lohnberechtigung oder -verantwortung eines Merkmals. Die Aktualisierung des e-Werts je Merkmal erfolgt vergleichbar zur tabellarischen Version:

$$e_i = \begin{cases} 1, & \text{für } f_{sa,i} \neq 0 \\ \gamma \cdot \lambda \cdot e_i, & sonst \end{cases} \qquad \textbf{Glg. 2.8}$$

Für eine tiefergehende Herleitung der hier vorgestellten Methode sei auf Sutton [Sut12] und dort referenzierte Literatur hingewiesen. Man beachte des Weiteren, dass hier von diskreten oder zumindest diskretisierten Aktionen ausgegangen wird. Theoretisch sind zwar auch kontinuierliche Aktionsräume denkbar, auf die ebenfalls ein Partitionierungsverfahren angewendet wird. Dies wird aber hier ebenso wie in der einschlägigen Literatur nicht weiter verfolgt.

2.3.3.4 Beispiel

Angenommen ein Zustand s kann eines oder mehrere der Merkmale f_1, f_2, f_3 haben, dann bezeichnet $\vec{f}_s^T = (f_1, f_2, f_3)$ den Merkmalsvektor[5] mit $f_i = 1$, wenn s das Merkmal f_i hat und $f_i = 0$, wenn nicht (näheres dazu in **2.3.4**). Für Sarsa(λ), wo Zustands-Aktions-Paare bewertet werden, ist $\vec{f}_{sa}$ eine

[5] f für *Feature Set*, wie es in der engl. Literatur heißt

Erweiterung von $\vec{f_s}$, bei dem die Merkmale um die Anzahl an Aktionen vervielfacht werden. Hierfür gilt $f_{ij} = 1$, wenn s das Merkmal f_i hat und a_j die aktuelle Aktion ist. Ausgehend davon, dass es zwei mögliche Aktionen a_1, a_2 gibt, so ist beispielsweise für die Aktion a_2 und für einen Zustand s_t mit den Merkmalen f_1, f_3:

$$\vec{f}_{sa}^{T} = (\underbrace{0,0,0}_{a_1}, \underbrace{1,0,1}_{a_2}) \;\; | \;\; Merkmale(s_t) = \{1,3\} \; und \; a = a_2$$

Der Parametervektor $\vec{w}$ hat bei den hier drei Merkmalen und zwei Aktionen ebenfalls insgesamt $3 \cdot 2 = 6$ Elemente. Je Aktion also drei Gewichtungswerte, die für die Bewertung der Merkmalsausprägung der Zustände s und für eine spezifische Aktion stehen:

$$\vec{w}^{T} = (\underbrace{w_{11}, w_{21}, w_{31}}_{a1}, \underbrace{w_{12}, w_{22}, w_{33}}_{a2})$$

Die Bewertung eines Zustands-Aktions-Paares ergibt sich dann gemäß **Glg. 2.6** aus der Summe der gewichteten Merkmale, z.B. zu:

$$Q(s_t, a_2) = \vec{w}_t^{T} \cdot \vec{f}_{sa} := (1.1, -2.0, 3.2, -4.7, 0.2, 3.5) * (0,0,0,1,0,1)^{T} = -4.7 + 3.5 = -1.2$$

2.3.3.5 Tabellarische Interpretation

Für das *Projekt AMEE* wurde eine leicht abgeänderten Organisation der Parameter (Gewichte) und Merkmale entwickelt, die eine andere Perspektive auf die lineare Approximationsfunktion ergibt und eine starke Ähnlichkeit zur tabellarischen Funktion, wie sie eingangs in **2.3.1** vorgestellt wurde, offenbart. So kann man je Aktion einen Spaltenvektor für die Gewichte w vorsehen und erhält die Parameter der Bewertungsfunktion als Matrix $\boldsymbol{W}$. Bezeichne nun $\vec{a} = (a_1, a_2)^{T}$ den Aktionsvektor mit $a_i = 1$ für die Aktion $a =$ *i'te Aktion aus der Menge aller Aktionen*, dann ergibt sich die Bewertung äquivalent zu **Glg. 2.6** durch:

$$Q(s,a) = W \cdot \vec{a} \cdot \vec{f}_s^{T}$$

mit beispielsweise

$$\boldsymbol{W} := \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cc} a_1 & a_2 \end{array} \\ \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \end{array} \qquad \vec{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \qquad \vec{f}_s^{T} = (f_1, f_2, f_3)$$

Angewendet auf obiges Zahlenbeispiel:

$$Q(s_1, a_2) = \begin{pmatrix} 1.1 & -4.7 \\ -2.0 & 0.2 \\ 3.2 & 3.5 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot (1 \quad 0 \quad 1) = \begin{pmatrix} -4.7 \\ 0.2 \\ 3.5 \end{pmatrix} \cdot (1 \quad 0 \quad 1) = -4.7 + 3.5 = -1.2$$

Die Lohnberechtigung wird in dieser Variante wie **W** ebenfalls als gleichgroße Matrix **E** dargestellt. Hat der Startzustand beispielsweise die Merkmale f_1, f_3, wie im vorigen Beispiel, dann wird für die erste Aktion, sei es a_2, im entsprechenden Zeilenvektor das erste und dritte Element auf Eins gesetzt. Habe der Folgezustand, die Merkmale f_1, f_2 und die nächste Aktion sei wieder a_2 erfolgt die Aktualisierung der Lohnberechtigung äquivalent zu **Glg. 2.8**:

$$\boldsymbol{E_1} = \begin{matrix} \\ f_1 \\ f_2 \\ f_3 \end{matrix} \begin{matrix} a_1 & a_2 \end{matrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \xrightarrow{\gamma\lambda := 0.81} \boldsymbol{E_{2\_tmp}} = 0.81 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \xrightarrow{f_{s2}=(1,0,1)^T,\, a^+=a_2} \boldsymbol{E_2} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0.81 \end{pmatrix}$$

Die Aktualisierung der Parameter (vgl. **Glg. 2.7**) ist dann eine Matrizenaddition. Angenommen, der Bewertungsfehler ist $\delta = 0.8$ (vgl. **Glg. 2.3**) lautet die Aktualisierungsgleichung bei einem Lernfaktor $\alpha = 50\%$ beispielsweise:

$$\boldsymbol{W}_2 = \boldsymbol{W}_1 + \alpha\delta\boldsymbol{E}_2 = \begin{pmatrix} 1.1 & -4.7 \\ -2.0 & 0.2 \\ 3.2 & 3.5 \end{pmatrix} + 0.5 \cdot 0.8 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0.81 \end{pmatrix} = \begin{pmatrix} 1.1 & -4.3 \\ -2.0 & 0.6 \\ 3.2 & 3.82 \end{pmatrix}$$

2.3.3.6 Der Algorithmus

Abb. 2.9 zeigt eine auf die tabellarische Interpretation angepasste Variante von Suttons Algorithmus-Beispiel für den Linearen Gradientabstiegs Sarsa(λ).

Seien **W** und **E** Matrizen mit so vielen Elementen, wie möglicher Merkmale, **W** die Matrix der Gewichte der Q-Funktion, **E** die der Lohnberechtigungsfaktoren
Sei $\vec{f}_s$ ein Vektor von Merkmalen eines Zustands s und initial **0**
Sei $\vec{a}$ ein Indizierungsvektor der möglichen Aktionen $\mathcal{A}$ und initial **0**
Initialisiere **W** nach Bedarf der Aufgabe, z.B. **W** = **0**
Wiederhole (für jede Episode):
  $\boldsymbol{E} = \mathbf{0}$
  $S, A \leftarrow$ Startzustand und –aktion der Episode
  Wiederhole (für jeden Schritt der Episode):
    $\overrightarrow{f_s} \leftarrow f_{s,i} = 1$ für Merkmale des Zustands S, 0 sonst
    $\overrightarrow{a} \leftarrow a_j = 1$ für $A =$ j'te Aktion aus $\mathcal{A}$, 0 sonst
    Führe Aktion A aus, beobachte Lohn R und Folgezustand S⁺
    (S⁺ und A⁺ bezeichnen den Folgezustand und die nächste Aktion)
    $\vec{f}_{s^+} \leftarrow f_{s^+,i} = 1$ für Merkmale des Zustands S⁺, 0 sonst
    Wähle nächste Aktion a⁺ entsprechend der Strategie (z.B. ε-greedy):
    Mit Wahrscheinlichkeit $1 - \mathcal{E}$:
      Für alle $a \in \mathcal{A}(S^+)$:
        $\overrightarrow{a} \leftarrow a_j = 1$ für $a =$ j'te Aktion aus $\mathcal{A}$, 0 sonst
        $Q(S^+, a) \leftarrow \boldsymbol{W}\vec{a}\vec{f}_{s^+}^{T}$
      $A^+ \leftarrow \arg \max_{a \in \mathcal{A}(S^+)} Q(S^+, a)$
    else
      $A^+ \leftarrow$ zufällige Aktion $\in \mathcal{A}(S^+)$
      $\overrightarrow{a} \leftarrow a_j = 1$ für $A^+ =$ j'te Aktion aus $\mathcal{A}$, 0 sonst
      $Q(S^+, A^+) \leftarrow \boldsymbol{W}\vec{a}\vec{f}_{s^+}^{T}$
    $\delta \leftarrow R + \gamma Q(S^+, A^+) - Q(S, A)$
    $\boldsymbol{E} \leftarrow \gamma\lambda\boldsymbol{E}$
    $e_{ij} \leftarrow 1$ für $f_{s,i} = 1$ und $a_j = 1$, (ersetzende Spuren)
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\delta\boldsymbol{E}$
    S←S⁺; A←A⁺
  bis S⁺ Endzustand

Abb. 2.9: Linearer Gradientenabstiegs-Sarsa(λ) mit binären Merkmalen und ε-Greedy Strategie (vgl. [Sut12], Kap. 9.3). *Projekt AMEE* Interpretation.

2.3.3.7 Fazit

Die tabellarische Interpretation veranschaulicht die strukturelle Ähnlichkeit des *Linear Gradient-Descent-SARSA(λ)* mit dem ursprünglichen Sarsa(λ), sowie den Zusammenhang der Parameter und einem Zustands-Aktionspaar. Sie verdeutlicht, dass der Sarsa(λ)-Algorithmus sowohl für die tabellarische Ausprägung für diskrete Zustände, als auch für die lineare Variante für kontinuierliche Zustände weitgehend identisch ist, da mit der Matrizendarstellung die wesentlichen Aktualisierungsgleichungen dieselben sind. Lediglich die Berechnung der Zustands-Aktions-Bewertung Q(s,a) variiert leicht. Beim diskreten Sarsa(λ) gibt ein einzelnes Element der Tabelle (Matrix) den Wert wieder, beim linearen Sarsa(λ) ist es die Summe mehrerer Elemente eines Spaltenvektors aus der Matrix, eben die gewichteten Zustandsmerkmale. Zumindest im *Projekt AMEE* hat die tabellarische Interpretation das Verständnis für den Algorithmus verbessert und ermöglichte die Entwicklung eines Frameworks, dass auf beide Algorithmen anwendbar ist (siehe **3.3**).

2.3.4 Merkmalskodierung

Eine wesentliche Eigenschaft der Funktionsapproximation im RL-Kontext ist, dass w und f mehrere Zustände zusammenfassen. Ein zweidimensionaler (aber kontinuierlicher) Zustandsraum kann beispielsweise in sich überschneidende Kreise oder Ellipsen eingeteilt werden, was Sutton als *grobe Kodierung* bezeichnet (vgl. **Abb. 2.10**). Ein Zustand besitzt das Merkmal f_i, wenn er in dessen Bereich liegt und hat dann den Wert 1, ansonsten ist $f_i = 0$. Neben dieser *binären Kodierung* ist aber auch eine radiale Basisfunktion denkbar, bei der die Merkmalswerte im Intervall $[0,1]$ liegen (vgl. [Sut98], Kap. 8.3.3). Etwas recheneffizienter ist die Kachelkodierung (Tiling), eine diskrete Partitionierung des Zustandsraums.

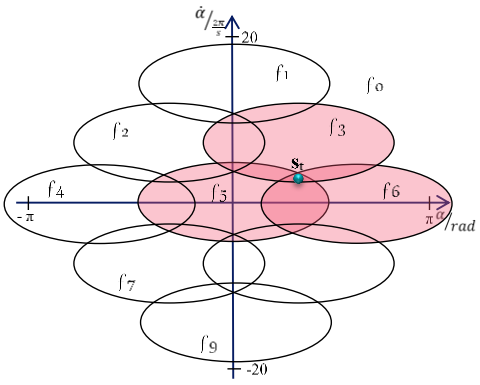


Abb. 2.10: Bsp.; Grobe Kodierung der Merkmale Winkel α und Winkelgeschwindigkeit $\dot{\alpha}$. Der Zustand S_t besitzt die Merkmale f3, f5 und f6.

2.3.4.1 Tile Coding

In erster Linie unterteilt die Kachelkodierung den Wertebereich einer Dimension in eine endliche Anzahl disjunkter Intervalle I_{d1} bis I_{dN} (d dient hier als Bezeichner der Dimension, bspw. Winkel α) und sorgt für eine Diskretisierung des Wertebereichs, indem nur noch die Zugehörigkeit eines konkreten Wertes zu einem Intervall I_{dn} betrachtet wird. Eine solche Partitionierung wird als Kachelung bezeichnet (im Folgenden wird der engl. Begriff ***Tiling*** verwendet). Das Tile-Coding nutzt meist mehrere, sich überlappende Tilings, um eine breitere Generalisierung dieses Approximationsverfahrens zu erzielen. Liegt ein Wert α im Intervall $I_{\alpha j}$, so sagt man α hat das Merkmal f_j. Bei mehreren Tilings besitzt α in der Regel so viele Merkmale wie die Anzahl an Tilings (vgl. **Abb. 2.11**) und man spricht von einem ***Feature Set*** $\mathcal{F}_\alpha$, das die Indizes der Merkmale von α beinhaltet. Der Bewertungswert V (oder Q) zu α, ergibt sich dann aus der Summe der zu den Merkmalen von α gehörenden Gewichte $w_i \mid i \in \mathcal{F}_\alpha$. Durch das Anpassen dieser zum Testwert α gehörenden Gewichte um den Fehlerbetrag δ, erfolgt die Annäherung zur optimalen Schätzfunktion.

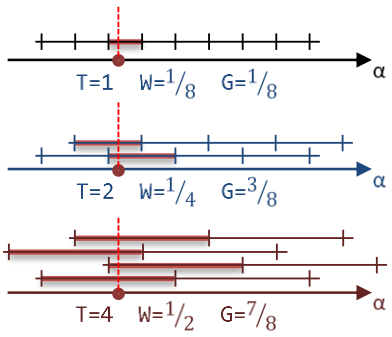


Abb. 2.11: T=1, 2, und 4 Tiling Anordnung; W = Intervallweite; G = Generalisierungsbreite (vgl. [She05]). Je Anordnung sind die Intervalle, in denen ein konkreter Wert α liegt, hervorgehoben. Alle Anordnungen haben hier eine Auflösung von 1/8.

Beispiel

Die zweite Tiling Anordnung in **Abb. 2.11** besteht aus $2 \cdot 4 = 8$ Intervallen, beziehungsweise Merkmalen und besitzt entsprechend viele Gewichte w_i. Der markierte Wert α liegt im ersten und vierten Intervall ($\mathcal{F}_\alpha = \{1, 4\}$). Angenommen die aktuelle Gewichtung betrage $\vec{w} = (5, 4, 3, 8, 9, 0, 0, 5)^T$ und der Fehler δ sei -2, dann ist $V(\alpha) = w_1 + w_4 = 4 + 9 = 13$ und die Gewichte w_1 und w_4 werden auf 2 beziehungsweise 7 korrigiert.

2.3.4.2 Binäres Tiling

Bei der oben beschriebenen Variante, dass ein Zustand s oder Wert α ein Merkmal besitzt oder eben nicht, nennt Sutton *Binäres Tiling*. Alternativ könnte aber auch das Merkmal selbst ein Gewicht besitzen, das beispielsweise angibt, wie nah der aktuelle Zustand am Kachelzentrum ist (vgl. Radiale Basisfunktion in [Sut98], Kap. 8.3.3), beziehungsweise – angelehnt an Fuzzylogik – in welchem Maße das Merkmal zutrifft. Es sind derzeit jedoch keine Untersuchungen bekannt, ob sich entsprechende Alternativen positiv auf die Approximation auswirken und sich dadurch der erhöhte Rechenaufwand lohnen würde.

2.3.4.3 Tilings-Anordnung

Die Tilings-Anordnung – Anzahl der Tilings, Breite der Intervalle – hat direkten Einfluss auf den Grad der Generalisierung und ist eine Abwägung zwischen Rechenressourcen und Approximationsgenauigkeit. Je höher die Auflösung der Tilings-Anordnung ist, desto besser ist offensichtlich die Approximationsgenauigkeit, allerdings auf Kosten der Rechenressourcen. Wie **Abb. 2.11** zeigt, kann dieselbe Auflösung durch verschiedene Anordnungen erzielt werden. Sherstov et. al. [She05] haben gezeigt, dass bei gleicher Auflösung eine hohe Tiling-Anzahl die Approximation während der ersten Lernepisoden beschleunigt, da aufgrund der größeren Generalisierungsbreite die erzielten Erfahrungen auf mehr und insbesondere auch seltener besuchte Zustände verteilt werden. In einer späteren Lernphase jedoch, wenn ein gewisses Maß an Wissen erlangt wurde, kehrt sich der Vorteil ins Gegenteil um und eine geringe Anzahl Tilings mit folglich geringerer Generalisierungsbreite ist vorzuziehen. Unabhängig von der Problemstellung gibt es nicht die eine beste Anordnung. Stattdessen stellen Sherstov et. al. ein RL-Verfahren mit adaptiver Tilings-Anordnung vor.

2.3.4.4 Mehrdimensionales Tiling

Wendet man Tile-Coding auf zwei Dimensionen an, wie in **Abb. 2.12** dargestellt, wird die Verwendung des Begriffs *Tile* deutlich. Jedoch ist das Verfahren nicht auf ein oder zwei Dimensionen beschränkt. Die Organisation der Tilings bestimmt die Anzahl der Merkmale $\vec{f}$ und ihrer Gewichte $\vec{w}$, welche sich aus der Summe der Tiles je Tiling für jede Dimension ergibt. Für das Beispiel aus **Abb. 2.12** wären das zwei Tilings mit je drei Tiles für den Winkel α und $2 \cdot 4$ Tiles für die Winkelgeschwindigkeit $\dot{\alpha}$ und somit $\dim(\vec{f}) = \dim(\vec{w}) = 14$. Die Wahl der Merkmale fügt dem RL-System das aufgabenspezifische Domänenwissen hinzu und die Anordnung der Tilings bestimmt die Approximations-genauigkeit und Ressourcenkomplexität. Das bisher beschriebene Standardvorgehen beim Tile-Coding ist eine einfache Linearkombination der Merkmale: $V(s) = \vec{f}_s \cdot \vec{w}^T$. Diese bildet aber keine Abhängigkeiten zwischen ihnen ab, weswegen gegebenenfalls zusätzliche Merkmale für die Kombination von Eigenschaften hinzugenommen werden. So könnte es beispielweise bei einem Roboterarm mit zwei Freiheitsgraden von Bedeutung sein, ob der erste Winkel größer oder kleiner als der zweite Winkel ist. Eine genauere Betrachtung von Merkmalskombinationen wird in **4.2** vorgenommen.

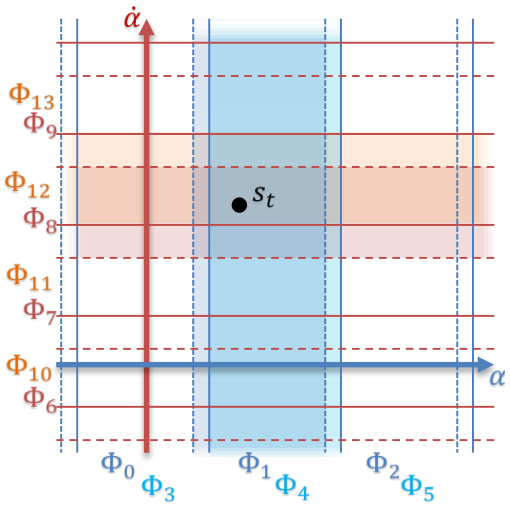


Abb. 2.12: : Beispiel - Position und Tempo eines Gelenks. Winkel α mit einem $2 \cdot 3$ Tiling und Winkelgeschwindigkeit $\dot{\alpha}$ mit einem $2 \cdot 4$ Tiling. Der Zustand s_t zum Zeitpunkt t besitzt die Merkmale $\Phi_1, \Phi_4, \Phi_8, \Phi_{12}$.

3 Konzeptionelle Voruntersuchungen

Aufgrund der Komplexität der Lernaufgabe beschränkt sich diese Arbeit im Wesentlichen auf den Lernalgorithmus. Nichts desto trotz ist es für das Projekt AMEE notwendig und für das Verständnis der Lernaufgabe und des Lösungsansatzes hilfreich, einige Rahmenbedingungen näher zu beleuchten. Dieser Abschnitt betrifft nicht unbedingt den Kern der Arbeit, sondern beschreibt Ausflüge an die Randbereiche zur Einordnung in das Gesamtkonzept. Er beschreibt Grundlagenarbeiten und stellt eine experimentelle Basis für eine künftige Integration in die AMEE-Anwendung dar. Der erste Unterabschnitt befasst sich dabei mit einem Softwarekonzept für verteilte Roboteranwendungen. Experimente zur Sensorik und Umgebungserfassung sowie -analyse finden in **3.2 Robot Vision** statt. Ein neues Basisframework für Reinforcement Learning, auf dem die Experimente zum Lernverfahren beruhen, stellt **3.3 RL Framework** vor.

3.1 LittleAMEE

Die Arbeit an LittleAMEE bietet ein Softwaretechnisches Konzept für eine verteilte Roboteranwendung mit Echtzeitanteilen in der Beinsteuerung und ereignisgesteuerten Modulen der höheren Logik im Roboterverhalten. Es wurde experimentell das Robotics Development Studio R4 von Microsoft® (MS-RDS) auf die Verwendbarkeit in einem geländefähigen, vierbeinigen Roboter geprüft. Wesentliche Motivation dabei war es, die Konzepte der Vorarbeiten aus [Bet10] und [Ruh11] zusammenzuführen. Erstere Arbeit skizziert ein Eventsystem mit der von Microsoft® angebotenem *Concurrency and Coordination Runtime (CCR)* und den *Decentralized Software Services (DSS)* Framework aus dem Robotics Studio in einem Laufsystem. In der zweiten Arbeit wurde gezeigt, dass eine Trennung von zeitkritischen und zeitunkritischen Tasks mit „intelligenten“, externen Kontrollern realisiert werden kann (vgl. **Abb. 3.1**).

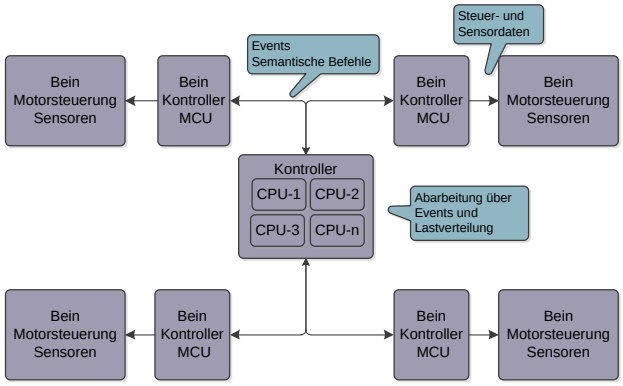


Abb. 3.1: Konzept einer verteilten Kontrollerarchitektur – Trennung von zeitkritischen und nicht zeitkritischen Aufgaben auf Soft- und Hardware Ebenen [Ruh13]

Für erste Tests des Kontrollerkonzepts ([Bet10], [Ruh12]) wurde mit dem Lego® Mindstorm® V2 Bausystem eine verkleinerte Version des AMEE Quadruped Rough Terrain Robot mit einer vereinfachten Konstruktion eines Laufsystems erstellt und LittleAMEE getauft. Zwei NXT V2 mit der Lego® Firmware dienten als externe Kontroller. Die Grundzüge des Deliberative- / Reactive-Layer Konzepts [Reb08] wurden implementiert und es wurde gezeigt, dass die geplante Softwarearchitektur es ermöglicht, die ereignisbasierte, höhere Logik mit dem reaktiven, zeitkritischen Unterbau sinnvoll zu einem Laufverhalten zu verbinden.

In diesem Kapitel wird das Proof of Concept für die Software Architektur dargestellt. Dafür erfolgt in **3.1.1** die vereinfachte mechanische Konstruktion und ab **3.1.2** eine schrittweise Entwicklung der Software des Laufsystems, die die Steuerung der externen Kontroller im Rahmen eines komplexeren Laufverhaltens umsetzt.

3.1.1 Mechanischer Aufbau

Die komplexe mechanische Konstruktion von AMEE kann mit dem Mindstorm® System kaum nachgebaut werden. Um die Grundkonzepte experimentell zu testen, ist es aber ausreichend feste Schrittlängen zu erproben, da nur die Interaktion zwischen zeitkritischen Kontrollern und dem eventgetriebenen Hauptsystem von Interesse sind. Im ersten Konstruktionsentwurf (siehe **Abb. 3.2**) wurde folglich auf das sich um die Vertikal-Achse drehende Schultergelenk für eine seitliche Verdrehung und zusätzlich auf das Fußgelenk verzichtet. Die Konstruktion hatte zwei Freiheitsgrade mit denen entlang der Laufrichtung und innerhalb des Workspace theoretisch alle Punkte auf einer Linie erreicht werden können. Der Endeffektor bot eine nahezu punktuelle Auflage und war mit einem Kontaktsensor ausgestattet. Erste rudimentäre Tests zeigten jedoch, dass das Drehmoment des oberen Motors nicht ausreichte, um die untere Beinkonstruktion über einen Winkel von etwa 30° anzuheben. Daher musste auf eine simplere kinematische Lösung zurückgegriffen werden.

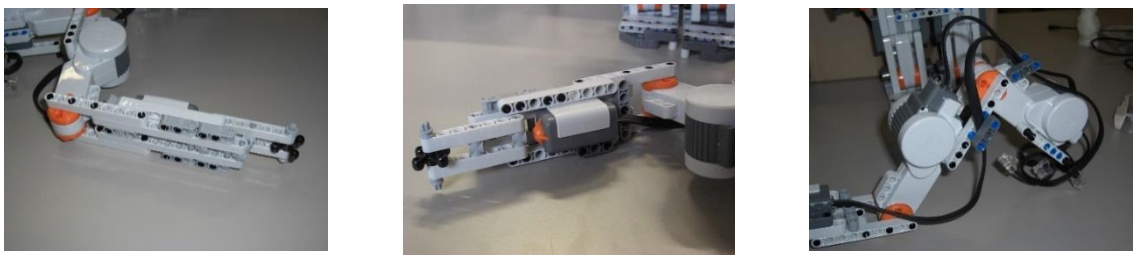
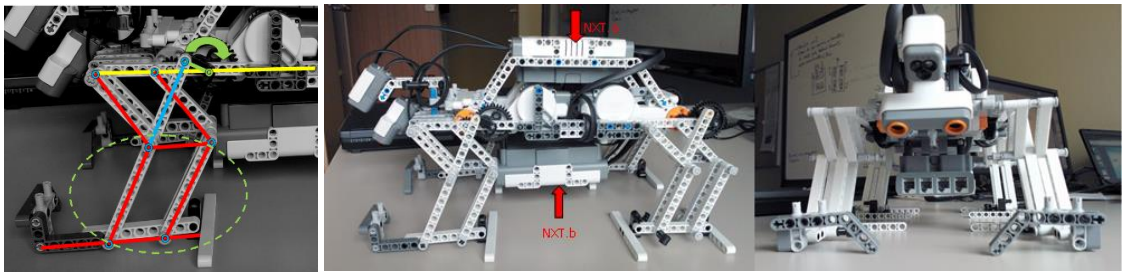


Abb. 3.2: erster Beinentwurf mit zwei Freiheitsgraden. Verworfen wegen eines zu geringen Drehmoments.

Das mechanische Konzept wurde für LittleAMEE schließlich auf insgesamt vier Antriebe abstrahiert. Jeder Fuß kann damit nur auf einer festen Kreisbahn bewegt werden. Dies wird in **Abb. 3.3 (a)** mit der unterbrochenen grünen Line dargestellt. Die kinematische Stützkonstruktion (rote Linien, **Abb. 3.3 (a)**) hält den Fuß immer parallel zum Torso (gelbe Line) des Roboters. Durch einen Exzenter wird die

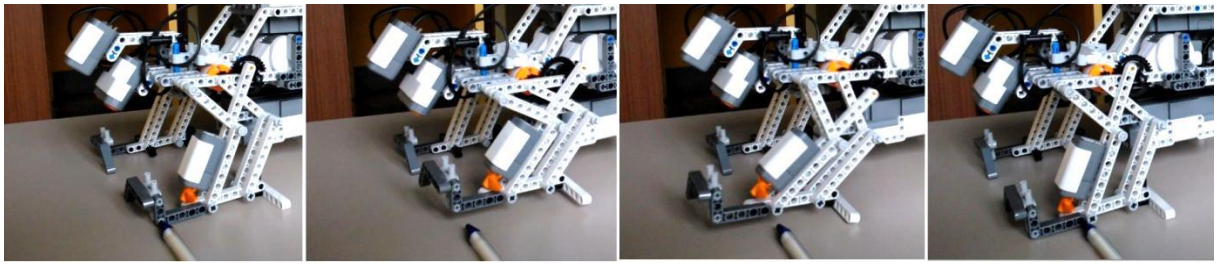
Drehbewegung des Antriebs mit einer 3:1 Untersetzung auf die Beinkonstruktion übertragen. Zwar reicht das Drehmoment der Antriebe des Mindstorm® V2 Systems, um auch ohne Untersetzung zu arbeiten, jedoch reagiert der NXT V2 Kontroller sehr empfindlich auf Drehmomentspitzen. Vermutlich soll damit einer Verletzungsgefahr mit dem auch von Kindern genutzten Baukastensystem vorgebeugt werden[6]. Mit diesem Aufbau können die vier Beine unabhängig bewegt werden. Der Lego® NXT V2 Kontroller kann drei Antriebe verwalten. Für diese Konstruktion werden zwei NXT Kontroller verwendet, die jeweils das vordere und das hintere Beinpaar ansteuern (**Abb. 3.3 (b)**, NXT.a & NXT.b). Die NXT Kontroller kommunizieren (über Bluetooth®) mit dem Hauptrechner, auf dem das Robotics Studio Framework arbeitet.

Das Voranschreiten des Roboters soll durch eine koordinierte Bewegung einzelner Beine erfolgen. Zu diesem Zweck, wurde die 360° Drehung eines Beinantriebs in vier Zwischenpositionen untergliedert (siehe **Abb. 3.4**). Idle steht für die Grundstellung, bei der die Konstruktion auch ohne aktives Drehmoment (ausgeschalteter Motor) stabil steht. Bei Up ist das Bein angehoben und bei Down vorgesetzt. Die letzte Position Pulled resultiert in derselben Stellung wie Idle, aber mit aktivem (bremsenden) Motor.



(a) (b) (c)

Abb. 3.3: (a) Aufbau des LittleAMEE Beins. (b),(c) Gesamtaufbau des Roboters



(a) (b) (c) (d)

Abb. 3.4: Zwischenpositionen des Bewegungsablaufs. (a) Idle, (b) Up, (c) Down, (d) Pulled (= Idle)

[6] Dem Projekt Team liegt der Source-Code des NXT V2 vor. Der Aufwand diesen Code zu verändern erschien uns aber zu groß, da die Drehmomentkontrollen an unterschiedlichen Stellen mehrfach implementiert wurden.

3.1.2 Verhaltensmodell

Der Projektversuch hatte das Ziel, die Steuerung eines Antriebs – der zeitkritische, reaktive Anteil in den AMEE-Beinen – zu einem Verhaltensmodell eines Beines zu abstrahieren und in einen Service zu überführen. Darüber hinaus galt es, die vier Beine (Instanzen des Beinservice) derart zu koordinieren, dass sich die Konstruktion voran bewegt (Orchestration). Dieser Anteil soll im späteren AMEE-Roboter von dem weniger zeitkritischen, planenden und ereignisgesteuerten Layer übernommen werden. Eine Nutzerschnittstelle setzte dann Befehle wie „gehe langsam“, „gehe schnell“ oder „halte an“ um (siehe **3.1.3**). Für die Umsetzung der Ziele wurde der Prozess einer evolutionären, experimentellen Prototypenentwicklung gewählt und die Visual Programming Language (VPL) genutzt, die neben dem Potential in der Schülerlehre eben für diese Art der Prototypenentwicklung besonders geeignet ist.

In der Endphase wurde schließlich eine verteilte Roboter-Anwendung, bestehend aus funktional getrennten Software Services (DSS) die zu einem Gesamtverhalten orchestriert werden, in C# statt mit VPL entwickelt. Per Manifest, erstellt mit dem Manifest Editor des Robotics Studio, wird stattdessen definiert, welche Services gestartet werden sollen, welche Startkonfiguration sie haben und welche Serviceinstanz Partner einer anderen Serviceinstanz ist. Die Anwendung kann dann durch Starten eines DSS-Nodes mit Übergabe der Manifest Datei initiiert werden. **Abb. 3.5** gibt eine Gesamtübersicht der DSS-Anwendung für LittleAMEE.

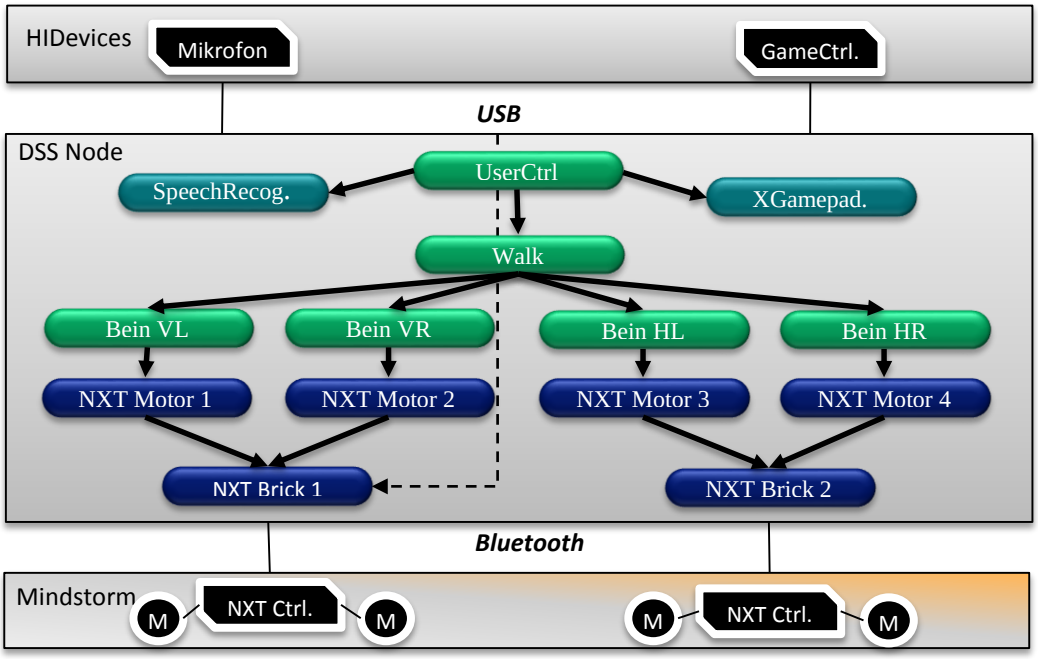


Abb. 3.5: Systemübersicht der LittleAMEE Roboteranwendung. Blau: Services der Hardware Steuerung. Grün: Services der Verhaltens Orchestration. Dunkelgrün: Services der Mensch-Maschine Schnittstelle. Schwarz: Aktorik, Sensorik, Hardware.

3.1.3 Mensch-Maschine-Interaktion

Zum einen wurde eine Steuerung per Gamecontroller, wie er von Spielekonsolen bekannt ist, eingesetzt, und des Weiteren eine Sprachsteuerung etabliert. Sowohl Gamecontroller als auch Mikrofon werden dafür an den PC oder Laptop angeschlossen, auf dem die DSS Roboteranwendung läuft. Der Zugriff erfolgt über die mit Robotics Studio mitgelieferten Services *XInputGamepad* und *SpeechRecognizer*, welche dem neu erstellten Service **LittleAMEEUserControl** als Partner beiseite gestellt werden.

Tabelle 3.1 listet die in der User Control umgesetzten Befehle auf. Da der NXT-Controller selbst keine Sprachausgabe hat, aber einzelne Töne abspielen kann, wurde zusätzlich diese Funktion für eine optionale Roboterrückmeldung getestet, die die Varianten einer Mensch Maschine Interaktion potentiell erweitern. Beim Sprachbefehl „AMEE gib laut“, spielt der Roboter die aus dem Kinofilm „Ein unheimliche Begegnung der dritten Art“ bekannte Begrüßungstonfolge[7] ab.

Tabelle 3.1: Nutzer Befehle

Walk-Befehl	Gamepad Input	Sprachbefehl
Go	DPad hoch	„AMEE[8] vor“
SwitchGait(fast)	DPad rechts	„AMEE schnell“
SwitchGait(safe)	DPad links	„AMEE langsam“
SwitchGate	DPad runter	---
Stop	---	„AMEE anhalten“
Maintain	---	„AMEE bereit“
---	Start Knopf	„AMEE gib laut“

3.1.4 Fazit

Es wurde gezeigt, dass unter der Prämisse, dass der Roboter AMEE ein statischer Läufer ist, eine ereignisgesteuerte, serviceorientierte Orchestration vier einzelner, externer Beinsteuerungen, die die ausgelagerten Echtzeitanteile des Gesamtlaufsystem kapseln, zu einem komplexeren Verhalten mit multiplen Laufarten möglich ist. Die Systemintegration des Kontrollerkonzepts aus ([Ruh13], [Ruh11], siehe auch **Abb. 3.6**) nach DSS gemäß den Vorüberlegungen [Bet10] war erfolgreich. Darüber hinaus wurden erste Ansätze der Mensch-Maschine Interaktion umgesetzt, die das Softwaresystem des Roboters zu einer prototypischen Anwendung komplettieren.

Letztlich wurde die Beinkoordination doch außerhalb der DSS-Architektur umgesetzt [Ruh14]. Denn im bisherigen Konzept liegt die Kontrolle der Beine teilweise in der reaktiven pre-online Schicht (**Abb. 3.6**)

[7] g', a', f', f, c'
[8] Sprich »Ejmi«

in der auch Informationen über die Neigung und Beschleunigung des Torsos einfließen und gegebenenfalls eine schnelle Systemreaktion erfordern. In der DSS[9]-Architektur (vgl. [Bet10]) ist es vorgesehen, Services für häufige und schwach zeitkritische Aufgaben auf einem separaten Rechner und langwierigere oder nicht zeitkritische Services auf einem weiteren Rechner auszuführen. Dies könnte ein geeigneter Ansatz sein, wurde aber aus strategischen Entscheidungen nicht näher untersucht.

Der in [Bet10] dargestellte Vorteil der Robotics Studio Softwarearchitektur im Umgang mit asynchronen Aufgaben (CCR[10]) speziell im Kontext einer verteilten Roboteranwendung ist mehr oder wenig hinfällig geworden, da mit .NET 4.0 und der Task Parallel Library (TPL) dieselben Konzepte der asynchronen Programmierung inzwischen nativ unterstützt werden. Darüber hinaus hat das Unternehmen Microsoft im September 2014 überraschend die Robotics Research Abteilung aufgelöst, und somit die Weiterentwicklung des Robotics Studios eingestellt. Dennoch vereinfacht die DSS-Architektur weiterhin die Entwicklung einer verteilten Roboteranwendung. Daher werden voraussichtlich zwar einzelne Aufgaben (Algorithmen, Kommunikation, etc.) als separate und gegebenenfalls systemnative Programmbibliotheken entwickelt, aber über DS-Services in die Gesamtanwendung integriert und orchestriert.

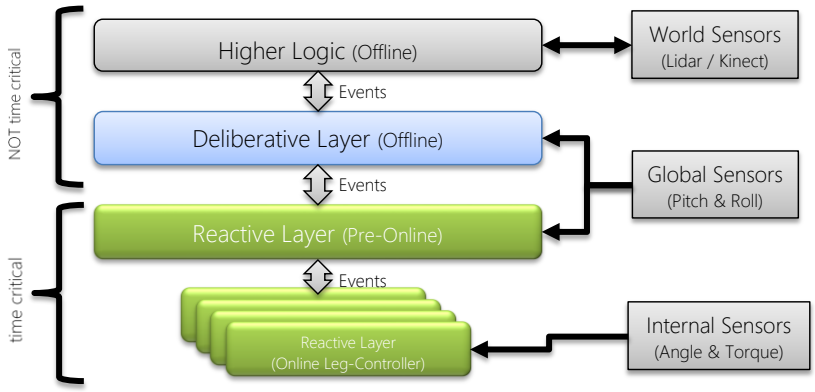


Abb. 3.6: Hauptkontroller (Laufsystem) Layerarchitektur des Roboters AMEE [Ruh13]

[9] Decentralized Software Services
[10] Concurrency and Coordination Runtime

3.2 Robot Vision

Nachdem sich der vorige Abschnitt über LittleAMEE mit einem grundlegenden softwaretechnischen Konzept für eine verteilte Roboteranwendung beschäftigte, sollen nun ein perspektivischer Ansatz zur visuellen Erfassung der näheren Umgebung und Datenvorverarbeitung im Kontext der Lernaufgabe für sichere Schrittwahl angegangen werden. Auf Basis des hier vorgestellten, noch unabhängigen, Prototyps einer visuellen Analyse können künftig weitere Konzepte zur Merkmalsgewinnung für den Lernalgorithmus erarbeitet und erprobt werden. Vor allem aber hilft dieser Ansatz dem Verständnis der Modellierung der Lernaufgabe, die in **Abschnitt 4** vorgenommen wird, bei der eine fiktive Vorverarbeitung eines Bodenscans vorausgesetzt und mit einem abstrakten Unebenheitsmaß gearbeitet wird.

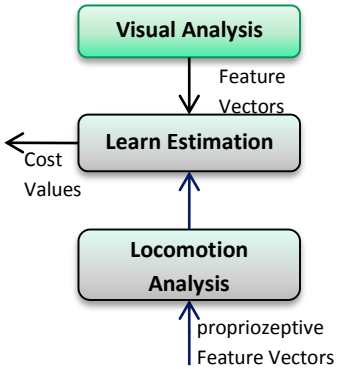


Abb. 3.7: Einordnung der Robot Vision in die Systemarchitektur (vgl. Abb. 1.1).

3.2.1 Motivation

Der Quadruped Rough Terrain Robot (QRTR) ist ein pseudodynamischer Läufer dessen Kontrollerkonzept u.a. eine deliberative Architekturschicht vorsieht (**Abb. 3.6**), in der fortlaufend Trittpositionen für die einzelnen Beine unter Berücksichtigung der gewünschten Richtung und aktuellen Pose und Beschleunigung berechnet werden. Da es auch stark zerklüftetes Gelände zu bewältigen gilt, wird eine dynamische Kinematik benötigt, die der Laufmaschine ein sicheres Vorankommen ermöglicht [Ruh14].

Ein möglicher Ansatz wäre ein reaktives System, das von einem festen Schrittmuster ausgehend[11], welches von der deliberativen Schicht vorgegeben wird, eine dynamische Anpassung des Arbeitspunktes zur Erhaltung der Stabilität und des Gleichgewichts vornimmt. In diesem Zusammenhang ist auch ein Regelwerk mit einer Neurofuzzyregelung denkbar. Forscher an der Stanford University setzten hingegen auf eine vorangehende, detaillierte visuelle Erfassung des Geländes und einem Expertenregelwerk (siehe [Kol09], **2.1.2**). Für kleine Geländeausschnitte und verschiedene Geländeformen gab ein Experte die beste Trittposition vor. Nach einem Geländescan versucht das System,

[11] Bspw. Fuß immer einen halben Meter vor und unter den Torsomittelpunkt setzen

ähnliche Formen wiederzuerkennen, um dann mittels des Expertenwissens die beste Trittposition auszuwählen.

Unser Ansatz für den deliberativen Kontroller ist es, in relativ kurzen Zeitabständen ebenfalls einen Scan des vorausliegenden Geländes vorzunehmen und halbwegs ebene Flächen zu finden, die in etwa die Größe des Roboterfußes haben. Ein 3D-Scanner (LIDAR, Kinect®, Stereokamera,…) liefert hierfür eine 3D Punktwolke und der RANSAC-Algorithmus sucht darin Flächen. RANSAC ist ein etabliertes und relativ performantes Clustering Verfahren, wenn es darum geht, einfache geometrische Modelle in einer Punktwolke zu finden. Einzelheiten zum Algorithmus und seinen Varianten sind in der gängigen Fachliteratur nachzulesen und sollen an dieser Stelle nicht tiefer beleuchtet werden. Der erhoffte Vorteil dieser Methode ist es, ein sichereres und adaptiveres Verfahren zur Fortbewegung zu erhalten, als es ein rein reaktives, geregeltes System bieten würde aber auch gleichzeitig ein performanteres Verfahren gegenüber eines auf Expertenwissen basierendes, detailliertes Analysemodell.

3.2.2 Vorbereitung

Als 3D Scanner steht die kostengünstige Kinect® for Windows und das zugehörige SDK zur Verfügung. Zum Zeitpunkt des Versuchsaufbaus wird die Developer Preview Version der Kinect 2 verwendet. Großteile der höheren Softwarelogik sind in C# (.NET) entwickelt, daher werden .NET Frameworks bevorzugt. So basiert die RANSAC Implementierung auf dem Accord Framework[12], welches wiederum eine Erweiterung von AForge[13] ist. Dabei konnte im Zuge der Umsetzung des prototypischen Versuchsaufbaus die eher akademisch motivierte Implementierung noch erheblich optimiert werden. Um das Verfahren überprüfen zu können, ist eine Visualisierung der Ergebnisse unbestreitbar von großem Vorteil. Eine einfache interaktive 3D Grafik ist folglich Bestandteil des Versuchs. Das Helix 3D Toolkit[14] erweitert die WPF[15] nativen Möglichkeiten zum Erstellen und Anzeigen solcher Grafiken. Alternativ hätte sich auch die offene, in nativem C++ geschriebene, PCL-Library angeboten, die diverse Algorithmen rund um Punktwolken, auch verschiedene RANSAC-Varianten, implementiert hat und Hilfsmittel zur Visualisierung mitbringt. Jedoch wäre die Integration in den auf .NET basierenden Versuchskontext zu aufwändig. Die Wahl der Basisframeworks hat letztlich keinen Einfluss auf die grundsätzliche Methodik des Verfahrens oder der Ergebnisanalyse. Im Folgenden wird die Software-Architektur des entwickelten Robot Vision Prototyps vorgestellt, um an ihr die Ansätze zur visuellen Umgebungserfassung zu erläutern. Es wird dabei kein Anspruch auf Vollständig erhoben, sondern stellt nur eine Konzeptstudie im Kontext der Aufgabe, eine sichere Schrittposition zu finden, dar.

[12] http://accord-framework.net/ (zuletzt besucht am 22. Sep 2014)
[13] http://www.aforgenet.com/ (zuletzt besucht am 22. Sep 2014)
[14] https://github.com/helix-toolkit (zuletzt besucht am 22. Sep 2014)
[15] **W**indows **P**resentation **F**ramework

Die Software untergliedert sich dabei grob in vier funktionale Einheiten (vgl. **Abb. 3.8**):

- **`K4Wv2PointCloud`**
  Steuerung der Sensorik und des zugehörigen SDK
- **`RANSAC`**
  Umsetzung des RANSAC-Algorithmus
- **`RansacPlane`**
  Kapselung des Algorithmus, spezialisiert auf Ebenen in einer Punktwolke
- **`PlaneFinder`**
  Hauptklasse zum Erfassen von Ebenen, Kapselung von Sensorik und Datenverarbeitung

Der Weg von einem Scan hin zu Ebenen im Raum führt dabei über den Sensor, der ein Tiefenbild liefert über eine optionale Segmentierung, um mehrere Ebenen zu finden, einer Umwandlung der Rohdaten zu einer 3D-Punktwolke und der letztlichen Anwendung des RANSAC-Algorithmus.

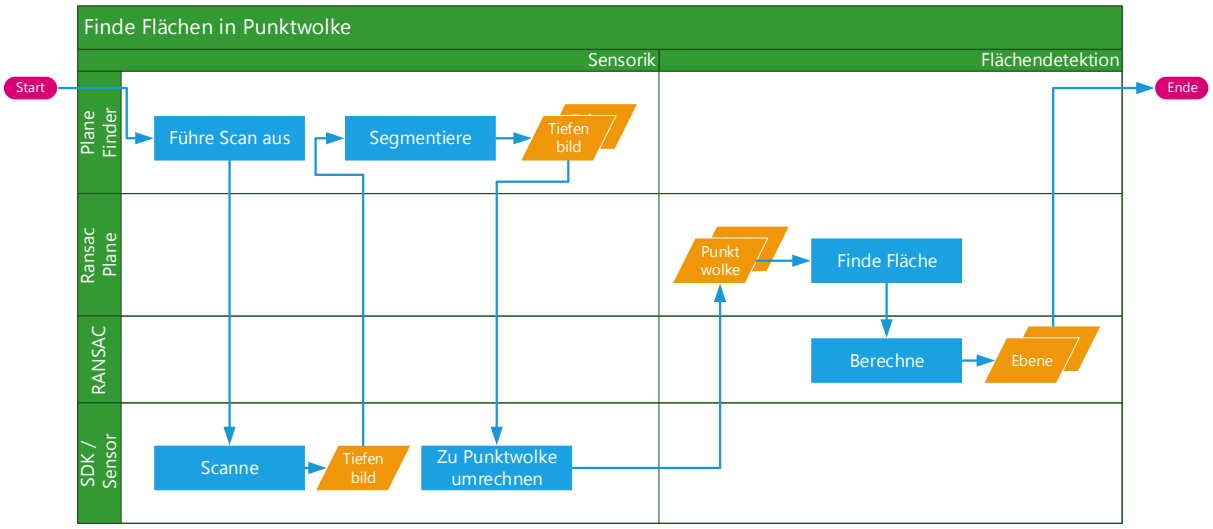


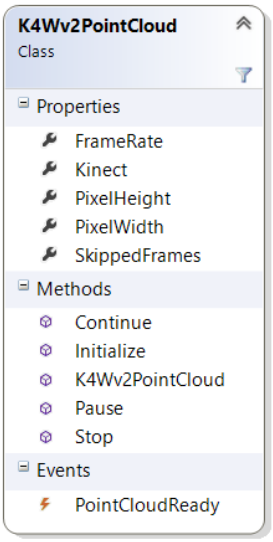
Abb. 3.8: Funktionaler Ablauf der Flächendetektion untergliedert nach Funktionseinheiten des Prototyps.

3.2.3 Punktwolke erfassen

Die Kinect® liefert mit etwa 30 Bildern pro Sekunde ein Farb-, ein Tiefen- und ein Infrarotbild sowie ein Skeletonframe, welches bis zu sechs Personen erfasst. Jeder Frametyp kann separat abgefragt werden und wird mit einem FrameArrived Event angekündigt. Die Übertragung der Daten erfolgt erst und nur dann, wenn mit einem Reader das zum Event gehörende Frame abgerufen wird. Geschieht dies nicht schnell genug, wird der Frame hardwarelokal überschrieben. Alternativ bietet der Reader noch eine Methode zum Abrufen des aktuellsten Frames an. Der Vorteil dieses Hardware- und SDK-Merkmals liegt definitiv darin, dass weder Speicher- noch Rechenressourcen unnötig beansprucht werden, der Nutzer entscheidet, wann Daten abzuholen und zu verarbeiten sind. Zur Erfassung einer 3D Punktwolke ist nur das Tiefenbild relevant: Ein 2D Pixelbild mit Tiefeninformationen. Sind

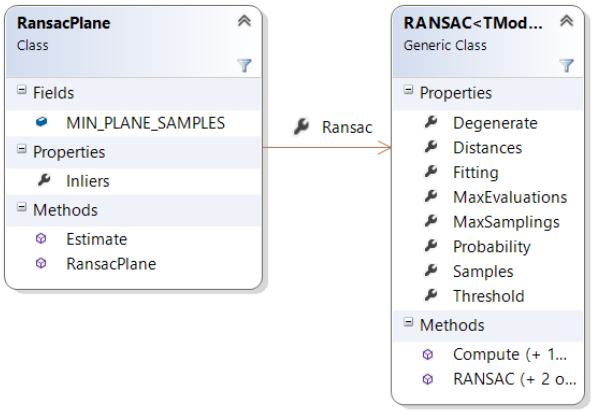
Sichtfeldwinkel und Linsenverzerrung bekannt, lässt sich das Tiefenbild in 3D-Punkte umrechnen, was das Kinect® SDK bereits über angebotene Methoden übernimmt.

Mit **`K4Wv2PointCloud`** wurde eine auf die Erfassung einer Punktwolke spezialisierte Wrapperklasse entwickelt. Sie steuert den Sensor, ruft die Tiefenbilder ab, nimmt eine optionale Segmentierung vor und bildet daraus die 3D-Punktwolke, welche wiederum durch ein klasseneigenes Event `PointCloudReady` bereitgestellt wird. Das Tiefenbild, welches mit 30 Frames pro Sekunde geliefert wird, enthält etwa 217 Tausend Pixel. Je nach Rechenressourcen erfolgt die Umrechnung zur Punktwolke schnell genug, um die Framerate auch für Punktwolken einzuhalten. Unter Umständen ist aber die Weiterverarbeitung zu aufwändig für 30 Punktwolken pro Sekunde. Die Daten würden dann im Speicher gepuffert werden und das Rechnersystem über kurz oder lang überlasten. Deshalb kann mit **`FrameRate`** eine maximale Frequenz für das `PointCloudReady` Event eingestellt werden. Intern werden dabei solange Events der Kinect® verworfen, bis das eingestellte Zeitintervall erreicht ist. Alternativ kann mit **`SkippedFrames`** die Anzahl zu verwerfender Tiefenbilder eingestellt werden.



3.2.4 Eine Fläche detektieren

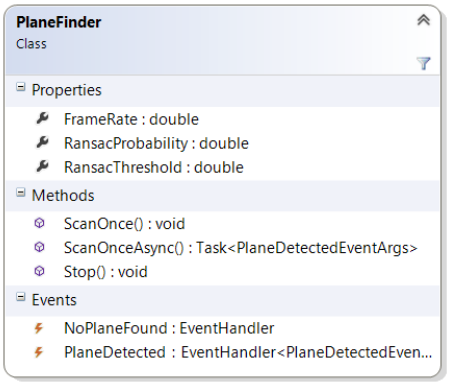
Mit dem RANSAC Algorithmus lassen sich Flächen in Punktwolken detektieren. Das Accord Framework dient hierbei als akademische Implementierungsbasis. Die Klassen **`RansacPlane`** und **`RANSAC`** wurden für das eigene Projekt lediglich hinsichtlich einer Parallelisierung mittels TPL[16] und weitere Performance steigernden Maßnahmen angepasst.



Für eine Detektierung von ebenen Flächen in einer Punktwolke wird **`RansacPlane`** benötig, `RANSAC` ist ein automatisch konfigurierter Bestandteil der Klasse, worin der Algorithmus allgemein umgesetzt ist. Der Methode **`Estimate(Point3)`** übergibt man die Punktwolke und erhält die Struktur **`Plane`** zurück. Sie enthält die Flächengleichung und Ebenennormale der ermittelten Fläche, die bei einer begrenzten Anzahl von Iterationen diejenige mit den meisten zugehörigen Punkten (Inliers) ist. Dabei werden die Indizes der Inlier-Punkte im Array der **`Inliers`**-Property[17] gespeichert. Diese können für gegebenenfalls

[16] Task Parallel Library aus .NET
[17] Als Properties werden in C# Klassenfelder mit getter- und setter-Methode bezeichnet

weitere Analysen verwendet werden. So gibt das Verhältnis der Inliers-Anzahl zur Gesamtzahl aller Punkte unter Umständen ein Maß über die Zerklüftheit oder Rauheit der Fläche im Kontext ihrer Umgebung und ist im Zusammenhang mit der Ausrichtung der Ebenennormalen als Steilheitsmaß ein Indikator für die Güte als Trittposition.



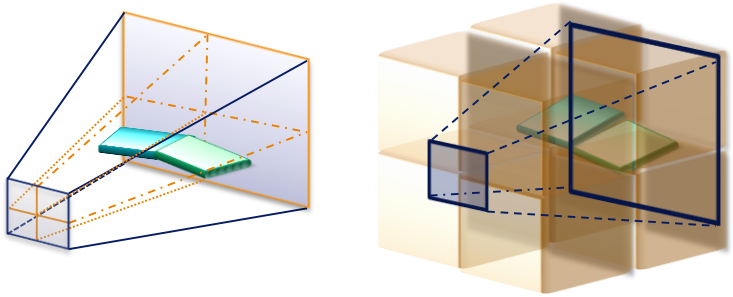
Die in diesem Prototyp nach außen wichtigste Klasse und Schnittstelle zum Sensor und Algorithmus ist der **`PlaneFinder`**. Er kapselt die Verwendung von `K4Wv2PointCloud` und `PlaneRansac`. `PlaneFinder` bietet eine asynchrone und eine eventbasierte Methode an, um einen einmaligen Scan vorzunehmen und darin eine Fläche zu detektieren. Die eventbasierte Methode **`ScanOnce()`** meldet bei der `K4Wv2PointCloud` einen `PointCloudReady` Eventhandler an, der die gelieferte Punktwolke an `PlaneRansac` übergibt und letztlich seinerseits das Ergebnis über das **`PlaneDetected`** Event publiziert. Unter Umständen wird keine Fläche gefunden, was durch das **`NoPlaneFound`** Event bekannt gegeben wird. Der Nutzer von `ScanOnce()` sollte sich folglich bei beiden Events anmelden. Die asynchrone Methode **`ScanOnceAsync()`** gibt die Prozesskontrolle zurück bis das `PointCloudReady` Event erscheint. Auch hier wird die Punktwolke an `PlaneRansac` übergeben, aber das Ergebnis dann direkt zurückgegeben. Der Nutzer von `ScanOnceAsync()` kann mit dem Schlüsselwort `await` auf das Ergebnis warten.

3.2.5 Mehrere Flächen detektieren

An sich kann durch RANSAC nur genau eine Ebene (oder anderes geometrisches Modell) ermittelt werden. Zum Aufspüren multipler Flächen in einer Punktwolke mittels RANSAC gibt es verschiedene Verfahren. Beispielsweise kann der Algorithmus mehrfach angewandt werden, wobei nach jedem Durchlauf die zur gefundenen Fläche gehörigen Punkte aus der Punktwolke subtrahiert werden, um auf die restliche Punktwolke den Algorithmus erneut loszulassen.

Aufgrund des Bestrebens etwa Roboterfuß große Ebenen zu ermitteln, ist unser erster Ansatz stattdessen erst das Tiefenbild beziehungsweise die Punktwolke zu segmentieren, um auf jedes Segment den Algorithmus einmal anzuwenden. Dabei gibt es einen wesentlich Unterschied, ob das Tiefenbild oder die bereits daraus errechnete Punktwolke segmentiert wird. Bei der Unterteilung des Tiefenbildes mit anschließender Umwandlung der Subbilder in mehrere Punktwolken entstehen röhrenähnliche Sichtfelder innerhalb derer jeweils eine Ebene detektiert werden kann (**Abb. 3.9 (a)**). Gegenüber der Segmentierung der Punktwolke spart dies Rechenressourcen, jedoch sind Größe und auch Form der gefundenen Flächen stark von der Sensor-Perspektive abhängig. Unterteilt man hingegen die Punktwolke mit ihrem räumlich kartesischen Koordinatensystem in Subwürfel (manchmal auch Voxel genannt), sind die räumlichen Grenzen klarer definiert und weitere Analysen der Flächen oftmals leichter umzusetzen (**Abb. 3.9 (b)**). Die Vor- und Nachteile beider Methoden sind

bei Bedarf und Plattform individuell abzuwägen. Ohne eine Präferenz andeuten zu wollen, arbeitet dieser Prototyp mit einer Segmentierung des Tiefenbildes.



(a) (b)

Abb. 3.9: Segmentierung des Tiefenbildes vs. Punktwolke zum Detektieren multipler Flächen. Wahl der Methode hat Auswirkung auf Größe und Form gefundener Flächen.

3.2.6 Visualisierung

Die graphische Visualisierung des Prototyps dient der Kontrolle über das Verhalten des Algorithmus und Auswirkung der Parametriesierung. Theoretisch und für die künftige Integration in das Gesamtsystem ist sie irrelevant, da die abstrahierten Flächendaten selbstverständlich ausreichen. Der Nutzer kann in der Oberfläche den RANSAC-Algorithmus parametrisieren und die Segmentierung in Zeilen und Spalten angeben. Das Hauptbild bietet eine im virtuellen Raum navigierbare 3D-Ansicht des Scan- und Detektierungsergebnisses in der Messpunkte ohne Flächenzugehörigkeit gelb und Flächenpunkte in unterschiedlichen Farben dargestellt werden. In der **Abb. 3.10** ist gut zu erkennen, wie die Bodenkacheln, perspektivisch bedingt durch die Tiefenbildsegmentierung, Trapezformen annehmen.

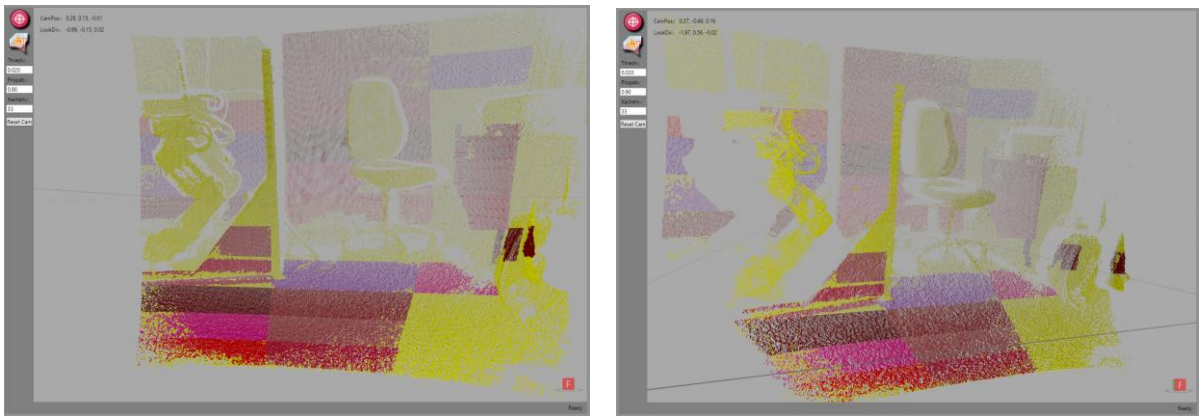


Abb. 3.10: Ein Labor-Scan in zwei Perspektiven – Erkannte Flächen, beziehungsweise Kacheln in der Punktwolke farbig hervorgehoben. In den gelben Bereichen wurde keine Fläche detektiert.

3.2.7 Fazit

Das Experiment mit dem Kinect® Sensor und dem RANSAC Algorithmus zeigt einige Ansätze zur Merkmalsgewinnung und Datenvorverarbeitung für den Lernalgorithmus, der auf Basis einer visuellen Umgebungsanalyse Schätzungen abgeben soll. Darüber hinaus bietet es für die Modellierung der Lernaufgabe eine Vorstellung über die Bedeutung letztlich abstrakter Zustands-Kennzahlen, wie „Unebenheit“ oder „Steilheit“ und gibt Anregung für die Entwicklung ergänzender, analytischer und regelbasierter Systeme im Deliberative Layer des Roboters [Ruh14].

3.3 RL Framework

Die Beurteilung, was eine gute mögliche Schrittposition ist, kann erlernt werden – so die Thesis dieser Arbeit – und zwar mittels des Reinforcement Learning Algorithmus Sarsa(λ). Zur letztlichen programmiertechnischen Umsetzung der Lernaufgabe, aber vor allem für die Eignungsprüfung und Analyse von detaillierteren Aspekten des Algorithmus ist es sinnvoll, ein Rahmengerüst für das Design und die Implementierung der multiplen Experimente heranzuziehen. Das eigens hierfür entwickelte RL Framework *RLLibrary.NET*, wobei RL für Reinforcement Learning steht, bietet eine modular aufgebaute, flexible Software Vorlage.

RL Glue – der Akademikerstandard

Ein im akademischen Umfeld weit verbreitetes Open Source Framework für Reinforcement Learning ist RL Glue[18] [Tan09]. Das Ziel dieses Software Tools ist es, verschiedene Lösungen miteinander vergleichen zu können und es legt hierfür ein Hauptaugenmerk auf den modularen Aufbau der RL-Architektur und der socket-basierten Kommunikation zwischen den Modulen. Es werden Middleware-Schnittstellen für mehrere Programmiersprachen wie C++, Java, Lisp oder die Software MATLAB angeboten.

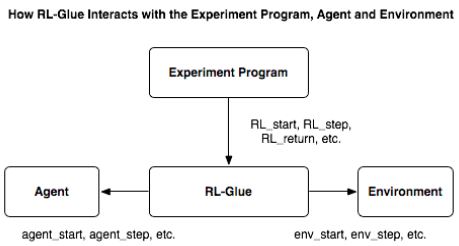


Abb. 3.11: RL Glue Software Architektur. Pfeile entsprechen Richtung der Funktionsaufrufe
http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html
(Stand 27.04.2014)

Trotz der Chancen, die das fertige Framework mit sich bringt, wurde mit Hinblick auf eine Integration in das zu großen Teilen auf .NET basierte AMEE Gesamtsystem und der Anwendungsdomäne embedded robotics applications auf Basis der RL Glue Java Middleware ein .Net Framework entwickelt. Losgelöst von den multilingualen Restriktionen erfolgte anschließend ein komplettes Reengineering, so dass letztlich ein neues RL Framework entstand, die RLLibrary.NET, mit generischen Ansätzen, wo dies sinnvoll erschien. Eine Motivation hierfür war unteranderem, dass dadurch auch auf die ASCII

[18] Erhältlich unter: http://glue.rl-community.org/wiki/Main_Page (letzter Zugriff 27.04.14)

basierte Kommunikation über Sockets verzichtet und weitere systemspezifische, Performance steigernde Maßnahmen ergriffen werden können. Des Weiteren erhöht eine weitergehende Modularisierung des Frameworks die Flexibilität in der Implementierung.

3.3.1 RLLibrary .NET

Ausgehend vom Grundmodell (siehe **Abb. 2.6**) ist die **Umwelt** derart modelliert, dass sie eine Methode anbietet, um eine Aktion auszuführen und die Reward und Folgezustand zurückliefert. Die Komponente **`IEnvironment`** ist dafür zuständig, die Welt zu modellieren. Aus Sicht der Robotik ist sie eine Schnittstelle zwischen der für das Lernen notwendigen, zusammengefassten Eigen- und Außenwahrnehmung der Maschine und ihrer Steuerung. Die Methode **`DoAction()`** führt einen abstrahierten Befehl aus, an dessen Ablaufende dieser bewertet und die neue Wahrnehmung zum Folgezustand des Weltmodells überführt wird.

Die wesentliche Methode der **Agenten** Komponente **`IAgent`** ist die wiederholte Ausführung eines Lernschritts: **`DoStep()`**. Sie ist der Kern des jeweiligen RL-Algorithmus, wo Aktionen auf Basis der aktuellen Strategie ausgewählt, mittels der Umweltkomponente durchgeführt und die Bewertungsfunktion angepasst wird. Der Agent und sein Lernalgorithmus wird parametrisiert mit der Lernrate α, dem Abschwächungsfaktor γ sowie für die TD(λ) Varianten mit dem Spurzerfallsfaktor λ.

Hilfsfunktionen, die den jeweiligen Lernalgorithmus begleiten, sind auf oberster Ebene der Software Architektur als separate Submodule des Agenten modelliert. So sind die Strategie-, Bewertungs- und Lohnberechtigungsfunktion außerhalb des Agenten extra zu implementieren. Was auf den ersten Blick vielleicht als architektonischer Overhead erscheint, ermöglicht letztlich die einfache Austauschbarkeit einzelner Funktionen und erhöht somit die Flexibilität des Frameworks mit Hinblick auf künftige Weiterentwicklungen und Optimierungen des Robotersystems bezüglich Datenstrukturen, Performance oder in begrenztem Maße auch Verfeinerungen des Lernverfahrens an sich.

Für die Aktionswahlstrategie gibt es zwei Ausprägungen. Die im RL-Kontext ursprüngliche Variante π ist eine **`StochasticPolicy`**, die zu einem Zustand s die Wahrscheinlichkeit wiedergibt, die Aktion a zu wählen ($\mathcal{S} \times \mathcal{A} \rightarrow [0,1]_{\mathbb{R}}$). Bei den TD-Varianten wie Sarsa-λ hingegen wird meist eine **`ExplorationPolicy`** verwendet, die die Aktion aufgrund eines Satzes von Basiswerten, normalerweise den Bewertungen der möglichen Folgezustände, wählt ($\mathbb{R}^n \rightarrow \mathcal{A}$). Die Schnittstelle für Strategieimplementierungen gibt somit nur die entsprechende funktionale Abbildung vor.

Auch die Bewertungsfunktion ist vordergründig erst einmal eine Abbildung von Zustand, beziehungsweise Zustands-Aktions-Paar auf einen reellen Wert und ein konkretes Agentenmodul implementiert je nach RL-Variante **`ValueFunction`** V oder **`StateValueFunction`** Q, ob als Matrix, Liste oder einer anderen gerade passenden Datenstruktur.

In den TD(λ)-Verfahren kommt noch der Lohnberechtigungsfaktor *eligibility* hinzu. Viele RL-Umsetzungen speichern diesen gemeinsam mit den Zustandsbewertungen ab. Da aber die Berechnung des *eligibility* variieren kann, ist auch hierfür eine **`EligibilityFunction`** vorgesehen, die zu einem Zustands-Aktions-Paar ihre Lohnberechtigung wiedergibt ($\mathcal{A} \times \mathcal{S} \to \mathbb{R}$). Soll eine Datenstruktur beispielsweise beides halten, den Bewertungs- und Lohnberechtigungswert, implementiert sie dann `StateValue-` und `EligibilityFunction`.

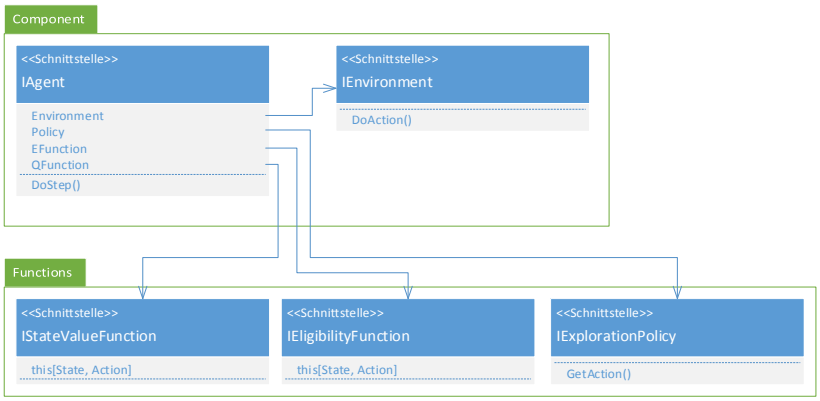


Abb. 3.12: Vereinfachtes Klassendiagramm aus RLLibrary. Eine RL-Software implementiert auf oberster Ebene die Umwelt- und Agentenkomponente. Der Kern des Lernalgorithmus liegt im Agenten, für den weitere Datenstrukturen und Hilfsfunktionen umzusetzen sind, wie bspw. Q-Funktion, E-Funktion und die Strategie.

Damit ist eine flexible, modulare Grundstruktur für verschiedene Umsetzungen von Reinforcement Learning gegeben, ob für Dynamic Programming, Monte Carlo Methoden oder Temporal Diference Learning (vgl. [Sut98]). Ein Vorteil der RLLibrary .NET gegenüber RL-Glue ist die optionale Verwendung von generischen Typen für Zustand und Aktion und der damit einhergehenden Typsicherheit sowie eines potentiell geringeren Daten-Overheads[19]. Zu jeder Hauptkomponente und Hilfsfunktion bietet RLLibrary noch eine generisch typisierte Variante auf Basis der Marker-Interfaces **`IState`** und **`IAction`** an.

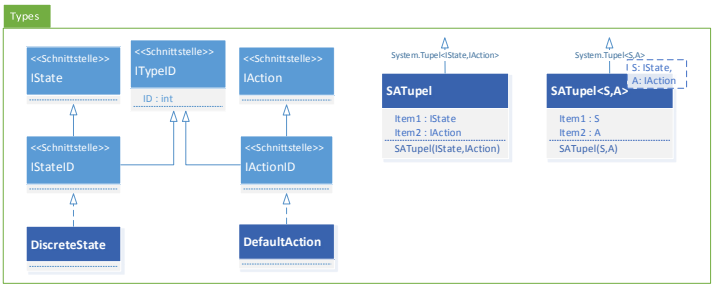


Abb. 3.13: Klassendiagramm. RLLibrary Datenmodell für Zustand und Aktion.

[19] Die Zustands- und Aktionsdatenstrukturen in RLGlue sehen Felder für int, double und char gleichzeitig vor. Es obliegt dem Programmierer darauf zu achten, auf die für ihn gerade richtigen Felder zuzugreifen.

3.3.2 Sarsa(λ) Programmiermodell

Auf Basis der eingangs vorgestellten RLLibrary Grundstruktur lässt sich der Sarsa(λ)-Algorithmus zu großen Teilen vorimplementiert und generisch anbieten. Wie sich weiter unten zeigt, kann der in RLLibrary enthaltene **`SarsaLambda<S,A>`** nicht nur für verschiedene Aufgabenmodelle, sondern darüber hinaus auch für unterschiedliche Ausprägungen des Algorithmus wie *Linear Gradient Descent Sarsa(λ)* (siehe **2.3.3**) verwendet werden, was durch die Modularisierung der Hilfsfunktionen erreicht wird. So muss der Anwender nicht den gesamten Algorithmus, sondern nur die Zustands-Wert-Funktion Q und die Strategie implementieren, wobei für letztere auch auf die in der RLLibrary enthaltenen `EpsilonGreedy` Klasse zurückgegriffen werden kann – eine Implementierung der häufig eingesetzten gleichnamigen Strategie. Die Umsetzung der Umwelt-Komponente ist selbstverständlich eine separate Aufgabe, die dem Anwender nicht abgenommen werden kann. Die Referenz darauf muss dem `SarsaLamda`-Agenten bei der Instanziierung übergeben werden. Des Weiteren verwendet `SarsaLambda` eine Standardumsetzung der Eligibility-Funktion, die aber vom Anwender durch eine eigene ausgetauscht werden kann. Für Effizienztests und anderen Analysen des Lernalgorithmus bietet `SarsaLambda` neben der Anzahl von Schritten je Lernepisode die oft auch aufschlussreiche erzielte Gesamtbelohnung an.

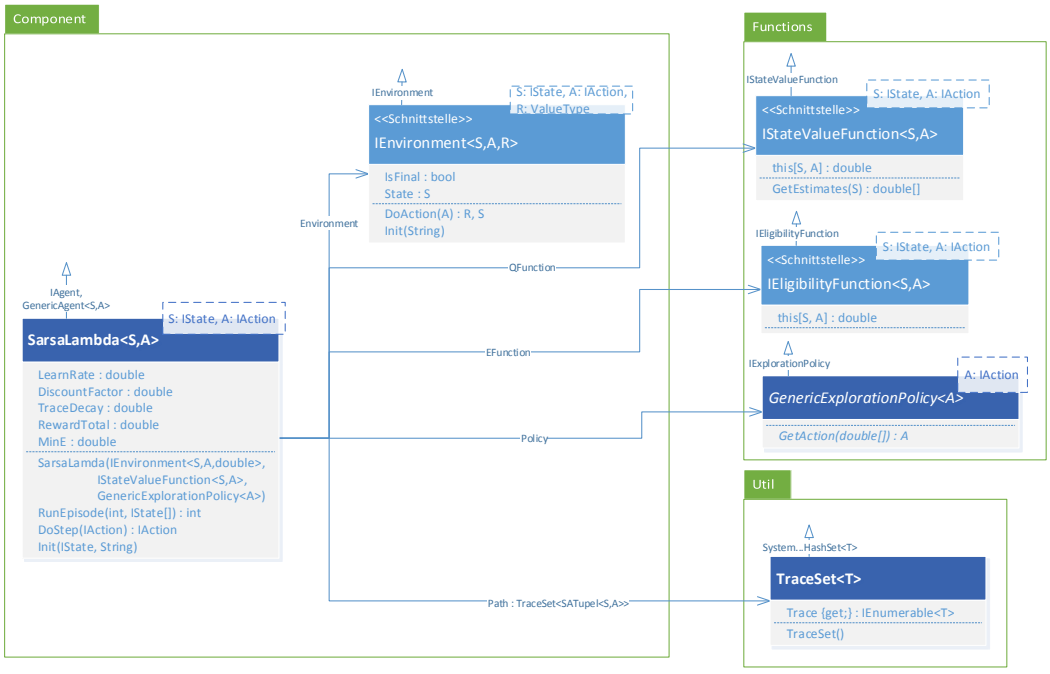


Abb. 3.14: Klassendiagramm. Basisprogrammiermodell für Sarsa(λ) in der RLLibrary. Dem Agenten *SarsaLamda* muss bei Instanziierung noch die Umwelt, Zustands-Wert-Funktion Q und Strategie übergeben werden.

Damit bietet RLLibrary .NET die fortschrittlichste und wichtigste Methode der in [Sut98] vorgestellten grundlegenden RL-Verfahren an. Eine Erweiterung des Frameworks um weitere Basis-Algorithmen ist

zwar möglich, aber mit Hinblick auf das Zielsystem AMEE derweil nicht umgesetzt, da sie für die Roboteranwendung irrelevant sind.

Demo Anwendung

Als Machbarkeitsnachweis und praktische Einführung in das Framework setzt die Beispielanwendung SarsaForDummies ein Grid World Szenario um, bei dem der Roboteragent durch Sarsa(λ) erlernt, innerhalb einer Rasterwelt von beliebiger Position aus den kürzesten Pfad zu einem festen Zielpunkt zu wählen. Der Nutzer kann die Parameter des Algorithmus frei einstellen, einzelne Roboterschritte oder eine ganze Episode bis zum Erreichen der Zielkoordinate durchführen und verfolgen, wie sich die Lernerfahrung auf die einzelnen Werte der Zustands-Wert-Funktion Q auswirkt und wie sich die Lohnberechtigung (eligibility) verteilt. SarsaForDummies nutzt hierfür die SarsaLambda Implementierung der RLLibrary und implementiert die Q-Funktion als zustandsindizierte Kollektion von jeweils acht reellen Werten für die Bewertung der acht möglichen Bewegungsrichtungen, die den Aktionsmöglichkeiten entsprechen. Für die übrigen Hilfsfunktionen reichen die Standardimplemen-tierungen der Library aus. Darüber hinaus ist, abgesehen von den visuellen Komponenten der Anwendung, nur die Umwelt implementiert, in der die Rasterwelt und die Bewegungskosten definiert sind. Die Anwendung bietet einem Einsteiger in die Materie somit eine visuelle, nachvollziehbare Erfahrung zum Verhalten des Sarsa(λ)-Algorithmus und dem Framework-Anwender eine praktische Orientierung für eigene Entwicklungen.

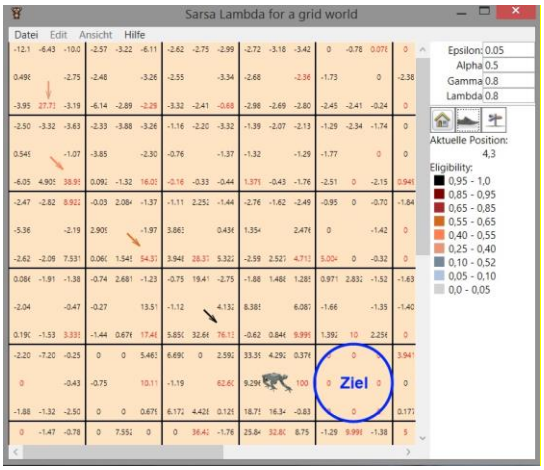


Abb. 3.15: Demo-Programm SarsaForDummies. Der Roboter kann horizontal, vertikal, diagonal zur Nachbarkachel gehen, jeweils mit Bewegungskosten quittiert. Die Zielkoordinate liefert eine positive Belohnung und beendet eine Lernepisode. Die beste Aktion in einem Feld ist rot hervorgehoben. Farbige Pfeile symbolsieren die Lohnberechtigung.

3.4 Zusammenfassung

Die Arbeit an LittleAMEE bietet ein Softwaretechnisches Konzept für eine verteilte Roboteranwendung mit Echtzeitanteilen in der Beinsteuerung und ereignisgesteuerten Modulen der höheren Logik wie Interaktion, Koordination oder Analyse und Lernen im Roboterverhalten. Robot Vision zeigte einen Ansatz zur Umgebungserfassung und Datenvorverarbeitung für den Lernalgorithmus. Letzter fällt in die Kategorie des Reinforcement Learning in der speziellen Ausprägung Sarsa(λ). Hierfür wurde ein neu Entwickeltes Framework und Programmiermodell vorgestellt, welches als Basis für die folgenden Experimente und spätere Umsetzung der Lernaufgabe, wie sie in **Abschnitt 4** vorgestellt wird, dient.

4 Modellierung der Lernaufgabe

Der Auftrag:
Erfasse den Boden der näheren Umgebung und ermittle Eigenschaften wie Steigung, Unebenheit, etc. Anhand dieser Eigenschaften soll eine Trittposition ausgewählt werden. Ob diese Trittposition gut ist, ist durch ausprobieren zu erlernen. Der ausgeführte Schritt wird beispielsweise anhand der Ausführbarkeit, ob er rutschfest war und wie die Kraftaufnahme und Drehmomententwicklung war, also anhand dessen, wie sich der Schritt anfühlt bewertet.

Als Lernalgorithmisches Gerüst ist die Wahl auf Reinforcement Learning gefallen, das ein fortwährendes, auf Erfahrungen basiertes Lernen ermöglicht und somit gegenüber klassischen Regelungs- oder Klassifizierungsaufgaben potentiell eine größere Flexibilität und ein breiteres Einsatzspektrum bietet. Der Kern und die erste Herausforderung bei dieser und aller vergleichbaren Lernaufgaben ist die Modellierung der selbigen (Abschnitt **4.1**). Dieser wird sich hier gewidmet, wobei bezüglich der äußeren Einflüsse einige generalisierende Annahmen und Abstraktionen vorgenommen werden indem die Umwelterfassung (Stereo Vision) und die „haptische“ Rückmeldung (Propriozeption) simuliert werden. Die Modellierung der Lernaufgabe erfolgt vorerst unter einem mehr theoretischen Aspekt als Eignungsstudie für die reale Anwendung und vergleichende Anwendungsfälle.

Eine besondere Herausforderung mit weitreichenden Konsequenzen für das Aufgabenmodell im Detail liegt im Umgang mit kontinuierlichen Wertebereichen, die unter anderem in Form von abstrakten Maßen der Bodenbeschaffenheitsanalyse zu erwarten sind. Daher beschäftigt sich die Modellierungsaufgabe auch mit dem Aspekt *Tile Coding* (Abschnitt **4.2**) und daraus resultierender Modellierungs-varianten.

4.1 Aufgaben Modell

Das grundlegende Konzept sieht vor, dass der Roboter eine visuelle Analyse der vor ihm liegenden Bodenbeschaffenheit vornimmt und in Rückkopplung mit einer propriozeptiven Bewegungsanalyse erlernt, was gute Trittpositionen sind. Die Definition einer guten Trittposition ist an sich schon nicht trivial, da verschiedenste Aspekte zu berücksichtigen und teilweise auch von der Roboterkonstruktion und der Laufart (Muster, dynamisch, statisch,…) abhängig sind. Beim Schreitroboter AMEE gilt es vordergründig, gute Schrittpositionen in dem Sinne zu finden, dass sie stabilen Halt bieten, die

Maschine weit voranbringen und kollisionsfrei erreichbar sind. Für die Evaluierung des Reinforcement Learnings für das gegebene Lernziel wird folgend ein vereinfachtes Simulationsszenario erstellt. Um die Komplexität der hiesigen Untersuchung zu reduzieren, wird das Problem darauf reduziert, möglichst ebene Untergründe als stabil zu interpretieren und ein ausgewogenes Verhältnis zwischen Unebenheit und Zielführung zu finden. Das Ziel sei dabei eine zu erreichende Wegkoordinate, die beispielsweise von einer übergeordneten Pfadplanung vorgegeben wird und als Episodenende für den RL-Algorithmus dient. Basis für die Entscheidung und Zustand im RL-Kontext sei ein simuliertes analytisches Abbild der Roboter-umgebung. Die Aufgabe ist, dass der Agent zu gegebenen Start- und Zielkoordinaten einen möglichst kurzen aber auch ebenen Pfad erlernt.

4.1.1 Unebenheitsmaß u

In der Realanwendung sollen die nähere Roboterumgebung mit Sensoren wie Laserscanner, Stereo- oder Time-of-Flight Kamera abgetastet und geländespezifische Merkmale extrahiert werden. In diesem Szenario sei – ohne Einschränkung der allgemeinen Verfahrenseigenschaften – ein fiktives Unebenheitsmaß von Null bis Eins für die Bodenbeschaffenheit gegeben, wobei Null für absolut glatte und Eins für maximal raue Oberflächen stehe. Eine Umgebung lässt sich dann durch eine Funktion simulieren, die zu einer gegebenen reellen Koordinaten (x, y) in der Ebene das Unebenheitsmaß $\mathrm{u} \in [0, 1]$ wiedergibt:

$$welt\colon \mathbb{R}^2 \rightarrow [0,1] \subset \mathbb{R}$$

Im Anhang (**A.3**) sind einige mögliche Ausprägungen der Funktion abgebildet.

4.1.2 Vergeltung (Reward)

Begriffsbestimmung

Das Alleinstellungsmerkmal im Bestärkenden Lernen ist die Belohnung oder Bestrafung einer Aktion, in der englischen Fachliteratur zusammenfassend mit *reward* bezeichnet. Hier soll dafür der Begriff der *Vergeltung* verwendetet werden. Im allgemeinen Sprachgebrauch, wird der Begriff zwar mit Strafe oder Sühne verbunden. Im eigentlichen Wortsinne bezeichnet die Vergeltung aber eher eine Gegenleistung nach dem Gleichheitsprinzip für erbrachte Dienste oder beim Tausch von Waren und kann somit als Oberbegriff für Bestrafung und Belohnung verwendet werden.

Vergeltungstypen

Im Reinforcement Learning kann man in der Regel zwischen zwei Prinzipien der Vergeltung unterscheiden. Bei der ersten Variante gibt es nur am Episodenende, beim Erreichen des Aufgabenziels, eine Belohnung. Die einzelnen Schritte werden hingegen nicht vergolten. Dieses Prinzip findet sich häufig in spielartigen Aufgaben, bei denen allein der Sieg entscheidend ist und Auskunft über die Qualität vorangegangener Spielstände liefert. Die andere Art der Vergeltungsmodellierung

bestraft jeden Zwischenschritt mehr oder minder stark und gleicht dies gegebenenfalls mit einer hohen Belohnung bei Erreichen des Ziels aus. Ein typisches Beispiel dafür sind Navigationsaufgaben, bei denen jeder Schritt mit Wegkosten verbunden ist und es gilt, die Kosten zu minimieren. Die zweite Variante liefert zusätzliche Informationen durch die Vergeltung nach jeder Aktion. Sherstov et. Al. [She05] und andere haben gezeigt, dass die Modellierung ergänzenden Wissens in die Vergeltung zu einem schnelleren Lernerfolg führt. Allerdings sollte man dabei beachten, dadurch keine zu strengen Vorgaben zu machen, wie ein Ziel zu erreichen ist. Denn eine Stärke des Bestärkenden Lernens ist gerade das Aufspüren unbekannter Lösungswege. Je mehr domänenspezifisches Wissen sich in die Vergeltung modellieren lässt, desto eher ist dies ein Indiz dafür, dass sich die Aufgabe effizienter mit einem Regel- oder Wissensbasierten Verfahren als mit maschinellem Lernen bewältigen lässt.

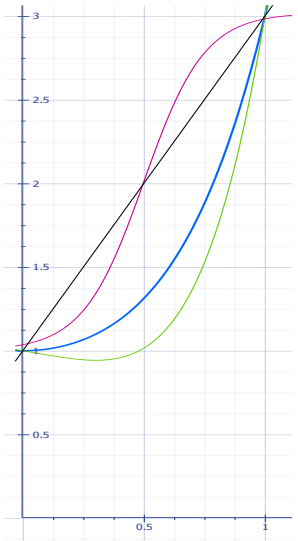


Abb. 4.1: Vergeltungsfunktionen in Abhängigkeit von der Unebenheit

Aufgabenspezifische Anforderungen

Unser Vierbeiner muss hier einen möglichst direkten Weg zu einem Etappenziel finden und dabei Unebenheiten vermeiden. Um dies zu erreichen, bietet sich die informative Vergeltung an. Sie soll weite Schritte in Richtung Ziel begünstigen und zusätzlich Unebenheit bestrafen. Das Ziel sei dabei eine zu erreichende Wegkoordinate, die beispielsweise von einer übergeordneten Pfadplanung vorgegeben wird und als Episodenende für den RL-Algorithmus dient. Eine Wegkoordinate zu erreichen, ist eine Navigationsaufgabe, bei der man das bewerte Konzept der Schrittkosten übernehmen kann. Ein einzelner Schritt kostet eine (Energie-) Einheit, unabhängig von der Schrittlänge. Man kann das damit begründen, dass auf längeren Wegstrecken die Anzahl der Schritte wohl stärker ins Gewicht fällt, als die vermutlich eher geringen Kostenunterschiede zwischen kurzen und langen Schritten. Ein direkter Weg wird dann implizit dadurch begünstigt, dass er geringere Kosten verursacht oder im RL-Kontext ausgedrückt, in der Summe weniger bestraft wird. Ein unebener Untergrund führt dazu, dass das Laufsystem vermehrt nachkorrigieren muss, was in erhöhte (Energie- / Zeit-) Kosten resultiert. Dies kann dadurch simuliert werden, dass das Unebenheitsmaß an der Trittposition den Schrittkosten hinzuaddiert wird. Mit einem Skalierungsfaktor für die Unebenheitskosten erhält man eine Stellgröße, um zu bestimmen, in welchem Maße Unebenheit akzeptiert wird. Eine extra Belohnung bei Erreichen des Ziels ist im Prinzip nicht nötig, da eine Gewinnmaximierung, wie es der RL-Algorithmus anstrebt, gleichbedeutend mit einer Kostenminimierung ist. Letztlich sollte sich ein Verhalten einstellen, dass den Schreitroboter zügig zum Ziel führt und dabei zu große Unebenheiten im Bogen umgeht.

Die konkrete Ausprägung der Vergeltungsfunktion beeinflusst direkt das Verhalten, dass erlernt wird. **Abb. 4.1** zeigt einige Beispiele bei denen die Grundkosten für einen Schritt jeweils Eins sind und die Unebenheit mit bis zu Zwei weiteren Einheiten bestraft wird:

- $-r = 2u + 1$
  Hier wird davon ausgegangen, dass sich die Zusatzkosten proportional zur Unebenheit verhalten.
- $-r = \frac{2}{1+e^{-8(u-0.5)}} + 1$
  Mit dieser Sigmoidfunktion steigen die Zusatzkosten zunächst langsam und dann immer stärker an, bis die Unebenheit so groß ist, dass ein noch etwas raueres Gelände nicht viel schlimmer ist.
- $-r = u^4 + u^2 + 1$
  Eine exponentielle Kostenexplosion mit zunehmender Unebenheit.
- $-r = \frac{9}{4}u^4 - \frac{1}{4}u + 1$
  Hier wird modelliert, dass ein wenig raues Gelände sogar vorteilhaft ist, weil es vielleicht mehr Halt bietet.

Selbstverständlich bedarf es eigentlich keines Lernalgorithmus, wenn man ein derartiges Regelwerk nutzen kann. Aber die hier beschriebene Vergeltungsfunktion simuliert auch lediglich die Rückmeldung, die im realen System durch Propriozeption erfolgt.

4.1.3 Umwelt und Aktion

Die Roboterwelt, bzw. Umgebung im RL-Kontext, besteht hier aus Unebenheitsmaßen in einer egozentrische Karte mit der Roboterposition als Mittelpunkt. Es wird an dieser Stelle ein radialer Scan der „Realwelt“ angenommen und die Karte daher ebenfalls radial mit einer Unterteilung in Sektoren und Spuren umgesetzt. Ein Kartenabschnitt ist dann durch Spur Sp und Sektor Se definiert, was eine erste Diskretisierung des zweidimensionalen Zustandsraums der Ebenenpositionen umsetzt. Eine Aktion ist in diesem Kontext die Wahl einer Trittposition, also eine Koordinate in der Ebene. Da es den RL-Algorithmus ressourcenschonender, performanter und eleganter macht, wenn auch die Menge der möglichen Aktionen diskretisiert und endlich ist, bietet es sich im Zusammenhang mit der Karte an, die Koordinatenwahl darauf zu vereinfachen, den Fuß auf die jeweilige Mitte des gewählten Abschnitts zu setzen, relativ zum Kartenmittelpunkt. Dadurch ist auch der eigentlich kontinuierliche Aktionsraum in eine endliche Auswahl aufgeteilt. Je Abschnitt gibt es eine feste, gleiche Anzahl von Messpunkten, deren Unebenheitswerte gemittelt für den jeweiligen Abschnitt gespeichert werden (**Abb. 4.3**). Die Abschnitte auf den äußeren Spuren haben dann eine geringere Auflösung und geben einen gröberen Mittelwert der *realen* Bodenbeschaffenheit wieder, als die auf den inneren Spuren, was künftigen, echten Bedingungen Rechnung trägt.

Der Zustandsraum setzt sich somit zunächst aus den diskreten Abschnitten und den darin gespeicherten beschränkten aber kontinuierlichen Unebenheitswerten zusammen. Da hier aber nicht allein die Unebenheit als Entscheidungsgrundlage über gute Trittpositionen dient, sondern auch das Voranschreiten, werden zusätzlich Informationen über zumindest die grobe Zielrichtung und die

Entfernung zum Ziel benötigt. Ohne diese Kenntnis würde das Erlernen der RL-Bewertungsfunktion nicht funktionieren. Um das zu verdeutlichen erinnere man sich daran, dass die Bewertung eines Zustands-Aktions-Paares die von diesem Zustand aus zu erwartende Gesamtvergeltung bis zum Erreichen des Episodenziels wiedergibt, wenn man eine bestimmte Aktion ausführt. Man stelle sich nun beispielsweise eine ebene Halle vor. Die Unebenheitsmaße der Kartenabschnitte würden dann fast immer alle den gleichen Wert anzeigen und ohne eine Entfernungsangabe würde sich ein Zustand weit entfernt vom Ziel nicht von einem nahen Zustand unterscheiden. Trotzdem entstünden unterschiedliche Kosten und derselbe Zustand würde verschieden bewertet werden. Es wäre unmöglich, eine zuverlässige Bewertung zu erlernen. Das gleiche Problem tritt ohne Angabe einer Zielrichtung auf. Denn in der fiktiven Halle haben alle Orte auf einem konzentrischen Kreis um das Ziel dieselbe Entfernung und dieselben Unebenheitsmaße. Jedoch verursachen immer andere Aktionen, nämlich die zum Zentrum führenden, die geringsten Kosten. Ohne Richtungsangabe in der Zustandsbeschreibung würde dasselbe Zustands-Aktions-Paar folglich unterschiedlich bewertet werden müssen.

In **Abb. 4.2** ist die Roboterkarte mit ihren Abschnitten auf eine schematisierte, beispielhafte Realwelt projiziert. Dabei zeigt die Realweltkarte die rohen Unebenheitsmaße an, je dunkler desto ebener. Für den oberen Abschnitt der Roboterkarte sind die Messpunkte angedeutet, deren gemittelte Messwerte das Unebenheitsmaß für den Abschnitt ergeben. Die Kompassrose zeigt die Richtung des Ziels an, die Distanz ist auf der Abbildung nicht wiedergegeben.

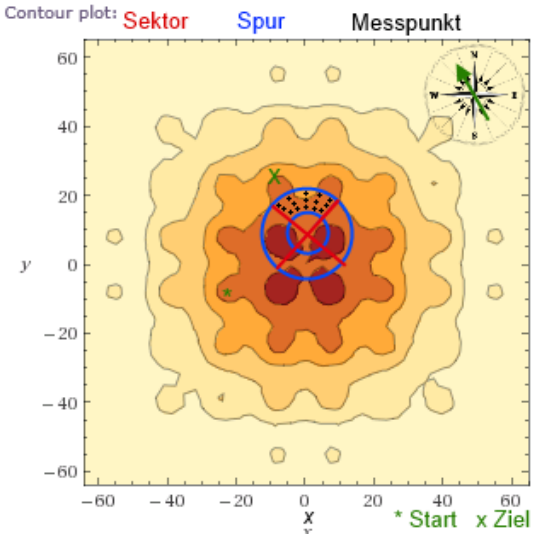


Abb. 4.2: Schematische Darstellung der Unebenheiten und deren Abbildung auf die radiale Agentenkarte. Je dunkler die Farbe, desto ebener ist der Untergrund.

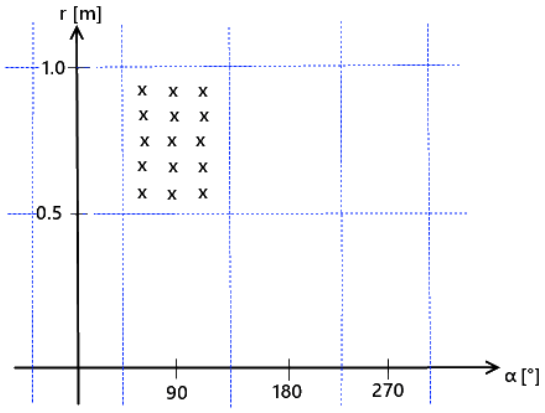


Abb. 4.3: Partitionierung von Messpunkten in Abschnitte. Messwerte werden gemittelt.

Für eine derartige Welt gibt es zwei Herausforderungen zu handhaben. Die kontinuierlichen Werte führen zu einem theoretisch unendlich großen Zustandsraum und die Welt ist aus Robotersicht nicht statisch, wie in dem eingehend beschriebenen Gridworld Beispiel, sondern veränderlich. Der Roboter bewegt sich nicht innerhalb der Karte, sondern die Karte bewegt sich mit dem Roboter und wird nach jedem Schritt aktualisiert. Es sei an dieser Stelle explizit hervorgehoben, dass es sich hier nicht um eine klassische, reine Navigationsaufgabe handelt, bei der die Verortung innerhalb eines Umweltmodells in Form einer Karte dem aktuellen Zustand entspricht und eine Aktion gleich der Bewegung von einem Kartenpunkt auf den nächsten ist. Vielmehr ist die Umwelt selbst, beziehungsweise ihre Struktur, der aktuelle Zustand und eine Aktion ist zwar auch hier eine Fortbewegung, führt aber nicht zu einer neuen Position sondern zu einer neuen Umgebungsstruktur, beziehungsweise einem neuen Kartenmodell.

Zusammenfassend besteht ein Zustand in diesem RL-Modell aus der in Sektoren $\mathcal{S}e$ und Spuren $\mathcal{S}p$ untergliederten Karte, die zusammen einzelne Abschnitte $\mathcal{A}b$ bilden, wobei jeder Abschnitt mit einem gemittelten Unebenheitswert $\mathcal{U}$ versehen ist. Des Weiteren sind noch die Zielrichtung $\mathcal{Z}$ und die Zieldistanz $\mathcal{D}$ bekannt. Die Aktionen $\mathcal{A}$ sind gleichbedeutend mit einem Schritt auf Abschnitt $\mathcal{A}b_i$.

$$\boldsymbol{state}\ \boldsymbol{s_t} = (\boldsymbol{u_t}(\mathcal{A}\boldsymbol{b_1}), \dots, \boldsymbol{u_t}(\mathcal{A}\boldsymbol{b_N}), \boldsymbol{\mathcal{Z}_t}, \boldsymbol{\mathcal{D}_t}) \quad \boldsymbol{action}\ \boldsymbol{a_t} \in \mathcal{A}\boldsymbol{b}$$

Das Mitteln von Unebenheitsmaßen je Abschnitt ist eine vorverarbeitete Merkmalsgewinnung[20]. Diese im Bereich der Mustererkennung oder dem maschinellen Lernen häufig genutzte Vorgehensweise stellt bereits eine erste Partitionierung der Flächenpunkte und Unebenheitsverteilung dar (vgl. **Abb. 4.3**). Sie liefert uns eine in Abschnitte diskretisierte Ebene und ferner eine in Sektoren unterteilte Zielrichtung. Durch das Vereinfachen der Aktion darauf, den Fuß auf die jeweilige Mitte des gewählten Abschnitts zu setzen, ist auch der eigentlich kontinuierliche Aktionsraum auf eine endliche Auswahl aufgeteilt. Es bleibt noch das beschränkte aber kontinuierliche Unebenheitsmaß sowie die Zieldistanz, auf welche das Tile Coding Verfahren angewendet werden. Ferner muss bei der Tilings Organisation noch bestimmt werden, ob Abhängigkeiten zwischen den Merkmalen modelliert werden.

4.2 Tile Coding

In **2.3.4.1** wurde das Tile Coding als Verfahren zur Abbildung des kontinuierlichen Zustandsraums in einen sogenannten Merkmalsvektor $\vec{f}$ mit zugehörigen Gewichten $\vec{w}$ vorgestellt, um so den theoretisch unendlich vielen Zustandsausprägungen Herr zu werden und sich einer mit $\vec{w}$ parametrisierten Zustandsbewertungsfunktion zu nähern. Der folgende Abschnitt stellt eine Variante zur automatisierten Erstellung von Tilings und der Berechnung der Tile-Nummer vor. Diese hier MultiTiling genannte Methode wird auf jede Dimension des Eingangsvektorraums einzeln angewandt,

[20] Engl.: feature extraction by pre-processing

da die Kombination der Merkmalsdimensionen eine gesonderte Betrachtung verlangt, die in **4.2.2** im Allgemeinen und in **4.2.3** für das Aufgabenmodell der Unebenheitskarte im Speziellen erfolgt.

4.2.1 Tilings Berechnung

Sofern eine systematische, berechenbare Tilings-Anordnung verwendet wird, benötigt eine Implementierung nur minimale Speicheranforderungen und die Merkmals- oder Tile-Indizes lassen sich zu gegebenem Zustandsvektor meist mit geringem Rechenaufwand ermitteln.

Hashing

Sutton und andere schlagen hierfür ein Hashing-Verfahren vor[21] (näheres in [Sut98]), dessen Vorteil hauptsächlich für sehr große Wertebereiche oder einen hochdimensionalen Zustandsraum und einer daraus resultierenden sehr großen Gesamtzahl an Tiles zu tragen kommt. Das Verfahren bildet mehrere Tile-Indizes auf einen kleineren Bereich ab, der vorgegeben werden kann. So kann zumindest im ersten Schritt dieser Merkmalsgewinnung eine kleine Auflösung beibehalten werden, auch wenn dies normalerweise etwa zu mehreren Millionen Tiles führe und entsprechend viele Gewichte w im Speicher gehalten werden müssten. Die ursprünglichen Tiles können hier als lose verbundene Sub-Tiles der dann beispielsweise nur noch ein paar hundert Hashing-Tiles betrachtet werden. Ein Nachteil des Hashing-Verfahrens sind Kollisionen, bei denen weit entfernte Tiles oder ganze Gruppen von Tiles, die eigentlich nicht miteinander im Zusammenhang stehen, auf denselben Hashwert abgebildet werden. Darüber hinaus verwendet die von Sutton vorgeschlagene und häufig genutzte Umsetzung nur die konjunktive Merkmalskombination (siehe **4.2.2**) und berücksichtigt nicht die speicher-effizienteren Kombinationsarten. Meist lassen sich auch die erzielten Hashwerte nur schwer nachvollziehen, was für den Lernalgorithmus im Allgemeinen zwar irrelevant aber für die Verständlichkeit des Gesamtverfahrens nachteilig ist.

In der wohl überwiegenden Mehrheit der realen Anwendungsfälle, insbesondere der Robotik, kommen übergroße Wertebereiche, für die der Hashing-Algorithmus ausgelegt ist, nicht vor. Sensordaten haben in der Regel konstruktions- und verfahrensbedingt vorgegebene Mess-genauigkeiten und -bereiche. Bezogen auf eine spezifische Aufgabe ist auch der Bereich von Interesse oft eingeschränkt und extreme Werte können ausgeblendet oder zusammengefasst werden. Durch geeignete Vorverarbeitung der Systemeingangsdaten können oft Merkmale extrahiert werden, die als geringer dimensionierter Eingangsvektor für die Lernkomponente dienen und somit das zweite Argument für Hashing, den Fluch der Dimensionalität[22], abschwächt. So entsteht die Unebenheitskarte beispielsweise aus einem Tiefenbild mit 640 x 480 Pixeln. Eigentlich ist jede Bildausprägung ein Punkt

[21] Sutton bietet eine online frei verfügbare fertige Implementierung an: http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/RLtoolkit/tiles.html (letzter Zugriff: 24.04.2014)

[22] Der Begriff *Curse of Dimensionality* wurde von Richard E. Bellman geprägt: *Adaptive control processes: a guided tour*, 1961, Princton University Press

in einem Raum mit so vielen Dimensionen wie Pixeln und einem theoretisch unendlichen reellen Wertebereich je Pixel. Aber einerseits erfasst der Sensor nur beschränkte Entfernungen von vielleicht 50 cm bis 5 Meter in maximal millimetergenauer Auflösung, alles darüber oder darunter sind unbrauchbare Daten. Außerdem interessieren nur abschnittsweise vorverarbeitete Maße an Unebenheiten, sagen wir zwanzig Stück, womit die Bilder nur in entsprechend zwanzig Freiheitsgeraden variieren, die im eingeschränkten Wertebereich von Null bis Eins liegen. Die eigentliche Herausforderung liegt somit meist in der domänenspezifischen Modellierung der Aufgabe, geeigneter Vorverarbeitung und der Tilings-Organisation, sodass der Merkmalsraum speichereffizient und gleichzeitig die Approximationsgenauigkeit hinreichend gut ist. Für die eingehend in **4.1** beschriebene Unebenheitskarte betrachtet der Abschnitt **4.2.3** die mögliche Größe des Merkmalraums genauer. An dieser Stelle sei lediglich vorweggenommen, dass sich für die hier betrachtete Aufgabe Tilings-Organisationen finden lassen, dessen Komplexität sich in einem akzeptablen Rahmen bewegen und deswegen eine leichter nachvollziehbare und simplere Methode zur Tilings-Berechnung in Frage kommt.

MultiTiling

Eine naheliegende wie einfache und für die späteren Untersuchungen hier umgesetzte Variante, eine kontinuierliche Dimension zu unterteilen ist es, gleichgroße Intervalle als Tiles zu nehmen. Ist zudem der Wertebereich beschränkt und die Anzahl der Tiles oder die Auflösung bekannt, lassen sich alle notwendigen Informationen wie Tile-Grenzen, -Indizes und -Mittelpunkte zu einem gegebenen Datum leicht errechnen. Nach der Vorstellung des MultiTiling Ansatzes erfolgt eine Betrachtung der Parametrisierung des Verfahrens, die für adaptive Tilings-Anordnungen relevant ist.

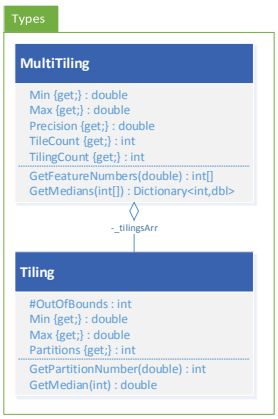


Abb. 4.4: MultiTiling in RLLibrary .NET

Vorerst wird je eine einzelne Dimension des Zustandsraums betrachtet und darauf die Kachelkodierung angewendet, da bei einer Kombination mehrerer Dimensionen die Tile-Nummerierung abhängig von der Tilings-Organisation ist, die in **4.2.3** noch betrachtet wird. Bei der Erstellung eines **`MultiTiling`** werden die Anzahl der Tilings (kurz **#T**), der Wertebereich $\mathbf{r}$ (engl.: *range*) sowie die gewünschte Auflösung $\boldsymbol{\delta} = \frac{\boldsymbol{r}}{\boldsymbol{n}}$ vorgegeben. Bei einem einzigen Tiling entstehen so $\mathbf{n}$ Tiles mit jeweils einer Länge von $\mathbf{tLen} = \boldsymbol{\delta}$ und mit Null beginnend durchnummeriert. Zu gegebenem Eingangswert x errechnet sich die Tile-Nummer dann einfach aus $idx = \left\lfloor \frac{x - \min(r)}{tLen} \right\rfloor$, wobei $\min(\mathrm{r})$ die Untergrenze des Wertebereichs und $\lfloor x \rfloor$ eine ganzzahlige Abrundung von x bezeichnet. Eingangswerte außerhalb des Wertebereichs liefern alle einen „*Out of Range*“-Index.

Bei gleichem δ aber zwei Tilings, wird die gewünschte Auflösung erzielt, in dem die einzelne Tile-Länge auf $\mathrm{tLen} = 2 \cdot \boldsymbol{\delta}$ verdoppelt und der Beginn des zweiten Tilings gegenüber dem ersten um δ versetzt werden. Für drei Tilings ist dann $\mathrm{tLen} = 3 \cdot \boldsymbol{\delta}$ und der Versatz des dritten Tilings ist -δ (vgl. **Abb. 4.5**).

Aus den Vorgaben $\#\mathbf{T}, \mathbf{r}, \boldsymbol{\delta}$ lassen sich so alle notwendigen Daten zur Erstellung der einzelnen Tilings errechnen:

Gegeben sind $Anzahl\ Tilings := \#T, Wertebereich := r, Auflösung\ \delta$

$$Länge\ eines\ Tiles \ := \mathbf{tLen} = \#\mathrm{T} \cdot \delta \qquad \textbf{Glg. 4.1}$$

$$Anz.Tiles\ je\ Tiling := \ \#\mathbf{tT} = \left\lfloor \frac{\mathrm{r}}{\mathrm{tLen}} \right\rfloor \qquad \textbf{Glg. 4.2}$$

$$Gesamtzahl\ Tiles \ := \ \ \#\mathbf{t} = \#\mathrm{T} \cdot \#\mathrm{tT} \qquad \textbf{Glg. 4.3}$$

Die Tilings werden abwechselnd rechts und links verschoben, sodass sich der Versatz der einzelnen Tilings-Bereiche folgend ergibt:

$$Versatz\ des\ i'ten\ Tiling := \mathbf{offset_i} \ = (-1)^{i+1} \cdot \left\lceil \frac{i}{2} \right\rceil \cdot \delta \quad | \, 0 \leq i < \#T \qquad \textbf{Glg. 4.4}$$

Für das i‘te Tiling lässt sich zu gegebenem Wert x die Tile-Nummer mittels tLen und der Untergrenze des i‘ten Tilings-Wertebereichs Min_i berechnen:

$$\textbf{Tile-ID} = \ i \cdot \#tT + \left\lfloor \frac{x - Min_i}{tLen} \right\rfloor \quad | \, 0 \leq i < \#T \qquad \textbf{Glg. 4.5}$$

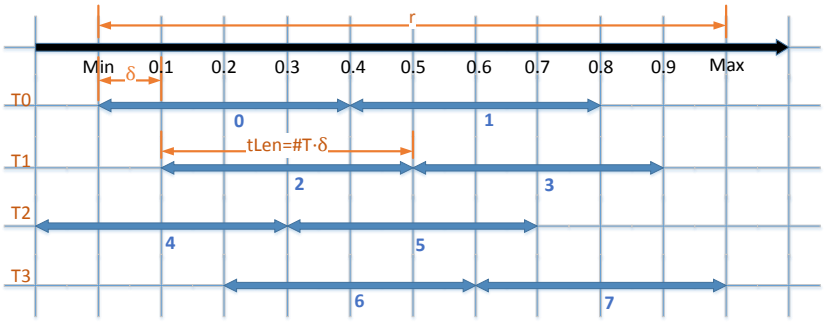


Abb. 4.5: MultiTiling. Automatische Tilings-Generierung für eine Dimension.

Sollen dynamische Tilings, wie in **2.3.4.1** angedeutet, unterstütz werden, müssen die Anzahl Tilings und die Gesamtzahl der Tiles oder die gewünschte Auflösung aufeinander abgestimmt sein, damit eine sinnvolle Aufteilung erfolgen kann, sodass die Gesamtanzahl Tiles – und somit die Zahl der Gewichte w – konstant bleibt. Wenn also die Tilings-Anordnung dynamisch angepasst wird, sollten Bedingungen an die Wahl der Auflösung δ und Tilingszahl #T gesetzt werden, die das MultiTiling per se noch nicht erfüllt. Aus der bisherigen Betrachtung und den oben aufgestellten Gleichungen **Glg. 4.1** bis **Glg. 4.5** ergeben sich die folgend erläuterten Tiles-Verhältnisse **1** bis **3**, bezüglich der Voraussetzungen für adaptives MultiTiling.

Die Ursache dieser gesonderten Betrachtung für adaptives MultiTiling liegt in **Glg. 4.2**, die zeigt, dass die Anzahl Tiles für ein Tiling unter Umständen abgerundet ist und folglich die Gesamtzahl Tiles #t bei gleicher Auflösung δ unterschiedlich ausfallen kann, je nachdem, wie viele Tilings erzeugt werden. Um dies zu verdeutlichen sei angenommen, dass die gewünschte Auflösung ein n'tel des Wertebereichs r sein soll und m Tilings erzeugt werden. Dann gilt:

Tiles-Verhältnis 1: Anzahl Tiles im MultiTiling bei freier Wahl von δ und #T

$$\delta = \frac{r}{n} \mid n \in \mathbb{N}, r \in \mathbb{R} \quad \text{und} \quad \#T = m \mid m \in \mathbb{N}, m \leq n$$

$$\Rightarrow tLen = \delta \cdot \#T = \frac{r \cdot m}{n} \Rightarrow \#tT = \left\lfloor r \cdot \frac{n}{rm} \right\rfloor = \left\lfloor \frac{n}{m} \right\rfloor \Rightarrow \#\boldsymbol{t} = \boldsymbol{m} \cdot \left\lfloor \frac{\boldsymbol{n}}{\boldsymbol{m}} \right\rfloor \leq \boldsymbol{n}$$

Nun soll für eine dynamische Tilings-Anzahl die Gesamtmenge an Tiles #t (und somit der Gewichte w) konstant bleiben, so muss man Bedingungen an die Konfiguration setzen: Wenn die Tilings-Anzahl m stets aus einer ganzzahligen Division von n hervorgeht, dann ist die Gesamtzahl an Tiles unabhängig von der Tilings-Anzahl immer n.

Tiles-Verhältnis 2: feste Anzahl Tiles im MultiTiling für bedingtes δ und #T

Seien gegeben: $r \in \mathbb{R}$ und $n, m \in \mathbb{N} \mid \exists p \in \mathbb{N} : m = \frac{n}{p}, m \leq n$

Dann sind $\delta = \frac{r}{n}$ und $\#T = m$

$$\Rightarrow tLen = \frac{r \cdot m}{n} = \frac{r}{p} \Rightarrow \#tT = \left\lfloor \frac{n}{m} \right\rfloor = p \Rightarrow \#\boldsymbol{t} = \boldsymbol{m} \cdot \boldsymbol{p} = \boldsymbol{n}$$

Um die Konfiguration von dynamischen MultiTilings zu vereinfachen, ist es ratsam, die Vorbedingungen an einer ganzzahligen Basis b, beispielsweise zwei, festzumachen. Beachtet man weiterhin $m \leq n$ für festes n, kann man den Divisor der Auflösung zu b^n setzen und die Tilings-Anzahl #T zu b^m wählen. Dann ist für jede Tilings-Anzahl #T die Gesamtzahl an Tiles gleich b^n.

Tiles-Verhältnis 3: Variante von Lemma 2 für feste Anzahl Tiles im MultiTiling

Seien gegeben: $r \in \mathbb{R}$ und $n, b \in \mathbb{N}$. Dann sind:

$$\delta = \frac{r}{b^n} \; und \; \#T = b^m \mid m \in \{0, \dots, n\}_{\mathbb{N}}$$

$$\Rightarrow tLen = r \cdot b^{m-n} \overset{m \leq n}{\Longrightarrow} tLen = \frac{r}{b^{n-m}} \Rightarrow \#tT = b^{n-m} \Rightarrow \#\boldsymbol{t} = \boldsymbol{b^n}$$

Das MultiTiling ist eine ressourcenschonende Art, das Tile-Coding zu automatisieren und bei geeigneter Kombination der Auflösung δ und Tilings-Anzahl #T gemäß **Tiles-Verhältnis 2** oder **Tiles-Verhältnis 3** für eine dynamische Tilings-Konfiguration verwendbar. Eine Generalisierung des MultiTiling für einen mehrdimensionalen Zustandsvektor ist aber nicht möglich, da die Gesamtgröße des Merkmalraums dann von der Kombinationsart der Merkmalsdimensionen abhängt. Daher ist es sinnvoller, für die jeweilige Aufgabenstellung und ihre Modellierung eine individuelle Implementierung

umzusetzen[23]. Die RLLibrary bietet das **`MultiTiling`** und ein einzelnes **`Tiling`** als Datentyp an. Eine Überprüfung der Vorbedingungen nach Lemma 2 oder 3 für dynamische Tilings wird aber nicht durchgeführt. Dies obliegt dem Anwender des Frameworks ebenso, wie die Kombination mehrerer **`MultiTiling`** zu einer multidimensionalen Datenstruktur.

4.2.2 Vektorgröße bei mehrdimensionalen Tilings

An dieser Stelle wird der allgemeine Zusammenhang zwischen Größe des Merkmalsvektors $\vec{f}$, beziehungsweise des gleichgroßen Gewichtungsvektors $\vec{w}$, und mehrdimensionaler Tilings bei einer Kombination von Merkmalen betrachtet, die Abhängigkeiten zwischen ihnen repräsentieren. Ein Verständnis darüber ist für die Modellierung der Lernaufgabe und insbesondere ihrer Komplexität relevant. Es hilft bei der notwendigen Abwägung von Ressourcen und Approximationsgenauigkeit. Zusammenfassend gibt es zwei grundsätzliche Betrachtungsweisen der Merkmale. Entweder sie werden als voneinander unabhängig oder abhängig behandelt, was sich jeweils sowohl auf die Größe des Merkmalraums als auch auf die Approximationsgenauigkeit des Verfahrens auswirkt.

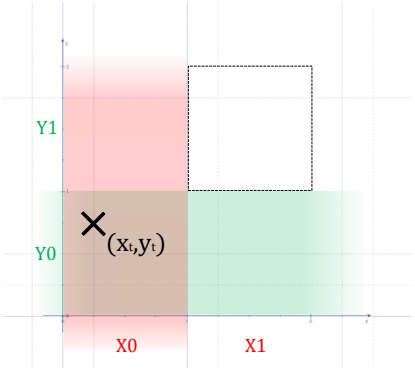


Abb. 4.6: Lineare Unabhängigkeit der Merkmale X und Y entspricht einer Disjunktion

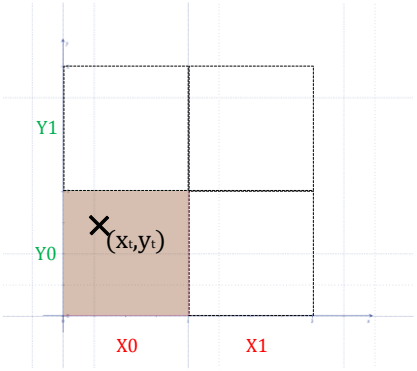


Abb. 4.7: Linear abhängige Kombination von X und Y entspricht einer Konjunktion

Je Dimension des Zustandsraums bestimmt sich die Gesamtzahl der Merkmale aus der Summe der Tiles je Tiling. Angenommen man habe die zwei Zustandsparameter X und Y, dann bezeichne $|\cdot|$ die Gesamtzahl der Merkmale von X, beziehungsweise Y. Ohne relationale Zusammenhänge zwischen den Parametern ergibt sich dann die Dimension des Merkmalvektors aus der Summe aller Merkmale mit einer Komplexität von O(n):

$$\dim(\vec{f}) = |X| + |Y|$$

Ein Zustand $s_t = (x_t \in X_0, y_t \in Y_0)$ zum Zeitpunkt t, gehört sowohl zum Intervall X_0 als auch zum Intervall Y_0 (hat die Merkmale X_0, Y_0). Die Lernerfahrung, durch verändern der beiden entsprechenden

[23] Die bei Sutton angebotene Tiling Software, basierend auf einem Hashing-Verfahren, behandelt zwar mehrdimensionale Eingangsvektoren, berücksichtigt dabei aber nur eine Kombinationsart der Dimensionen, vermutlich die Konjunktion (siehe **4.2.2**)

Gewichte beeinflusst alle Zustände die im Intervall X_0 oder Y_0 liegen (siehe **Abb. 4.6**). Die Generalisierungsbreite der Lernerfahrung über alle Eingangsdimensionen entspricht somit einer Disjunktion der Merkmalsintervalle. Dies hat ein hohes Maß an Generalisierung auf Kosten der Approximationsgenauigkeit. Steht X beispielsweise für die Winkelstellung eines Gelenks und Y für die Winkelgeschwindigkeit, so beeinflusst die Erfahrung im aktuellen Lernschritt alle Situationen, bei denen der Winkel im selben Intervall liegt, sagen wir „positiv kleiner Winkel“, unabhängig von der aktuellen Geschwindigkeit. Gleichzeitig beeinflusst die aktuelle Geschwindigkeit, sagen wir „schnell rechtsdrehend“, alle Zustände mit ähnlichem Tempo, unabhängig von der Winkelstellung. Für die Zustände, die auch in derselben Schnittmenge liegen, ist die Lernerfahrung aber am stärksten ausgeprägt, da sich hier die Gewichte der einzelnen Dimensionen aufsummieren.

Fügt man partiell Relationen $X \times Y$, beispielsweise der Art $x_t < y_t$ oder $2y_t \leq x_t \leq 4y_t$, als erweitertes Domänenwissen hinzu, ändert dies noch nichts an der Komplexität, da die Anzahl solcher Relationen $|X \times Y|$ lediglich hinzu addiert wird, die Relation quasi eine zusätzliche Dimension darstellt:

$$\dim(\vec{f}) = |X| + |Y| + |X \times Y|$$

Besteht eine generelle Abhängigkeit zwischen X und Y, muss jede Intervallkombination betrachtet werden. Ein Zustand $s_t = (x_t \in X_0, y_t \in Y_0)$ zum Zeitpunkt t gehört zur Schnittmenge der Intervalle X_0, Y_0 (hat das eine Merkmal $XY_{0,0}$) und der Einfluss der Lernerfahrung beschränkt sich auf die Zustände, die sowohl in X_0 als auch in Y_0 liegen (siehe **Abb. 4.7**). Die Generalisierungsbreite der Lernerfahrung über alle Eingangsdimensionen entspricht hier einer Konjunktion der Merkmalsintervalle, was zu einer höheren Approximationsgenauigkeit auf Kosten der Komplexität von $O(n^2)$ geht:

$$\dim(\vec{f}) = |X| \cdot |Y|$$

Behauptung 1

Auch eine disjunktive Kombination kann, sofern die entsprechenden Merkmale semantisch unabhängig sind, zu einer hinreichend guten Approximation der Schätzfunktion führen.

Letztere ist die in der Fachliteratur gemeinhin weitverbreitetste Verknüpfung von Merkmalsdimensionen aber mit Nichten die einzige. Das Hashing-Verfahren ist für die Handhabung dieser konjunktiven Kombination und des damit einhergehenden Fluchs der Dimensionalität ausgelegt. Abhängig von den Eingangsgrößen und der Aufgabenstellung kann jedoch auch die disjunktive Kombination mit ihrem kleineren Merkmalsraum zu hinreichend guten Approximationen führen, wobei dann das einfachere MultiTiling vorzuziehen wäre. Der Einfluss auf die Generalisierungseigenschaft bei disjunktiver oder konjunktiver Kombination ist vergleichbar mit der Wahl der Tilings-Anzahl je Dimension. Je höher die Generalisierungsbreite ist, desto schneller verbreitet sich die

Erfahrung, was während der ersten Lernepisoden zu einer beschleunigten Approximation hin zur optimalen Bewertungsfunktion führt, sich aber nachteilig auf die Spezialisierung des Wissens in späteren Lernepisoden auswirkt (vgl. [She05]). Aber während sich eine adaptive Tilings-Anzahl je Dimension umsetzen lässt, ist dies bei der Frage der Merkmalskombination nicht unbedingt möglich. Letztere verändert immer die Anzahl der Gewichte und die sind es, die das erlernte Wissen speichern. Tiles, beziehungsweise ihre Gewichte zu reduzieren, wäre gleichbedeutend damit, Wissen zu löschen. Die grundsätzliche Tilings Organisation, also die Kombination der Dimensionen, muss im Vorwege bei der Aufgabenmodellierung festgelegt werden und ist den individuellen Anforderungen anzupassen[24]. Neben der daraus resultierenden Größe des Merkmalsraums ist die Betrachtung der Generalisierungsbreite dabei zu berücksichtigen.

4.2.3 Tilings Organisation

Für oben beschriebenes Beispielszenario wurde der RL-Zustand in **4.1.3** als $state\ s_t = (u_t(\mathcal{Ab}_1), \dots, u_t(\mathcal{Ab}_N), \mathcal{Z}_t, \mathcal{D}_t)$ definiert. Obwohl die Auflösung und Anzahl der noch festzulegenden Unebenheits-Tilings bisher genauso offen gelassen wurde wie die konkrete Anzahl von Sektoren und Spuren, können und müssen für die Tilings Organisation vorab Abwägungen betrachtet werden, wie sich eine Merkmalskombination aufgrund von semantischen Abhängigkeiten untereinander auf die Größe des Merkmalsraums auswirkt. Zur Veranschaulichung der allgemeinen Auswirkungen wird ein Beispiel herangezogen (siehe auch **Abb. 4.8**). Ferner kann ohne Einschränkung der Allgemeinheit vorerst von einer konkretisierten Tilings-Anordnung (vgl. **2.3.4.1, 4.2.1**) für die Unebenheiten abgesehen und stattdessen ein einzelnes Tiling mit N Tiles betrachtet werden[25].

Beispiel

Die konzentrische Karte besitze zwei Spuren, untergliedert in vier Sektoren (Nord, Ost, usw.), also insgesamt $|\mathcal{Ab}| = |\mathcal{Se}| \cdot |\mathcal{Sp}| = 4 \cdot 2 = 8$ Abschnitte und $|\mathcal{Z}| = |\mathcal{Se}| = 4$ Zielrichtungen. Die Distanz ist in minimaler Schrittzahl angegeben und folgend partitioniert: Ein, Zwei, Drei, Vier bis Zehn, Mehr als Zehn ($|\mathcal{D}| = 5$). Das Unebenheitsmaß sei in unpassierbar (u > 0,9), sehr rau (u > 0,5), mäßig uneben (u > 0,1) und glatt (u ≤ 0,1) kategorisiert: $|\mathcal{U}| = 4$.

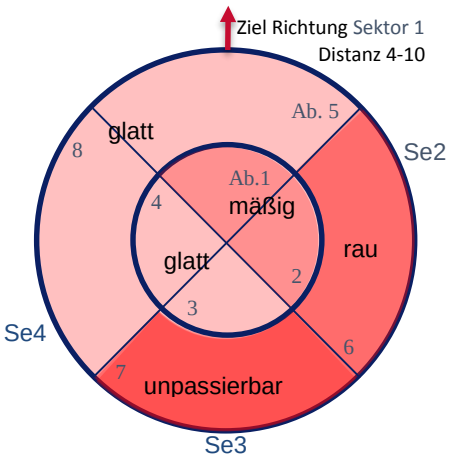


Abb. 4.8: Beispielzustand für eine Roboterkarte mit vier Sektoren und zwei Spuren. Das Ziel liegt in Richtung Sektor 1. Jeder Abschnitt hat ein Unebenheitsmaß aus {glatt, mäßig, rau, unpassierbar}.

[24] Deswegen bietet RLLibrary nur das eindimensionale MultiTiling und keine weitergehenden, multidimensionalen Datenstrukturen.

[25] Letztlich bestimmt die Gesamtzahl der Tiles für eine Eingangsdimension die Größe des Merkmalraums.

Kodierung 1

Die Mindestzahl der Merkmale und somit die Mindestgröße des Merkmalsvektors ergibt sich bei einer voneinander unabhängigen, disjunktiven Betrachtung der Merkmale (vgl. **Abb. 4.6**) $\mathcal{U}, \mathcal{Ab}, \mathcal{Z}, \mathcal{D}$ zu

$$\dim(\vec{f}_1) = |\mathcal{U}| + |\mathcal{Ab}| + |\mathcal{Z}| + |\mathcal{D}| = 4 + 8 + 4 + 5 = 21$$

und hat folglich eine lineare Komplexität O(n). Jedoch lässt sich der Zustand unseres Modells derart nicht abbilden.

Kodierung 2

Die Merkmale Abschnitt und Unebenheit sind offensichtlich voneinander abhängig, da die Karte ein Unebenheitsmaß je Abschnitt anzeigt. Ebenso wie der Zustandsvektor enthält folglich der Merkmalsraum mehrere Abschnittsunebenheiten $\mathcal{U}_{\mathcal{Ab}}$, genauer $|\mathcal{U}| \cdot |\mathcal{Ab}|$ an der Zahl. Die Zielrichtung steht in keinem direkten Zusammenhang zu den Unebenheiten oder Abschnitten. Durch die Relation $\mathcal{U} \times \mathcal{Ab}$ erhöht sich die Dimensionierung zu

$$\dim(\vec{f}_2) = |\mathcal{U}_{\mathcal{Ab}}| + |\mathcal{Z}| = |\mathcal{U}| \cdot |\mathcal{Ab}| + |\mathcal{Z}| + |\mathcal{D}| = 4 \cdot 8 + 4 + 5 = 41$$

was zu einer quadratischen Komplexität O(n²) führt.

Tabelle 4.1: Merkmalstabelle zu Kodierung 2 und dem Beispiel (Abb. 4.8). Die Feature-Menge zu diesem Zustand ist $F_{s,a} = \{2, 6, 9, 13, 15, 23, 28, 29, 33, 40\}$

i	Merkmal	$\vec{f}_i$
1	U(Ab1)=glatt	0
2	**U(Ab1)=mäßig**	**1**
3	U(Ab1)=rau	0
4	U(Ab1)=unpass.	0
…	…	0
6	**U(Ab2)=mäßig**	**1**
…		0
9	**U(Ab3)=glatt**	**1**

i	Merkmal	$\vec{f}_i$
…	…	0
13	**U(Ab4)=glatt**	**1**
…	…	0
15	**U(Ab5)=glatt**	**1**
…	…	0
23	**U(Ab6)=rau**	**1**
…	…	0
28	**U(Ab7)=unpass.**	**1**

i	Merkmal	$\vec{f}_i$
29	**U(Ab8)=glatt**	**1**
…	…	0
33	**Ziel = Se1**	**1**
…	…	0
36	Ziel = Se4	0
…	…	0
40	**Dist = 4-10**	**1**
41	Dist > 10	0

Kodierung 3a

Nun ist aber die Annahme naheliegend, dass eine leichte Abweichung von der Zielrichtung von Vorteil wäre, wenn man dadurch extrem raues Gelände umgehen könnte. Ein solches Domänenwissen kann durch Kombination der jeweiligen Abschnittsunebenheiten mit der Zielrichtung und der Distanz in unser RL-System integriert werden.

$$\dim(\vec{f}_{3a}) = |\mathcal{U}| \cdot |\mathcal{Ab}| \cdot |\mathcal{Z}| \cdot |\mathcal{D}| = 4 \cdot 8 \cdot 4 \cdot 5 = 640$$

Kodierung 3b

Auch in der Kodierung 3a ist die Komplexität *noch* quadratisch, da die Anzahl der Zielrichtungen durch die in $\mathcal{Ab}$ verrechnete Sektorenzahl bestimmt wird. Das lässt sich noch ein wenig durch eine dreiwertige Information reduzieren, ob der jeweilige Abschnitt in Zielrichtung liegt oder in einem

davon benachbarten Sektor oder weit entfernt von der gewünschten Richtung. Die Zielrichtung selbst ist dann in dieser Information implizit mit enthalten.

$$\dim(\vec{f}_{3b}) = |\mathcal{U}| \cdot |\mathcal{AB}| \cdot 3 \cdot |\mathcal{D}| = 4 \cdot 8 \cdot 3 \cdot 5 = 480$$

Tabelle 4.2: Merkmalstabelle zu Kodierung 3b und Beispiel (Abb. 4.8). Die Feature-Menge zu diesem Zustand ist $F_{s,a} = \{19, 84, 134, 189, 244, 339, 418, 429\}$

i	Unebenheit	Richtung	Dist	$\vec{f}_i$
1	U(Ab1)=glatt	richtig	1	0
2	U(Ab1)=glatt	richtig	2	0
...	...	...	...	0
19	**U(Ab1)=mäßig**	**richtig**	**4-10**	**1**
...	...	...	...	0
84	**U(Ab2)=mäßg**	**nahe**	**4-10**	**1**
...	...	...	...	0
134	**U(Ab3)=glatt**	**falsch**	**4-10**	**1**
...	...	...	...	0
189	**U(Ab4)=glatt**	**nahe**	**4-10**	**1**

i	Unebenheit	Ziel	Dist	$\vec{f}_i$
...	...	...	...	0
244	**U(Ab5)=glatt**	**richtig**	**4-10**	**1**
...	...	...	...	0
339	**U(Ab6)=rau**	**nahe**	**4-10**	**1**
...	...	...	...	0
418	**U(Ab7)=unpass.**	**falsch**	**4-10**	**1**
...	...	...	...	0
429	**U(Ab8)=glatt**	**nahe**	**4-10**	**1**
...	...	...	...	0
480	U(Ab8)=unpass.	falsch	>10	0

Kodierung 4

Die beste Approximation erhielte man wohl mit der in der Literatur gängigen Kombination aller Merkmale untereinander, sodass ein einziges Merkmalselement den Zustand vollständig abbildet. In diesem Beispiel wäre das eine Konjunktion der Zielrichtung und eine aller Abschnittsunebenheiten untereinander: $(\mathcal{U} \times \mathcal{AB}_1) \times \ldots (\mathcal{U} \times \mathcal{AB}_N) \times \mathcal{Z} \times \mathcal{D}$. Während die Dimensionen für $\vec{f}_2$ und $\vec{f}_3$ noch proportional mit der Anzahl der Abschnitte wachsen, würde diese Variante exponentiell steigen und sich somit in der Regel nur für sehr kleine Tilings eignen.

$$\dim(\vec{f}_4) = |\mathcal{U}|^{|\mathcal{AB}|} \cdot |\mathcal{Z}| \cdot |\mathcal{D}| = 4^8 \cdot 4 \cdot 5 = 1.310.720$$

Tabelle 4.3: Merkmalstabelle zu Kodierung 4 und Beispiel (Abb. 4.8). Die Feature-Menge zu diesem Zustand ist $F_{s,a} = \{82.097\}$

i	U(Ab1)	U(Ab2)	...	U(Ab8)	Ziel	$\vec{f}_i$
1	glatt	glatt	...	glatt	Se1	0
2	glatt	glatt	...	glatt	Se2	0
...						0
82.097	**mäßig**	**mäßig**	**...**	**unpass.**	**Se1**	**1**
...	...					0
262.144	unpass.	unpass.	...	unpass.	Se4	0

Kodierung 4b

Eine weitere naheliegende Vermutung ist, dass die semantische Kopplung zwischen den geographisch geprägten Unebenheitsmerkmalen und den eher navigatorischen Merkmalen Distanz und Zielrichtung schwächer ausgeprägt ist. Die Kodierung 4b betrachtet daher die Abschnittsunebenheiten unter-

einander sowie die Kombination aus Distanz und Richtung jeweils als voneinander abhängig, jedoch die Verknüpfung von Geographie und Navigation als untereinander unabhängig. Dadurch wird ein Kompromiss zwischen der rein konjunktiven und den vorigen eher disjunktiven Kodierungen erzielt, auch wenn die Komplexität immer noch exponentiell ist.

$$\dim(\vec{f}_{4b}) = |\mathcal{U}|^{|\mathcal{A}\mathcal{b}|} + |\mathcal{Z}| \cdot |\mathcal{D}| = 4^8 + 4 \cdot 5 = 65.556$$

Tabelle 4.4: Merkmalstabelle zu Kodierung 4b und Beispiel (Abb. 4.8). Die Feature-Menge zu diesem Zustand ist $F_{s,a} = \{20.525, 65.548\}$

i	U(Ab1)	U(Ab2)	...	U(Ab8)	$\vec{f}_i$
1	glatt	glatt	...	glatt	0
2	glatt	glatt	...	glatt	0
...					0
20.525	**mäßig**	**mäßig**	**...**	**unpass.**	**1**
...	...				0
65.536	unpass.	unpass.	...	unpass.	0

i	Dist&Ziel	$\vec{f}_i$
65.537	Dist=1/Se1	0
65.538	1/Se2	0
		0
65.548	**4-10/Se1**	**1**
		0
65.556	>10/Se4	0

4.3 Zusammenfassung

Weitgehend unabhängig von den nicht konkretisierten Modulen *Locomotion* und *Visual Analysis* konnte die Lernaufgabe für das Modul *Learn Estimation* modelliert werden, insbesondere der Zustand im RL-Kontext. Die Betrachtung des multidimensionalen und reelwertigen Zustandsraums führte zum Tile Coding Verfahren, dass einige Optionen hinsichtlich der Merkmalsorganisation offenbart, die unterschiedliche Auswirkungen auf das Lernverhalten und die Komplexität der Aufgabe haben. Diese werden in Experimenten näher evaluiert.

5 Versuchsaufbau

Die vorangegangene Modellierung der Lernaufgabe ist die Basis für jedes weitere Vorgehen. In ihr steckt die Hauptarbeit bei der Herangehensweise zur Lösung des Problems durch Abstraktion des selbigen hin zu einem Reinforcement Learning Problem. Im Detail bleiben aber einige Fragen noch offen, insbesondere bei der Organisation der Merkmale des Zustandsraums aber auch bezüglich grundlegender Parameter des Lernalgorithmus. In Experimenten soll bezüglich unseres Lernansatzes geklärt werden, wie die Lernparameter und die Merkmalsorganisation am besten auszugestalten sind, um ein möglichst gutes Lernergebnis zu erzielen. Es wird hierzu ein Testmodell und eine Testumgebung geschaffen anhand derer aussagekräftige, experimentell ermittelte Empfehlungen eruiert werden können. Im darauffolgenden **Abschnitt 6** werden die Experimente durchgeführt und deren Ergebnisse analysiert.

5.1 Hexagonalwelt

Die ersten Analysen sollen zeigen, dass die Aufgabe prinzipiell mittels Reinforcement Learning zu bewältigen ist, dass es überhaupt eine erlernbare Strategie gibt, die lediglich anhand der Bodeneigenschaften der unmittelbaren Umgebung gute Schrittpositionen erkennt. Dann stellt sich die Frage nach der Parametrisierung des Sarsa(λ) Algorithmus und der Tilings-Organisation für die kontinuierlichen Zustandsdimensionen. Welche Einstellungen begünstigen eine schnelle und gute Lernentwicklung? Um dies zu evaluieren, ist es notwendig, eine klar definierte Testumgebung zu erzeugen, bei der möglichst für alle Zustands-Aktions-Paare die optimalen Funktionswerte – die Kosten – von vornherein bekannt sind und mit dem erlernten Wissen verglichen werden können.

Mit der in **4.1** vorgestellten Modellierung der Lernaufgabe ist es noch recht komplex, eine solch wohldefinierte Testumgebung zu erzeugen, da allein schon die Bewegungsaktionen von der Organisation der Umgebungskarte abhängig sind und die möglichen Pfade und die resultierenden Vergeltungen schwer und wenig intuitiv vorherzusagen sind. Für eine Analyse und Optimierung des Lernalgorithmus wäre es darüber hinaus hilfreich, sich schrittweise der letztlichen Komplexität anzunähern, beginnend mit einzelnen Lernparametern wie der Lernrate (**2.3.2**) hin zur Tilings-Organisation der reellen Dimensionen (**4.2.3**). So ist es vorteilhaft, im ersten Schritt auf die tabellarische Implementierung von Sarsa(λ) zu setzten, da dieses Verfahren, etablierter, eleganter und leichter nachvollziehbar ist, als eine Implementierung mit Funktionsapproximation. Das Modell der

Lernaufgabe muss folglich erst zu einem diskreten Modell mit einem in einer zu bewältigen Größenordnung gegebenen Zustandsraum vereinfacht werden, um es spätestens für die Tilings-Analysen wieder auf ein kontinuierliches Modell zu verallgemeinern. Das folgend beschriebene Dampfross®-Szenario ermöglicht genau diesen Weg.

5.1.1 Dampfross

Bei dem Brettspiel Dampfross®[26] besteht das Spielfeld aus einer Karte von beispielweise Deutschland oder Italien, untergliedert in sechseckige Felder. Es gibt Ebenen, Berge, Wasserfelder und Städte, sowie Flüsse zwischen den Hexagonen. Ziel des Spieles ist es, Städte mit Bahnlinien zu verbinden, die man von einem Feld zum nächsten zieht und Wettrennen zu fahren. Der Bau eines Streckenabschnitts kostet je nach Feldtyp unterschiedlich viel und eine Flussquerung oder das Kreuzen gegnerischer Bahnlinien ist mit zusätzlichen Kosten verbunden.

5.1.2 Dampfross Lernaufgabe

Bei der Lernaufgabe dieser Thesis gibt es Abschnitte mit unterschiedlichen Unebenheitsmaßen, die Einfluss auf die Trittqualität oder eben indirekt auf die Bewegungskosten des Roboters haben. Dem Algorithmus stehen zusätzlich die Informationen über die ungefähre Richtung und Entfernung zum nächsten Etappenziel zur Verfügung. Wendet man diesen Grundgedanken auf das Spiel Dampfross® an, ist es eine äquivalente Aufgabe, wenn der Spieler nur das aktuelle Feld, wo seine Bahngleise beginnen oder bisher enden, sowie die benachbarten sechs Felder sehen kann. Man gibt ihm noch die ungefähre Richtung (bspw. „über Feld Nord-Ost“) und Entfernung zur nächsten Stadt an und der Spieler muss nun nur anhand dieser Informationen entscheiden, in welches Nachbarfeld er die Bahngleise zieht, um letztlich im Durchschnitt die geringsten Streckenkosten zur Stadt zu erzielen. Ein Mitspieler und konkurrierende Gleise wie beim Brettspiel werden hier ausgespart.

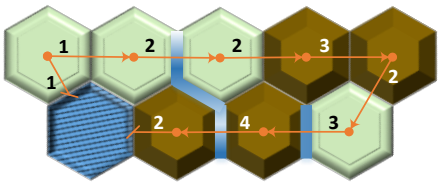


Abb. 5.1: Streckenkosten (Abweichend von Dampfross®). Die Bewegung auf ein Wasserfeld verursacht Kosten, wird aber nicht vollzogen.

[26] Spiel des Jahres 1984, David G. Watts, Queen Games
http://de.wikipedia.org/wiki/Dampfross , http://www.spieldesjahres.de/cms/front_content.php?idart=622
(besucht: 2. Jul 2014)

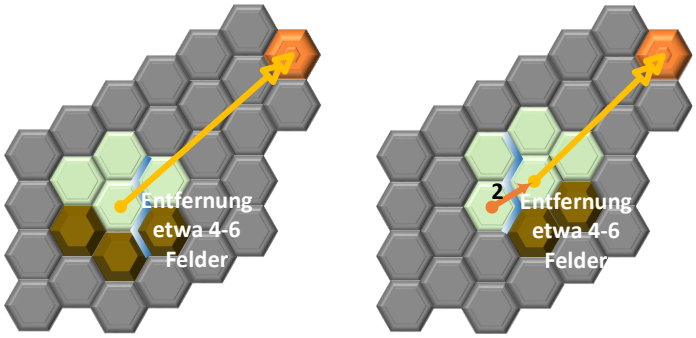


Abb. 5.2: Dampfross® Aufgabe. Die Landschaft besteht aus Ebenen, Bergen und Flüssen. Links: aktuelle Sicht mit Richtung und Entfernung der zu erreichenden Stadt. Rechts: Neue Sicht, nach einem Schritt mit zwei Streckenkosten. Aufgabe: Schätze, über welches Nachbarfeld die geringsten Streckenkosten zur Stadt entstehen.

5.1.3 Herausforderung der Lernaufgabe – Äquicausa Problem

Ist es überhaupt möglich, anhand solch magerer Informationen eine kostenoptimierte Strategie zu erlernen? Bei einer eher inhomogenen Umwelt unterscheiden sich die Zustände – die Sichtfelder – wahrscheinlich ausreichend, sodass die tatsächlichen Kosten nach gewisser Zeit erlernt werden. Vermutlich wollte man aber trotzdem dafür Sorge tragen, dass die Entfernung zum Etappenziel innerhalb eines engeren Radius liegt. Denn je größer der Radius, desto wahrscheinlicher gibt es zwei oder mehr Orte auf der Karte, bei denen Sichtfeld, Entfernung und Richtung (der RL-Zustand) gleich sind, aber unterschiedliche Streckenkosten zum Ziel aufweisen. Je homogener die Welt ist, beispielsweise eine, die überwiegend aus Ebenenfeldern besteht, desto eher tritt dieses Problem auf. Im Weiteren wird diese Herausforderung als ***Äquicausa Problem*** bezeichnet: Unter gleichen Bedingungen (Sichtfeld) müsste der Zustand unterschiedlich bewertet werden.

Dies stellt für die eigentliche Lernaufgabe aber tatsächlich kein Problem dar. Denn schließlich soll nicht eine konkrete Karte (Deutschland) erlernt werden, sondern eine Strategie für beliebige Karten. Treten die beschriebenen Härtefälle auf, wird der Lernalgorithmus mit Sicherheit länger benötigen, um akzeptable Schätzungen zu ermitteln. Letztlich stellt sich aber ein Durchschnittswert ein, der über lange Sicht eine global gültige Strategie darstellt, kosteneffizient Bahnlinien zu bauen oder eben, bezogen auf den Laufroboter, günstige Trittpositionen auszuwählen.

Behauptung 2

> Für das Äquicausa Problem stellt sich ein Durchschnittswert ein, der über lange Sicht eine global gültige Strategie darstellt.

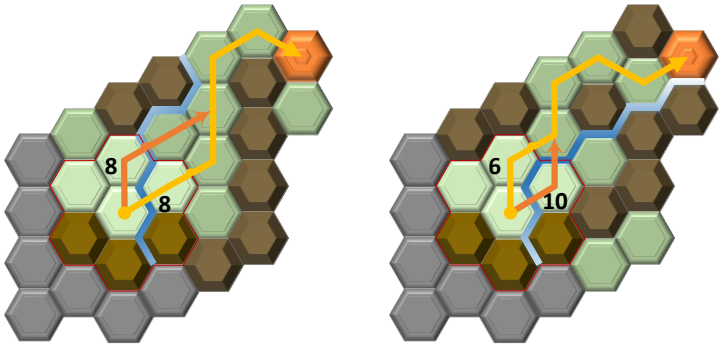


Abb. 5.3: Derselbe Zustand in unterschiedlichen Welten mit verschiedenen Streckenkosten. Das tatsächliche Sichtfeld ist rot umrandet. Die verblassten Felder zeigen die jeweils nicht sichtbare Umwelt zwischen Position und Ziel.

Beispiel

Die Herausforderung für das Erlernen einer globalen Strategie für beliebige Karten wird in **Abb. 5.3** verdeutlicht. Derselbe Zustand – Rot umrandet – tritt in zwei Welten auf. Das Gelände zwischen aktueller Position und Stadt ist für den Agenten nicht sichtbar, aber zur Verdeutlichung auf den beiden Abbildungen für die jeweilige Karte verblasst dargestellt. Trainiert der Agent auf der linken Welt, so lernt er mit der Zeit, dass die Wege über das nördliche und das nordöstliche Nachbarfeld jeweils mit acht Punkten die kostengünstigsten sind. Setzt man den Agenten nun in die rechte Welt, so ist für ihn die Ausgangssituation dieselbe. Die tatsächlichen Gesamtkosten sind hier mit sechs und zehn Punkten jedoch anders. Die bisher gelernte Strategie passt nicht mehr. Trainiert der Agent nun aber abwechselnd in den beiden Welten, so stellt sich mit der Zeit ein Mittelwert ein: Sieben Punkte für Norden, neun Punkte für Nordosten. Dies gibt zwar weder in der linken, noch in der rechten Welt die wirklichen Kosten wieder, stellt aber nichts desto trotz eine gute Strategie für beide Welten dar, wenn man sich für den durchschnittlich günstigeren Weg über den nördlichen Nachbarn entscheidet.

5.2 Hexagonale Gitter

Der Umgang mit einem hexagonalen Gitter ist etwas umständlicher als mit einem quadratischen. Spieleentwickler haben sich intensiv mit dem Thema auseinandergesetzt und elegante Methoden vorgestellt. Dieser Abschnitt erläutert einige, dieser hier relevanten und auf diese Aufgabe zugeschnittenen Methoden.

Hexagon

Ein reguläres Hexagon ist ein gleichseitiges, sechseckiges Polygon. Typische Ausrichtungen von hexagonalen Gittern sind horizontal (oben spitz) und vertikal (oben flach).

Viele Brett- und Computerspiele benutzen hexagonale Felder, die aus algorithmischer Sicht etwas umständlicher zu handhaben sind als einfache quadratische Feldgitter. Insbesondere aus der

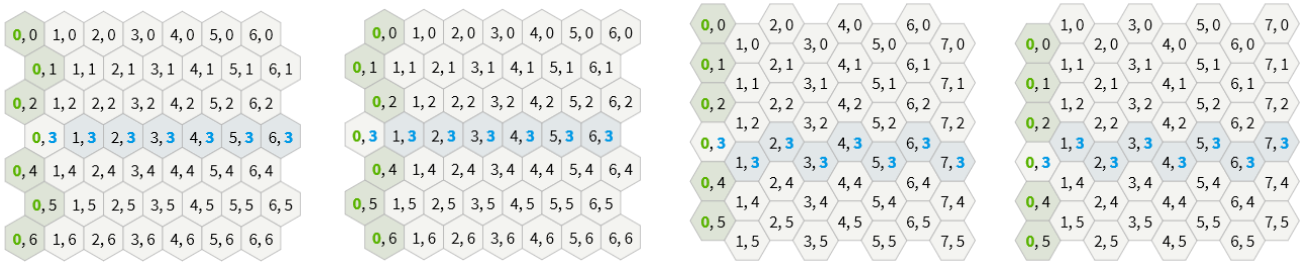
Spieleentwicklung kommen jedoch zahlreiche elegante Methoden zum Umgang mit Hexagonen. Dieser Abschnitt erläutert einige, dieser hier relevanten und auf diese Aufgabe zugeschnittenen Methoden. Die Betrachtung beschränkt sich dabei auf reguläre, gleichseitige Sechsecke. Für einen etwas Umfangreicheren Überblick zum Thema, sei die Web-Seite[27] von Amit Patel empfohlen.

5.2.1 Koordinatensysteme

Ein Raster von quadratischen Feldern ist praktisch immer auf die eine offensichtliche Art angeordnet. Für Hexagone gibt es jedoch mehrere Ansätze. Für das Modell in **5.1** werden axiale Koordinaten benutzt.

5.2.1.1 Versetzte Koordinaten

Ein weit verbreiteter Ansatz ist es, Zeilen oder Spalten gegeneinander zu versetzen, entweder jeweils die geraden oder die ungeraden, sodass es für die horizontalen und vertikalen Hexagone jeweils zwei Varianten gibt. Die Zeilen werden im Weiteren mit **r** (eng.: *row*) und die Spalten mit **c** (eng.: *column*) abgekürzt.



r-ungerade horizontal | r-gerade horizontal | c-ungerade vertikal | c-gerade vertikal

Abb. 5.4: Hexagonale, versetzte Rasterkoordinaten (Quelle: Amit Patel)

5.2.1.2 Kubische Koordinaten

Eine andere Methode ergibt sich, wenn man die drei Hauptachsen eines Hexagons betrachtet, im Gegensatz zu den zwei bei quadratischen Rastern. Es stellt sich eine elegante Symmetrie zwischen Hexagonen und Würfeln dar. Dazu stelle man sich einen Kubus, zusammengesetzt aus einzelnen Würfeln, vor und schneide eine diagonale Scheibe auf $x + y + z = 0$ aus. Projiziert man die Achsen des kartesischen dreidimensionalen Koordinatensystems auf die Schnittfläche, liegen sie genau auf den Diagonalen des Hexagonrasters. Jede Richtung auf dem Würfelgitter korrespondiert mit einer Linie

[27] www.redblobgames.com, *Hexagonal Grids*, zuletzt besucht: 13.06.2014

auf dem Hexagonraster. Jede Richtung im Hexagonraster zu einem Nachbarn entspricht einer Kombination von zwei Richtungen auf dem Würfelgitter, da sie genau zwischen zwei Achsen liegen. So ist eine Bewegung nach Norden gleichbedeutend mit plus einem Würfel entlang der Y-Achse und minus einem entlang der Z-Achse.

Alle Datenstrukturen und Algorithmen, die kubische oder axiale Koordinaten benutzen, setzen voraus, dass $x + y + z = 0$ gilt. Nur dann sind sie effizient einsetzbar.

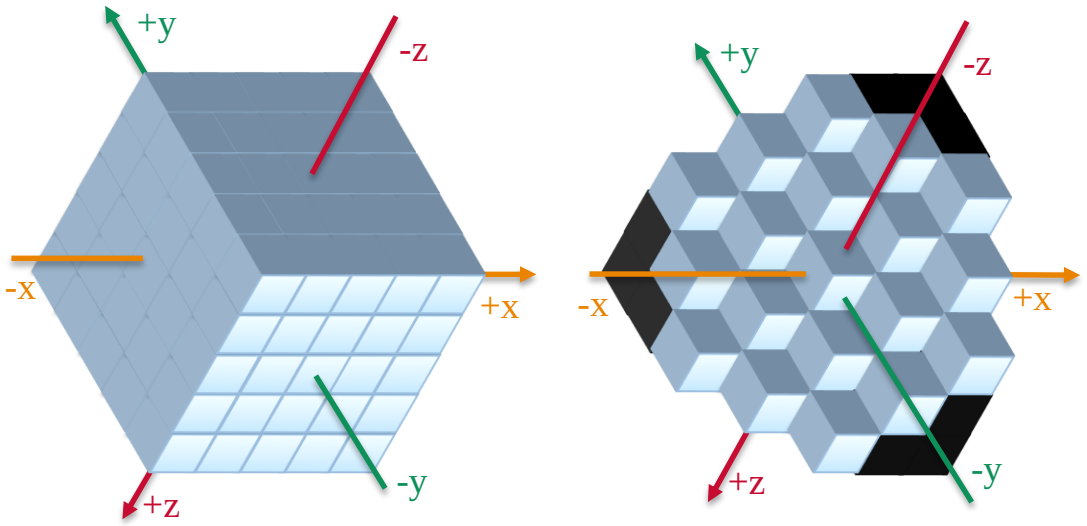


Abb. 5.5: Projektion vom kubischen Gitter auf ein hexagonales.

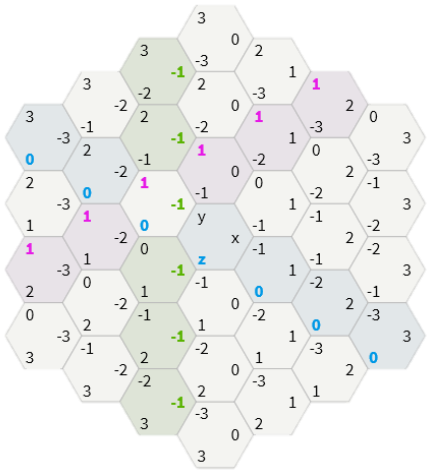


Abb. 5.6: kubische Koordinaten für ein Hexagonraster. Farblich hervorgehoben ist, dass je Richtung eine Achsenkoordinate gleich bleibt. (Quelle: Amit Patel)

5.2.1.3 Axiale Koordinaten

Unter der Prämisse, dass die Summe der Achs-Koordinaten Null ist, lässt sich einer der drei Werte aus den anderen zweien errechnen. Somit lässt sich ein Zwei-Achsen-Koordinatensystem realisieren. Es ist dabei freigestellt, welche der x, y, z Achsen auf die Spalten oder Reihen gelegt werden. Doch muss bedacht werden, dass es sich hier nicht um ein kartesischen Koordinatensystem handelt, da die Achsen 60° beziehungsweise 120° zueinander stehen, was das Speichern der Felder etwas umständlicher macht gegenüber der Variante mit versetzten Koordinaten. Dafür sind die Algorithmen für axiale Koordinaten prägnanter und klarer.

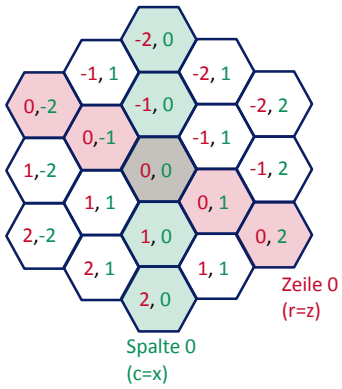


Abb. 5.7: Axiale 2D Hexagon-koordinaten

Die weiteren Erläuterungen beziehen sich ausschließlich auf axiale Koordinaten, wie sie in diesem Versuchsaufbau verwendet werden.

5.2.2 Hex-Gitter Operationen

Die Operationen hier beziehen sich auf axiale Koordinaten, bei denen die Z-Achse auf die Zeile (**r**) und die X-Achse auf die Spalte (**c**) gelegt ist. Für die Umrechnung in kubische Koordinaten gilt:
$y = -(z + x) = -(r + c)$.

5.2.2.1 Nachbarschaft

Eine Bewegung auf eines der sechs Nachbarfelder beinhaltet im kubischen Koordinatensystem das Inkrementieren eines der drei Koordinaten und Dekrementieren eines der beiden übrigen, sodass die Summe immer Null ist. Das macht – wie passend – $3 \cdot 2 = 6$ mögliche Änderungen. Eine effiziente Implementierung ist es, die sechs Varianten in einer *Lookup Table* vorzuhalten. Ausgehend vom kubischen System, erhält man auch die Tabelle für axiale Koordinaten:

```
enum Richtung { NO=0, N=1, NW=2, SW=3, S=4, SO=5 }

MoveCub := ( (+1, 0, -1), (0, +1, -1), (-1, +1, 0),
             (-1, 0, +1), (0, -1, +1), (+1, -1, 0) );
MoveHex := ( (+1, -1),    (0, -1),     (-1, 0),
             (-1, +1),    (0, +1),     (+1, 0) );
```

Mittels der Lookup Table lässt sich dann auch einfach eine Folge von Schritten umsetzen. So können beispielsweise drei Schritte nach Nordost und zwei nach Nord (die Reihenfolge ist beliebig) durch Multiplikation der Richtungskoordinaten errechnet werden:

```
hexCoord ← hexCoord + 3 · MoveHex[Richtung.NO] + 2 · MoveHex[Richtung.N]
```

5.2.2.2 Distanz

Angelehnt an die schachbrettartige Anordnung von Blöcken im New Yorker Stadtteil Manhattan, hat sich für die Distanz zweier Felder in einem quadratischen Raster der Begriff der Manhattan Distanz etabliert. Bei einem Würfel ist die Manhattan Distanz $d(C_1, C_2) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$. Wie eingangs beschrieben, entspricht die Bewegung um ein Feld im Hex-Gitter zweien im kubischen. Folglich ist die Distanz im kubischen und axialen Koordinatensystem eines Hex-Gitters die Hälfte der Manhattan Distanz für Würfel. Für die Distanzgleichung bei axialen Koordinaten nutzt man die Umrechnung in kubische Koordinaten:

```
int HexDistance(int c1, int r1, int c2, int r2) {
 // return (Abs(c1-c2) + Abs((-c1-r1)-(-c2-r2)) + Abs(r1-r2)) / 2;
    return (Abs(c1-c2) +   Abs(c2-c1 + r2-r1)   + Abs(r1-r2)) / 2;
}
```

Ohne Herleitung sei hier noch eine äquivalente Berechnung angegeben:

```
int HexDistance(int c1, int r1, int c2, int r2) {
    return Abs( Max( (c2-c1), (r2-r1), ((c2-c1) - (r2-r1)) ) );
}
```

5.2.2.3 Richtung

Gegeben sind ein Startfeld A und ein Zielfeld B. Zu ermitteln, in welche Richtung B liegt, ist gleichbedeutend mit der Aufgabe, eine Linie zwischen A und B auf die Hex-Felder zu interpolieren und dabei beim ersten Nachbarfeld von A abzubrechen. Bei Computer Graphiken wird für die Interpolation einer Linie häufig ein Digitaler Differential Analysierer[28] (DDA) genutzt. Um die Richtung zu errechnen, projiziert man zuerst die gedachte Start-Ziel-Linie auf das Null-Feld als Ausgangspunkt. Anschließend wendet man den ersten Schritt des DDA-Algorithmus an und rundet das Ergebnis.

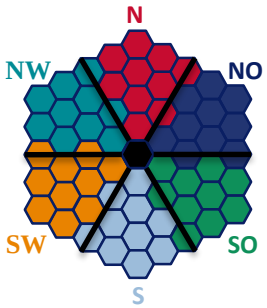


Abb. 5.8: Richtungen zu Zielfeldern

[28] Siehe http://en.wikipedia.org/wiki/Digital_differential_analyzer_(graphics_algorithm) (zuletzt besucht am 15.06.14)

```
Richtung HexRichtung(Hex A, Hex B) {
    Hex C ← B - A;          // Operatoren +/- für Hex-Datenstruktur implementiert
    Hex dirHex ← NextHexToTarget(Hex(0, 0), C);
    return (Richtung)dirHex;              // cast von Hex ∈ MoveHex auf Richtung
}
Hex NextHexToTarget(Hex A, Hex B) {
    int dist ← HexDistance(A.Col, A.Row, B.Col, B.Row);
    // erster Schritt der DDA-Linien-Interpolation:
    HexReel next ← A·(1 − 1/dist) + B·1/dist  // Sonderfall Distance = 0 hier nicht betrachtet
    return HexRound(next.Col, next.Row);
}
Hex HexRound(double rc, double rr) {
    HexCube cube ← HexRound(rc, -(rc + rr), rr); // HexCube: kubische Koords
    return new Hex(cube.X, cube.Z);
}
HexCube HexRound(double rx, double ry, double rz) {
    HexCube cube ← new HexCube(Math.Round(rx), Math.Round(ry) ), Math.Round(rz));
    if (Koordinatensumme nicht 0) then {
        Nimm zwei gleiche Koordinatenwerte und korrigiere auf Basis dieser
        den dritten, sodass x+y+z = 0
    }
    return cube;
}
```

Das Runden erfolgt über kubische Koordinaten, da damit die Sonderfälle besser behandelt werden können. Zu gegebener, reellwertiger Kubus-Koordinate (rx, ry, rz) werden alle Elemente auf den ganzzahligen Wert (ix, iy, iz) gerundet. Auch wenn rx + ry + rz = 0 gilt, gibt es keine Garantie, dass die Nullsummenregel auch nach dem Runden noch erfüllt ist. Alle Felder, die auf den Linienachsen (den Hex-Diagonalen) liegen, sind von der Verletzung der Invariante betroffen, da bei ihnen zwei von den drei Koordinaten gleich sind. Hier ist eine Korrektur zur Wiederherstellung der Nullsummenregel erforderlich. **Abb. 5.8** zeigt welche Felder vom Algorithmus auf welche Richtung gelegt werden.

5.2.3 Kartenspeicherung

Gegenüber versetzten Koordinaten verbraucht das axiale System erst einmal mehr Speicherplatz, wenn man die Koordinaten direkt in Array-Indizes überträgt. In der ersten Matrix in **Abb. 5.9** sind die überschüssigen Array-Felder grau unterlegt. Entweder nimmt man den erhöhten Speicherbedarf hin oder verwendet Hashing oder, die hier bevorzugte Variante, verschiebt die Spalten und kapselt den Arrayzugriff durch eine Umrechnung der Axialkoordinaten in Array-Indizes.

Die konkrete Umrechnung ist Abhängig von der Kartenform (rechteckig, dreieckig, hexagonal, etc.), von der Hexagon Anordnung (horizontal, vertikal) und davon, welche zwei kubischen Koordinaten für das axiale System genutzt werden. Letztlich sind aber alle Umrechnungen ähnlich. In **Abb. 5.9** ist die Karte aus vertikalen Sechsecken rechteckig zusammengesetzt und nutzt die (x,z)-Koordinaten für das Axialsystem. Die Umrechnung in Array-Indizes erfolgt dann unter Angabe der Axial-Koordinaten (r,c) durch `array[r + c/2][c]` (c/2 abgerundet).

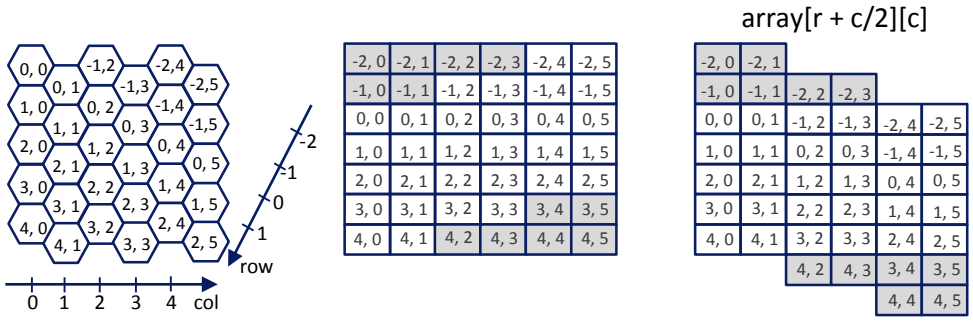


Abb. 5.9: Übertrag vom axialen Koordinatensystem in ein zweidimensionales Array und mit Versatz der Spalten

5.3 Tabellarische Zustandsmodellierung

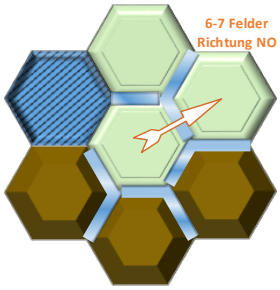


Abb. 5.10: Schematische Darstellung eines RL-Zustands im Dampfross® Szenario

Im Dampfross® Modell besteht der RL-Zustand aus dem Sichtfeld des Agenten, wiederum zusammengesetzt aus dem zentralen Hexagon, auf dem sich der Agent befindet und den sechs angrenzenden Feldern. Die aktuelle Position ist nicht direkt bekannt, denn sie ist für die Lernaufgabe nicht relevant. Stattdessen gehören zum Zustand die ungefähre Richtung und Entfernung zum Ziel (einer Stadt). Dabei ist die Richtung als eine der sechs möglichen Bewegungsrichtungen NO, N, NW, SW, S, SO definiert und die Entfernung als Hexdistanz (siehe **5.2.2.2**) angegeben. Für größere Entfernungen müssen Intervalle angegeben werden. Rein intuitiv begründet sind es hier insgesamt folgende zehn Entfernungen: (0-1, 2, 3, 4, 5, 6-7, 8-11, 12-19, 20-35, >35). Der RL-Zustand gibt die Umgebungseigenschaft in Form des Landschaftstyps (Ebene, Berg, Wasser, Stadt) eines jeden Hexagon im Sichtfeld wieder. Jedes der dem Zentrum angrenzenden Felder kann zudem ein Flusselement an der Kante zum Zentrum oder zum linken Nachbarfeld oder zu beiden hin haben[29]. Flusselemente die auf der Grenze des Sichtfeldes liegen – zwischen einem Nachbarfeld und dahinterliegendem Hexagon – gelten als nicht mehr sichtbar.

Um die Gesamtzahl der daraus resultierenden möglichen Zustände zu reduzieren, werden noch einige domänenspezifische Eigenschaften zur Umwelt und Bewegungsmöglichkeit berücksichtigt. So wird ein Stadtfeld wie eine Ebene behandelt[30] und die Kanten von Wasserfeldern sind nie Flüsse, beziehungs-

[29] Alternativ ließen sich die möglichen Flussläufe auch als zehnstellige Binärzahl modellieren. Jede der zehn sichtbaren Kanten hat einen Fluss oder nicht.

[30] Ein Streckelement in eine Stadt hinein hat keine Zusatzkosten und ist somit Kostenäquivalent zur Ebene.

weise können ignoriert werden, da das Ziehen auf Wasserfelder nicht erlaubt und somit eine Flussquerung nicht möglich ist, ebenso wie das Zentrumsfeld dann nicht vom Typ Wasser sein kann. Insgesamt ergibt das $120 \cdot 9^6 \cong 64 \cdot 10^6 \cong 2^{25}$ mögliche Zustände:

1. Ein Nachbarfeld ist ein Wasserfeld ohne Flusskante oder Ebene / Berg mit keiner Flusskante, einer zum Zentrum, einer zum linken Nachbarn oder je einer zum Zentrum und zum Nachbarn. Je Nachbarfeld gibt es damit $1 + 2 \cdot 4 = 9$ Ausprägungen.
2. Es gibt stets sechs Nachbarfelder, also $9^6 = 531.441$ Varianten der Nachbarschaft.
3. Das Zentrumsfeld ist vom Typ Ebene oder Berg: $2 \cdot 9^6 = 1.062.882$ Varianten.
4. Das Ziel liegt in einer von sechs möglichen Richtungen und hat zehn mögliche Entfernungen (Distanzintervalle): $10 \cdot 6 \cdot 2 \cdot 9^6 = \mathbf{63.772.920}$ **Zustände**

Mag der erste Eindruck ob der Vielzahl an Zuständen noch einschüchtern, so ist die Anzahl jedoch mit durchschnittlichen, handelsüblichen Rechnerkapazitäten und kleinen, Performanz und Speicher optimierenden Maßnahmen (siehe Anhang **A.1**) in der Implementierung zu bewältigen. Anhand des Dampfross-Szenarios können Entwicklungen des Lernprozesses und der Einfluss der Lernparameter untersucht werden mit der Perspektive, die Ergebnisse in das AMEE-Lernsystem zu überführen.

5.3.1 Perspektivenabhängigkeit

Eine auf den ersten Blick vielleicht eigenartig anmutende Konsequenz der Zustandsmodellierung, die sich auch im kontinuierlichen Beispiel fortsetzt, ist es, dass ein und dieselbe Hexagonkachel je nach Perspektive unterschiedlich wahrgenommen und letztlich auch unterschiedlich bewertet wird. Im Beispiel aus **Abb. 5.11** sind für zwei Perspektiven die gemeinsamen Kacheln hervorgehoben. Aus der linken Sicht, markiert durch einen blauen Punkt, ist der Zug auf die rechten Kacheln mit erhöhten Kosten für eine Flussquerung verbunden. Aus der rechten Perspektive (orangener Punkt) ist kein Fluss im Weg, folglich werden dieselben Kacheln günstiger bewertet, als zuvor aus der linken Perspektive. In der realen Welt wäre diese Situation vergleichbar damit, ob man vor oder hinter (unter) einer Leiter steht. Die Leiter von vorne zu erklimmen ist auf jeden Fall leichter als andersherum und die vordere Ausgangslage ist dann selbstverständlich günstiger zu bewerten – auch wenn es sich um dieselbe Leiter handelt.

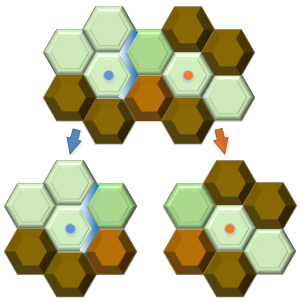


Abb. 5.11: Unterschiedliche Sicht auf und Bewertung von ein und derselben Kachel.

5.4 Kontinuierliche Landschaftsmerkmale

Während das diskrete Dampfrossmodell im Wesentlichen dazu dient, an einem vereinfachten Aufgaben-Modell zur Schrittpositionswahl zu zeigen, dass es überhaupt eine erlernbare Strategie gibt und um die Lernparameter in nachfolgenden Experimenten zu justieren, soll in ***DampfrossContinuum*** der Schritt hin zu kontinuierlichen (reellen) Wertebereichen gegangen werden. Dazu werden die bisherigen Landschaftstypen und Flusskanten je Kachel eines Sichtfeldes in reelle Zahlen überführt, die ein abstraktes Maß als landschaftstypische Kennzahl repräsentieren. Dabei ist je ein Wertebereich äquivalent zu einer Kachelausprägung (siehe **Tabelle 5.1**).

Anhand dieses modifizierten Dampfross®-Szenarios kann die Auswirkung verschiedener Tilings-Organisationen (Kap. **4.2**) überprüft werden. Das kontinuierliche Dampfross® dient hierbei als Referenz bezüglich Tempo und Genauigkeit des Lernprozesses.

Tabelle 5.1: Äquivalenztabelle Landschaftstyp einer Kachel zu reellem Landschaftsmaß.

Kachel-Landschaftstyp	Wertebereich Landschaftsmaß
Ebene (oder Stadt) ohne Fluss	[0.0, 0.1[[31]
Ebene, Flusskante zum linken Nachbarn	[0.1, 0.2[
Ebene, Flusskante zum Zentrum	[0.2, 0.3[
Ebene, Flusskante links und zum Zentrum	[0.3, 0.4[
Berg ohne Fluss	[0.4, 0.5[
Berg, Flusskante zum linken Nachbarn	[0.5, 0.6[
Berg, Flusskante zum Zentrum	[0.6, 0.7[
Berg, Flusskante links und zum Zentrum	[0.7, 0.8[
Wasser	[0.8, 0.9[

5.5 Testplanung

In der Literatur finden sich mehrere Untersuchungen zum Lernverhalten verschiedener RL-Algorithmen sowie der Auswirkung und eventueller Abhängigkeiten der Lernparameter. Die ersten Testvorhaben sollen die Angaben zur Parametrisierung von Sarsa(λ) in der tabellarischen Implementierung und bezogen auf die hiesige Lernaufgabe, in der Dampfross® Variante, überprüfen. Die vielversprechendsten Einstellungen werden anschließend für die Implementierung mit der Linearen

[31] [0.0, 0.1[bezeichnet ein halboffenes Intervall von 0 inklusive bis 0,1 exklusive. Die Verwendung eines Dezimal**punkts** dient der besseren Lesbarkeit.

Gradientenabstiegsmethode für Funktionsapproximation angewendet. Für letztere Umsetzung folgen schließlich Untersuchungen zur Tilings-Organisation. Als Referenz für die Lernparameter dienen im Wesentlichen die Experimente von Anne Wolter [Wol08], die in Ihrer Arbeit umfangreiche Untersuchungen zum Reinforcement Learning in der Roboternavigation vornimmt.

5.5.1 Testverfahren

In erster Linie gilt es zu prüfen, welchen Einfluss die Lernparameter auf Geschwindigkeit und Qualität des Lernprozesses haben. Dabei ist das Lerntempo durch die Anzahl von Lernepisoden definiert, die nötig sind, bis der Agent gelernt hat, mit adäquaten Kosten eine Strecke zwischen Start und Ziel zu bauen. Die verursachten Kosten je Episode sollten sich in der Entwicklung des Lernprozesses einem durchschnittlichen Minimum annähern. Die Konvergenzgenauigkeit und Stabilität ist ein Qualitätskriterium. Ein weiteres Qualitätsmerkmal ist die Genauigkeit der Kostenschätzung für alle Aktionsalternativen eines Zustands. Das geht über das bloße Erlernen der jeweils günstigsten Aktion je Zustand hinaus und ist daher erst einmal zweitrangig. Mit Hinblick auf den Einsatz des eigentlichen Lernverfahrens im Laufroboter AMEE, verdient es trotzdem eine genauere Betrachtung. Denn in Kombination mit dem Lauf-Kontroller (siehe [Ruh14]) sind die Kostenschätzungen je Zustand ein Merkmal unter anderen, die zur letztlichen Trittpositionswahl führen. Eine schlechte Kostenschätzung der Alternativen könnte den Entscheidungsprozess ungünstig beeinflussen.

5.5.2 Testbedingung

Um die Auswirkung unterschiedlicher Lernparameter in den jeweiligen Experimenten vergleichen zu können, ist es notwendig, für vergleichbare Testbedingungen zu sorgen, indem jeweils dieselbe oder eine äquivalente Aufgabe gestellt wird. Das heißt, die zu bauende Strecke sollte von Start zum Ziel jeweils dieselben minimalen Kosten haben. Die Hexdistanz ist dabei irrelevant, da die Streckenkosten im Wesentlichen durch die Landschaftsmerkmale beeinflusst werden. Ein vollständiger Test über alle 64 Millionen Zustände wäre sehr ressourcenintensiv. Um eine tendenzielle Aussage über die Einflüsse der Parameter zu erhalten ist es jedoch hinreichend, einige Stichproben zu einer Welt, bestehend aus einer kleinen Untermenge der möglichen Zustände, zu nehmen. **Abb. 5.12** zeigt zwei für diesen Zweck zufällig generierte Karten mit 9 x 10 Feldern, was jeweils 56 möglichen Zuständen entspricht. Der Aktionsradius des Agenten ist durch Wasserfelder am Rand der Karte beschränkt.

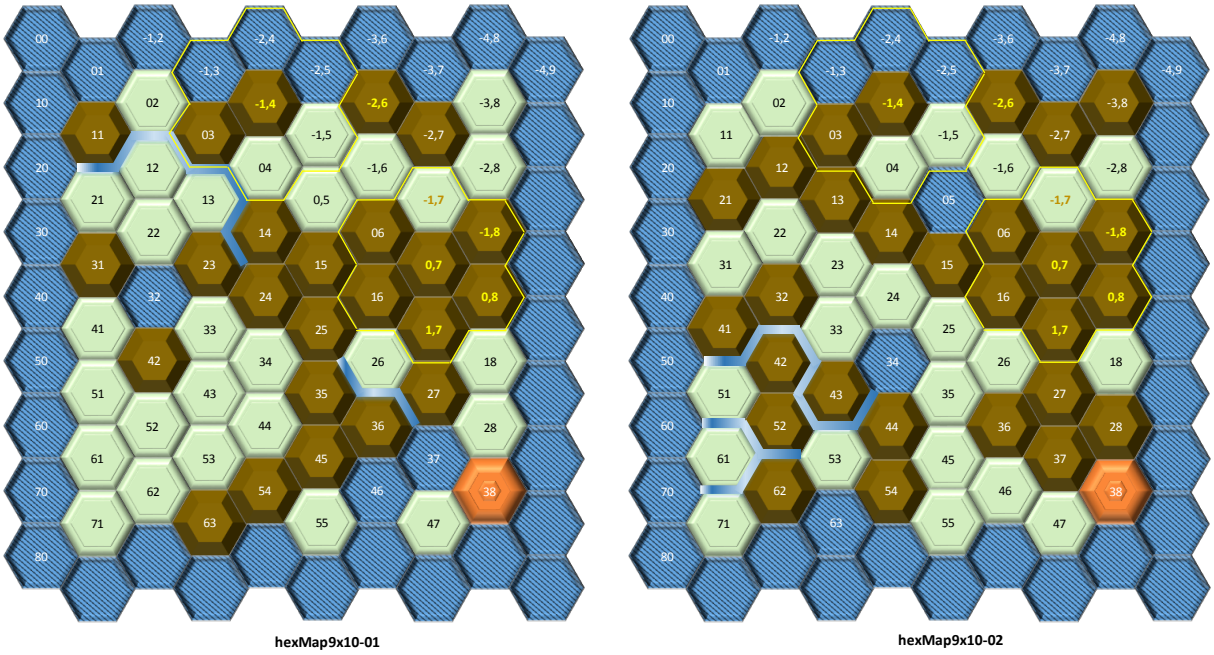


Abb. 5.12: Testwelten für das Dampfross®-Szenario. Zufällig generiert. Einige Koordinaten, z.B. (-1,4) und (0,7), werden auf beiden Karten jeweils auf denselben RL-Zustand abgebildet. Zielkoordinate ist immer (3,8).

Der Einfluss der Lernparameter wird an der *hexMap9x10-02* (rechte Karte in **Abb. 5.12**) getestet. Für gleiche Startbedingungen bei jeder Lernepisode sorgt die Wahl der Startkoordinate aus einer von sechs Feldern[32], die alle jeweils minimal elf Wegkosten zum stets selben Ziel auf (3,8) verursachen. Dies bietet einen Kompromiss zwischen Vergleichbarkeit der Lernepisoden und der tendenziellen Beschleunigung des Lernprozesses durch die *Exploring Starts*[33] Strategie [Sut12]. Die Schätzfunktion Q(s,a) wird bei jedem Testdurchlauf mit Null für alle Zustands-Aktions-Paare initialisiert und als Aktionswahl-Strategie nutzt das Experiment ε-Greedy, welche mit der Wahrscheinlichkeit 1-ε die jeweils (vermeintlich) kostengünstigste Aktion wählt und eine zufällige Alternative sonst.

Das Äquicausa Problem (siehe **5.1.3**) wird analysiert, indem der Agent abwechselnd eine Lernepisode in der hexMap9x10-01 und eine in der hexMap9x10-02 vollzieht. An einigen Koordinaten – von denen zwei in **Abb. 5.12** hervorgehoben sind – ist die unmittelbare Umgebung bei beiden Karten identisch, jedoch mit unterschiedlichen Minimalkosten zum Ziel. Neben der allgemeinen Lernentwicklung der optimalen Strategie sind die Kostenschätzungen speziell dieser gemeinsamen Zustände für das Äquicausa Problem von Interesse.

[32] Startkoordinaten sind: (-1,5), (-1,7), (-2,8), (4,1), (5,1), (7,1)

[33] Indem der Agent bei jeder Episode in einem anderen Startzustand beginnt, besucht er potentiell mehr Zustände, als bei einem fixen Startzustand und gewinnt früher ein breiteres Erfahrungsspektrum.

Für die Analysen bezüglich kontinuierlicher Dimensionsräume wird die Karte hexMap9x10-02 angepasst, indem jede Kachel einmalig einen zufälligen, reellen Basiswert zwischen 0,0 und 0,1 erhält. Je nach aktueller Position und Perspektive des Agenten auf eine Kachel, errechnet sich dann das Landschaftsmaß aus Landschaftstyp und Flusskanten durch Addition entsprechender Konstanten (siehe **Tabelle 5.2**), sodass eine Abbildung der ursprünglich diskreten Karte auf reelle Werte gemäß **Tabelle 5.1** vollzogen wird. Wie im eigentlichen AMEE-Szenario kann man sich den reellen Wert dann als ein Unebenheits- oder Schwierigkeitsmaß vorstellen.

Tabelle 5.2: Berechnung des reellen Maßes anhand von Basiswert und Kachelmerkmalen

Kachel Merkmal	Berechnung
Basiswert	in [0.0, 0.1[
Ebene (oder Stadt)	+0.0
Berg	+0.4
Wasser	+0.8
Flusskante zum linken Nachbarn	+0.1
Flusskante zum Zentrum	+0.2

5.5.3 Kennzahlen

Lernentwicklung

Die Entwicklung eines Lernprozesses kann an den Kosten je Episode – dem Gesamt-*Reward* – beobachtet werden. Dabei sind teils erhebliche Schwankungen der Werte zwischen aufeinander folgenden Episoden ein natürliches Verhalten des Lernalgorithmus, gerade in der Anfangsphase. Für eine aussagekräftige Lernkurve ist es daher sinnvoll, diese Streuung zu glätten, ohne den Trendverlauf dabei zu verdecken. Bei Anne Wolter [Wol08] zeigt sich eine gute Glättung, indem sie die Ergebnisse von jeweils zehn Episoden zu einem Mittelwert bündelt, was für die hier folgenden Experimente so über-nommen wird. Zusätzlich zu der Streuung innerhalb eines Testdurchlaufs kommt es aufgrund diverser Zufallsentscheidungen des Agenten auch zu unterschiedlichen Verläufen bei mehrmaliger Wieder-holung eines Tests. Aus diesem Grund besteht ein Experiment aus mehreren Testdurchläufen, bei denen die Q-Funktion jedes Mal neu mit Null initialisiert wird, der Agent somit ohne Vorwissen startet. Am Ende des Experiments werden aus den korrespondierenden, gemittelten Kosten je Durchlauf abermals Durchschnittswerte berechnet, welche letztlich die Kennzahlen der Lernkurve ergeben.

Zusammenfassend besteht ein Experiment aus $\mathbf{P}$ Testdurchläufen (engl.: **p**ass) mit jeweils $\mathbf{E}$ Episoden. In jedem Durchlauf werden von je zehn Episoden die Gesamtkosten (engl.: Total **R**eward) gemittelt. Dabei steht $\mathbf{\varnothing R_{pb}}$ für die durchschnittlichen Gesamtkosten der zehn Episoden im b'ten Bündel (engl.: **b**undle) und im p'ten Durchlauf. Über alle P Durchläufe hinweg, ergibt der Mittelwert der Bündel-Mittelwerte die Kennzahl C_b. Aus den Kennzahlen C_b wird die Lernentwicklungskurve interpoliert, die

die mittleren Kosten je Episodenbündel wiedergibt. **Tabelle 5.3** zeigt eine schematische Darstellung der Kosten, sowie der daraus resultierenden Mittelwerte und Kennzahlen.

Tabelle 5.3: Schematische Tabelle zur Auswertung der erzielten Gesamtkosten in einem Experiment

Pass: / Episode	Bundle	1		…	P		Classif. Rew. Number
1	1	R1	øR11	øRp1	R1	øRP1	C1 = ø(øRp1)
…		…			…		
10		R10			R10		
11	2	R11	øR12	øRpb	øRPb		Cb = ø(øRpb)
…		…					
20		R20					
	b		øR1b				
E-9	B = E/10	RE-9	øR1B	øRpB	øRPB		CN
…		…					
E		RE					

Lernqualität

Als Qualitätsmaß des Erlernten wird hier die Genauigkeit der Kostenschätzung über jeweils alle Aktionsalternativen eines Zustands (Sichtfeldes) herangezogen. Diese wird wiedergegeben durch die mittlere quadratische Abweichung (**MSE**) aller Q-Funktionswerte in Relation zu den tatsächlichen minimalen Kosten in der Testwelt. Die Minimalkosten für einen Irrweg setzten sich dabei aus den Streckenkosten für die erste Aktion und den Minimalkosten des Folgezustands zusammen. Da die Karten nur eine kleine Untermenge aller möglichen Zustandsausprägungen anbieten, wird die MSE auch nur über die im Experiment erreichbaren Zustände gebildet. Des Weiteren wird mit den MSE-Werten verfahren wie mit den Gesamtkosten zur Analyse der Lernentwicklung (vgl. **Tabelle 5.3**) und erfolgt parallel innerhalb eines Experiments, sodass je Experiment eine Kurve zur Lernentwicklung und eine zur Lernqualität entstehen.

5.6 Zusammenfassung

Die Lernaufgabe zur Auswahl sicherer Schrittpositionen wurde inspiriert durch das Brettspiel Dampfross® zu einer äquivalenten Aufgabe weiter abstrahiert, sodass klar definierte Testumgebungen erzeugt werden konnten, bei denen die optimale Schätzfunktion jeweils bekannt ist und mit den

Lernergebnissen verglichen werden können. Mittels des diskretisierten Testmodells kann im ersten Schritt die Parametrisierung des Sarsa(λ) Algorithmus und das Lernverhalten für gleiche Zustände, die sich in unterschiedlichen Umwelten befinden (Äquicausa Problem), eruiert werden. Für den Umgang mit kontinuierlichen Dimensionen durch Tile Coding wurde das diskrete Dampfross Modell in ein äquivalentes, kontinuierliches überführt, sodass eine Vergleichbarkeit mit den ersteren Analysen zur Parametrisierung gegeben ist und die Auswirkungen von Tilings Organisationen und Dimensionskodierungen untersucht werden können. Als Kennzahlen für die Analyse werden einmal die Gesamtkosten herangezogen, deren Entwicklung Auskunft über die Qualität der übergeordneten Strategie gibt. Zusätzlich liegt hier ein besonderes Augenmerk auf die mittlere Abweichung der Einzelabschätzungen von Zustands-Aktions-Paaren – die Basis aufgrund derer die Aktionsentscheidungen fallen und sich die Strategie herausbildet – und die Auskunft über die Genauigkeit der Einzelabschätzungen gibt.

6 Versuchsergebnisse & Analyse

Auf Basis des abstrahierten Dampfross® Testmodells werden für die Lernaufgabe zur Wahl sicherer Schrittpositionen eines vierbeinigen Roboters zuerst die Sarsa(λ) Lernparameter untersucht und bezüglich dieser Aufgabe optimiert. Die Resultate der ersten Untersuchungen werden anschließend als Grundeinstellung für die Experimente zur Tilings-Organisation im reellen Zustandsraum und zu Varianten der Merkmalskombination übernommen.

6.1 Lernparameter für Sarsa(λ)

Die Ergebnisse Anne Wolters [Wol08] zur Analyse der RL-Lernparameter im Kontext einer Navigationsaufgabe werden hier als Referenz und Vorlage herangezogen. Ihre zusammenfassenden Trendaussagen für eine gute Konvergenz der Lernentwicklung sind:

- Niedrige Explorationsrate $\varepsilon \in [0.01;\ 0.1]$
- Hohe Diskontierung $\gamma \geq 0.9$
- Mittlere Lernrate α und mittlerer Spurzerfallsfaktor λ
- Verschiede weitere Tendenzen speziell für Navigationsaufgaben

Im Folgenden werden der Einfluss der Lernparameter in der diskreten Zustandswelt des Dampfross®-Szenarios analysiert und mit den Trendaussagen von Anne Wolter verglichen. Da es sich bei diesem Szenario nur indirekt um eine Navigationsaufgabe und stattdessen mehr um eine Klassifizierungsaufgabe handelt, sind die Ergebnisse aus [Wol08] wahrscheinlich nicht vollständig übertragbar und bedürfen einer Überprüfung. Für die Basiskonfiguration der folgenden Tests wird, basierend auf oben genannten Empfehlungen, $\alpha = 0.5,\ \lambda = 0.7\ \gamma = 0.9$ und $\varepsilon = 0.1$ gewählt. Als Testwelt dient hexMap9x10-02 (**Abb. 5.12**, zweite Karte). Zu Beginn eines jeden Testdurchlaufs wird die Q-Funktion mit Null initialisiert und damit zuvor erworbenes Wissen gelöscht. Die ermittelten Kennzahlen werden zu Lernentwicklungs- beziehungsweise Lernqualitätskurven interpoliert. Die gezeigten Kurven finden sich in größerer Darstellung nochmals im Anhang **B.1**.

6.1.1 Explorationsrate ε

Da die Explorationsrate in der ε-Greedy Strategie den Anteil „dummer“ Aktionen angibt – die nicht der gerade am besten bewerteten – ist der Konvergenzwert der Streckenkosten erwartungsgemäß umso höher, je größer Epsilon ist. Verzichtet man hingegen gänzlich auf Erkundung ($\varepsilon = 0$), erreicht die Q-Funktion relativ früh ein lokales Minimum, das allerdings weit vom Optimum entfernt ist – die Lernkurve konvergiert gegen sehr hohe Streckenkosten. Ein besonderes Augenmerk dieser Untersuchung liegt auf dem von Wolter empfohlenen Bereich von 0,01 bis 0,1. Hier bestätigt sich, dass eine niedrige Erkundungsrate zwischen 0,01 und 0,05 die günstigste Lernentwicklung zeigt. Die Lernkurve konvergiert relativ stabil und nahe des Optimums.

Bezüglich der Lernqualität über alle Funktionswerte zeigt sich jedoch ein zur Lernentwicklung gegensätzliches Verhalten. Je höher die Erkundungsrate ist, desto genauer ist die Gesamteinschätzung.

Beide Lernkurven zeigen einen relativen stabilen und konstanten Verlauf – eine Konvergenz – ab etwa dem zehnten Episodenbündel. Ein Wissensgewinn findet kaum noch statt und man könnte das Lernen eventuell einstellen. Eine Strategie mit ε-Reduktion im Laufe des Lernprozess wurde hier nicht erwogen, da Wolter und andere zeigten, dass es dadurch keine signifikante Verbesserung gibt und sie die Konvergenz sogar verlangsamen kann.

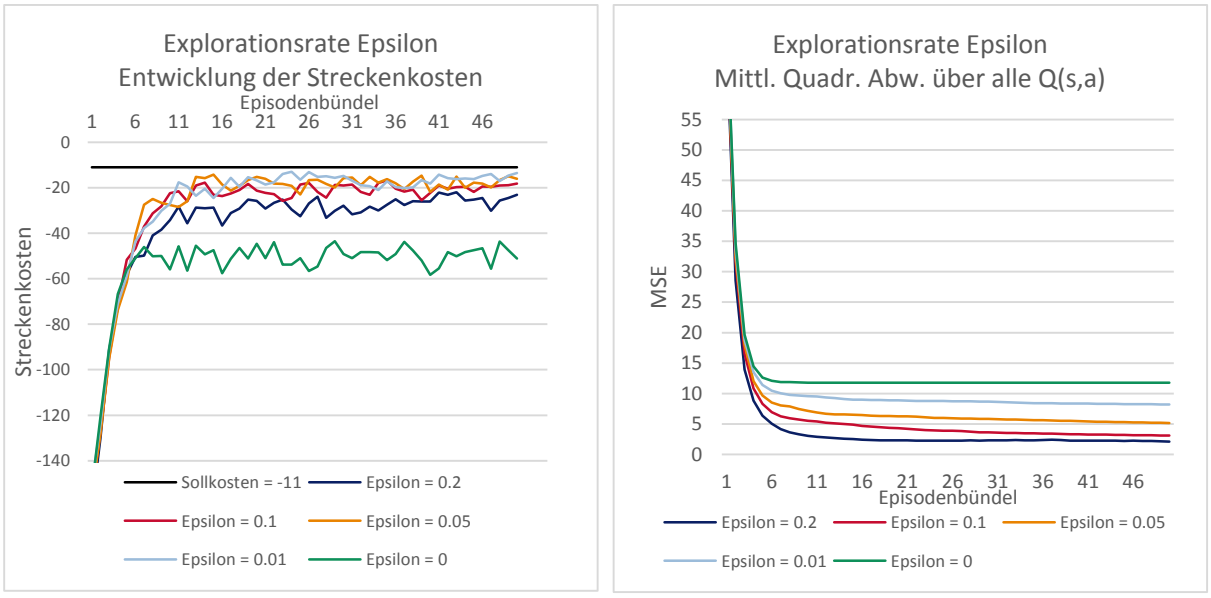


Abb. 6.1: Lernentwicklung und -qualität für Epsilon. (α=0.5, γ=0.9, λ=0.7, je 10 Episoden pro Bündel)

Die Anforderungen an die Explorationsrate, sowohl die günstigste Aktion je Zustand zu erlernen als auch eine korrekte Einschätzung des Gesamtzustands mit allen Aktionsalternativen geben zu können, sind hier widersprüchlich. Es ist ein Kompromiss nötig, der für die weiteren Untersuchungen auf $\varepsilon =$

0,03 festgelegt wird. Generell zeigt sich, dass der Agent größtenteils auf seine Erfahrung vertrauen und nur selten (etwa alle 33 Aktionen) mal was Neues probieren sollte.

6.1.2 Lernrate α

Bei der Lernrate Alpha, die angibt, in welchem Maße neu erworbene Erkenntnisse in die Wissensbasis – die Erfahrung – einfließen, werden die Ergebnisse von Wolter zur Lernentwicklung wieder weitgehend bestätigt. Ebenso die intuitive Vermutung, dass bei einem geringen Wissensstand zu Beginn des Lernprozess eine hohe Lernrate zu schnellem Wissensgewinn führt, während bei einem reichhaltigen Erfahrungsschatz eine Feinjustierung durch niedrige Lernraten besser ist, als sie durch neue Erlebnisse Großteils zu überschreiben.

Lernraten zwischen 0,3 und 0,7 – ein recht breites Intervall – zeigen vergleichsweise ähnlich schnelle Lernentwicklungen in der Anfangsphase. Eine gute Konvergenz bieten hingegen Werte ab 0,35 abwärts. Bei der Gesamtqualität ergibt sich ein eindeutiges Bild zugunsten hoher Lernraten. Zwar konvergieren alle Qualitätskurven relativ früh, jedoch scheint sich der anfängliche Vorteil einer hohen Lernrate auch auf die Einschätzung der Irrwege, die gerade in der Anfangsphase häufiger begangen werden, positiv auszuwirken, sodass die MSE auf einem niedrigeren Niveau konvergiert.

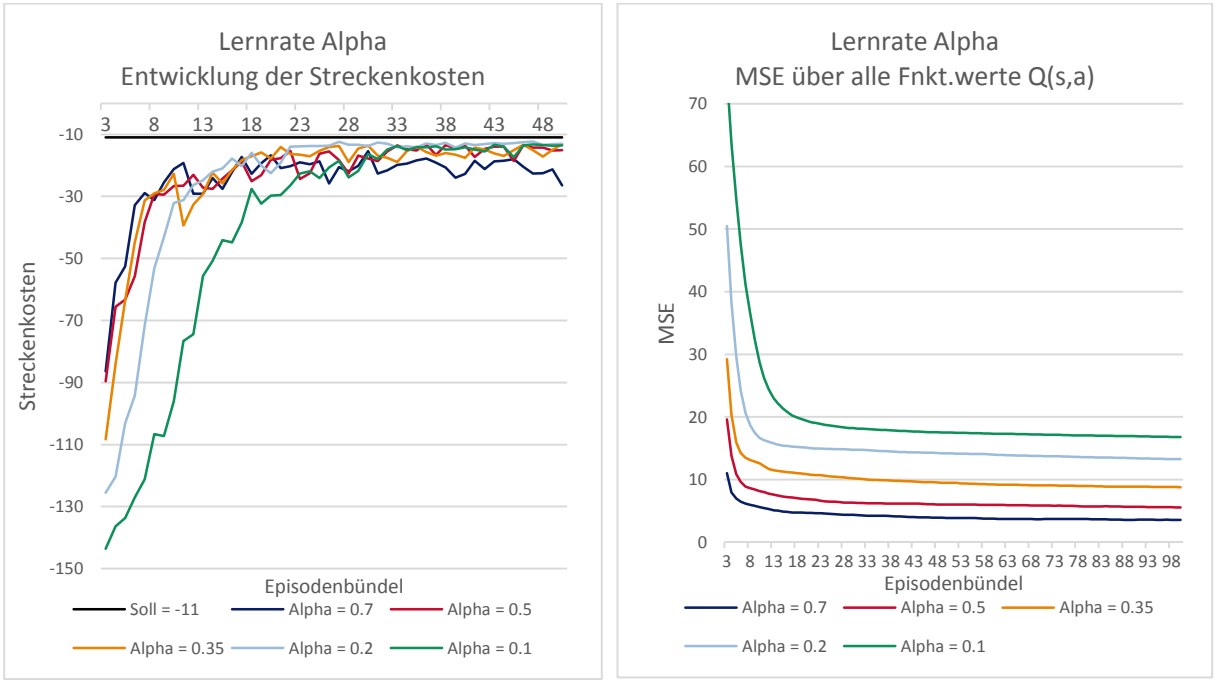


Abb. 6.2: Lernentwicklung und -qualität für Alpha. (ε=0.03, γ=0.9, λ=0.7, je 10 Episoden pro Bündel)

Wie auch bei der Explorationsrate herrscht hier wieder ein Gegensatz zwischen dem Anspruch der Konvergenzgenauigkeit und dem Anspruch auf Vollständigkeit des Wissens. Eine Alpha-Reduktion

abhängig davon, wie oft ein Zustand besucht wurde, könnte den Vorteil einer hohen Lernrate bezüglich der anfänglichen Lernentwicklung sowie der Wissensvollständigkeit und dem Vorteil niedriger Lernraten bezüglich Konvergenz der Kostenminimierung vereinen. Hierzu müsste in jedem Zustand zusätzlich die Anzahl der Vorkommnisse festgehalten werden.

Für die weiteren Untersuchungen wird hingegen ein konstantes $\alpha = 0{,}3$ als Kompromiss gewählt. Der Wert bietet eine vergleichsweise schnelle Lernentwicklung und gute Konvergenz, während die MSE sich im Mittleren Bereich bewegt.

6.1.3 Diskontierung γ

Die hier gestellte Aufgabe ist episodisch, da sie stets bei Erreichen der Zielkoordinate oder spätestens nach einer festen Anzahl von Aktionen beendet wird. Da Gamma eigentlich dazu dient, bei nicht-episodischen Aufgaben die Schätzung über zukünftige Kosten zu begrenzen, ist es offensichtlich, dass ein Wert von 1,0 das beste Ergebnis zur Lernentwicklung und ihrer Konvergenz liefert. Für diesen Lernparameter gilt dieselbe Aussage auch für die MSE, also die Lernqualität. Sehr viel deutlicher als bei der Navigationsaufgabe Wolters zeigt sich hier eine eklatante Verschlechterung schon ab $\gamma \leq 0{,}9$. Folgend wird der Diskontierungsfaktor auf 1,0 gesetzt.

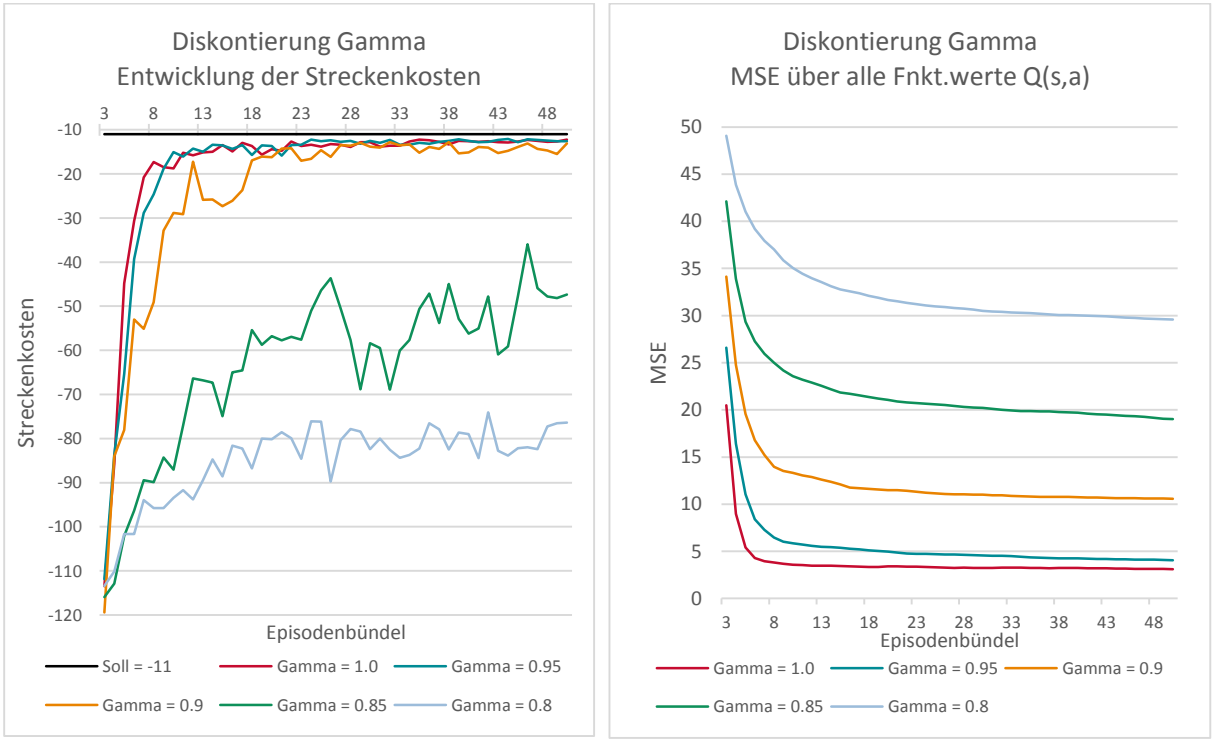


Abb. 6.3: Lernentwicklung und -qualität für Gamma. (α=0.3, ε=0.03, λ=0.7, je 10 Episoden pro Bündel)

6.1.4 Spurzerfall λ

Durch den Spurzerfallsfaktor wird bestimmt, in welchem Maße neue Erfahrungen an frühere Zustandsaktionspaare zurückgereicht werden. Ab Lambda = 0 (keine Rückwärtssicht) zunehmend, steigt sowohl das Lerntempo als auch die Lernqualität bis Lambda = 0,7. Ab Lambda = 0.8 wird es allerdings wieder schlechter, was sich stärker noch als bei der Lernentwicklung bei der Lernqualität zeigt. Nach einer anfänglichen guten Entwicklung des mittleren Fehlers, steigt der MSE ab etwa der 40'ten Episode wieder und bleibt auf hohem Niveau. Dieser Effekt verstärkt sich dann, je weiter sich Lambda der 1,0 nähert.

Tatsächlich bestätigt dies das als „Alpha-Lambda Abhängigkeit" bekannte Verhalten [Wol08]. Das beste Ergebnis wird hier durch die zuvor gewählte Standardkonfiguration von λ = 0,7 erzielt, was vermutlich da herrührt, dass die Lernrate unter anderem auf Basis dieses Wertes ermittelt wurde.

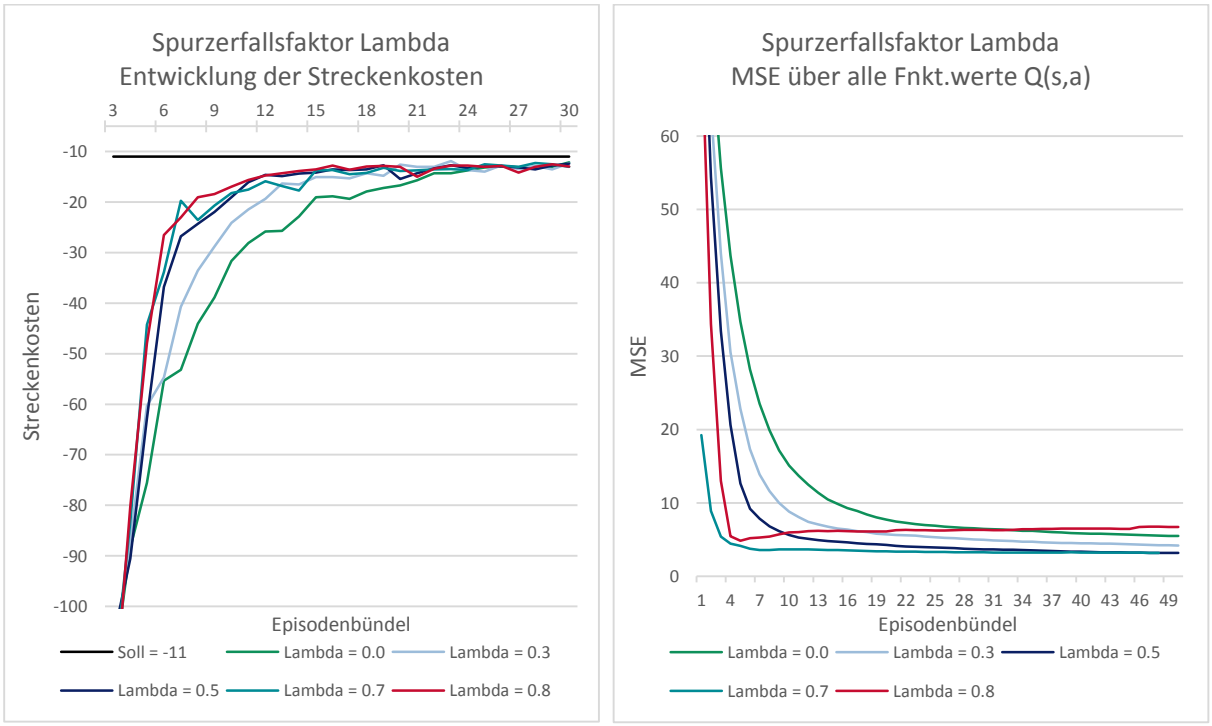


Abb. 6.4: Lernentwicklung und -qualität für Lambda. (α=0.3, ε=0.03, γ=1.0, je 10 Episoden pro Bündel)

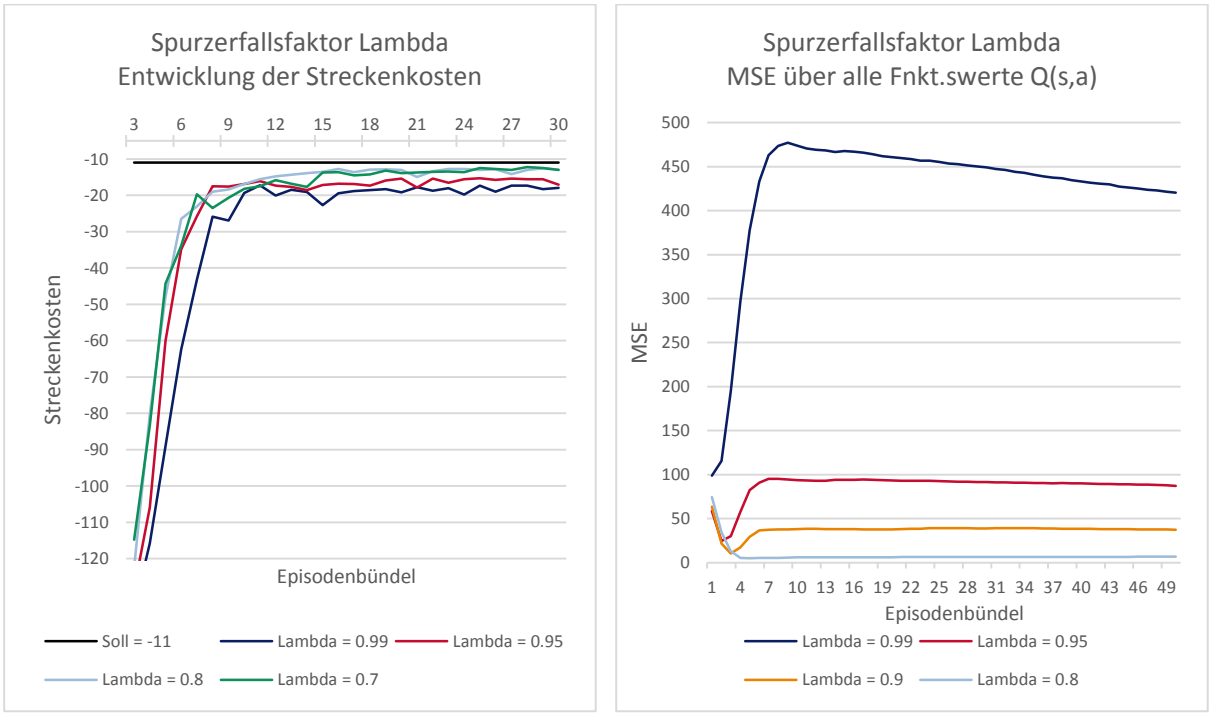


Abb. 6.5: Lernentwicklung und -qualität für Lambda. Deutliche Verschlechterung für $\gamma \geq 0,8$

6.1.5 Zusammenfassung Lernparameter

Im Allgemeinen haben die Tests gezeigt, dass sich der Einfluss der Lernparameter auf die Lernentwicklung auch bei der sich zu Wolter unterscheidenden Aufgabenstellung weitgehend wie vorausgesagt verhält. Die auf Basis der Empfehlungen von A. Wolter erstellte Anfangskonfiguration konnte im Detail optimiert werden wobei zusätzlich auf Vollständigkeit des Wissens auch bezüglich Aktionsalternativen Wert gelegt wurde. Als Resultat der Analyse werden künftige Untersuchungen mit folgenden Lernparametern durchgeführt:

- Lernrate $\alpha = 0,3$
- Diskontierung $\gamma = 1,0$
- Spurzerfallsfaktor $\lambda = 0,7$
- Explorationsrate $\varepsilon = 0,03$

Insgesamt sind diese Werte als ein eher konservatives Lernverhalten zu interpretieren, bei dem mehr auf bereits erworbenes Wissen vertraut und tendenziell wenig experimentiert wird. Ein gewisser Grad an Weitsicht ist dem Agenten mitgegeben, der Einfluss vorangegangener Aktionen wird mittelstark und die geschätzten künftigen Kosten hundertprozentig berücksichtig. **Abb. 6.6** zeigt den Vergleich der anfänglichen und optimierten Lernkurven.

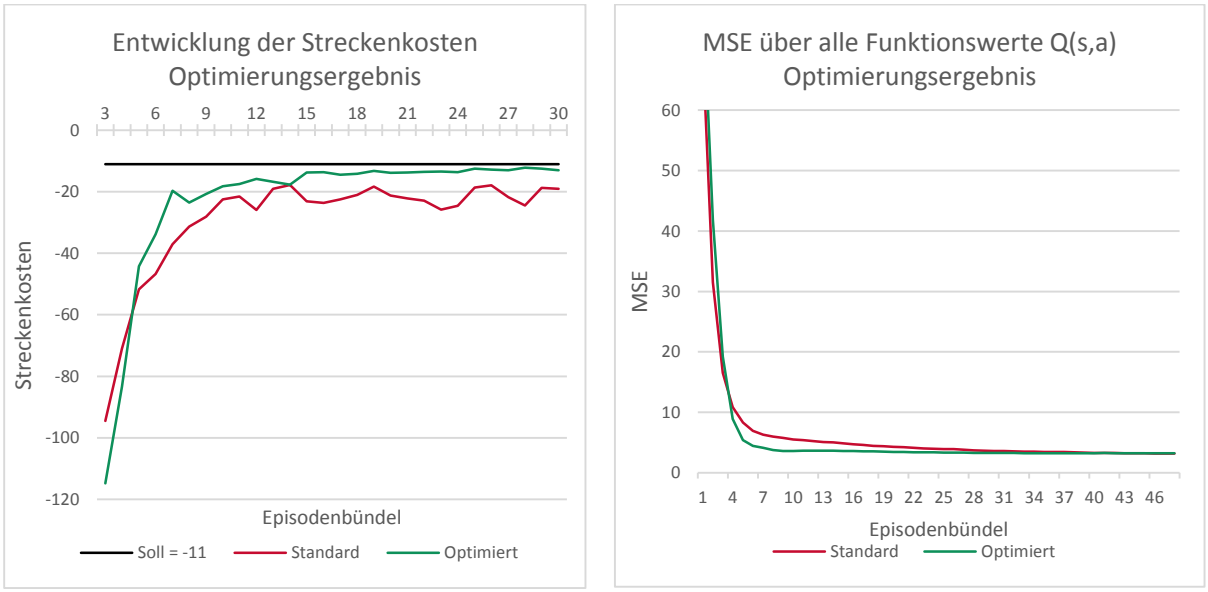


Abb. 6.6: Analyse der Lernparameter. Vorher-Nachher Vergleich.

6.2 Äquicausa Problem

Die in **5.1.3** beschriebene Herausforderung, dass das gleiche Sichtfeld mit denselben, zustands-definierenden Umweltbedingungen teilweise unterschiedlich bewertet werden müsste, sei hier als Äquicausa Problem benannt (siehe auch **Behauptung 2**). Es tritt im Dampfross Szenario beispielsweise auf, wenn auf zwei unterschiedlichen Karten identische Sichtfelder mit derselben Distanz und Richtung zum jeweiligen Ziel vorkommen, die Landschaften zwischen aktueller Position und Ziel und damit einher-gehend die Wegkosten aber variieren. Es wurde die Behauptung aufgestellt, dass sich im Zuge des Lernprozesses ein Mittelwert aus den einzelnen Kostenschätzungen einstellt, betrachtete man die Welten jeweils für sich allein. Um dies zu belegen, lässt das Äquicausa Experiment den Agenten bei jeder Lernepisode abwechselnd in einer der beiden Welten *hexMap9x10-01* und *hexMap9x10-02* laufen (siehe **Abb. 5.12**). An insgesamt sieben Koordinaten sind die Umgebungsbedingungen identisch. Für den Agenten ist es an diesen Positionen derselbe Zustand, egal in welcher Welt er sich tatsächlich befindet, allerdings unterscheiden sich jeweils die Minimalkosten. Die Startkoordinaten werden bei jeder Episode zufällig aus einer Auswahl von Koordinaten mit minimalen Wegkosten von zwölf Einheiten gewählt. Ob sich der Mittelwert einstellt, lässt sich am MSE-Maß feststellen, der über die Q-Funktionswerte aller Zustands-Aktionspaare der sieben gemeinsamen Zustände gebildet wird. Als optimalen Sollwert wird dem Experiment der zuvor errechnete Mittelwert der Sollwerte je Karte übergeben.

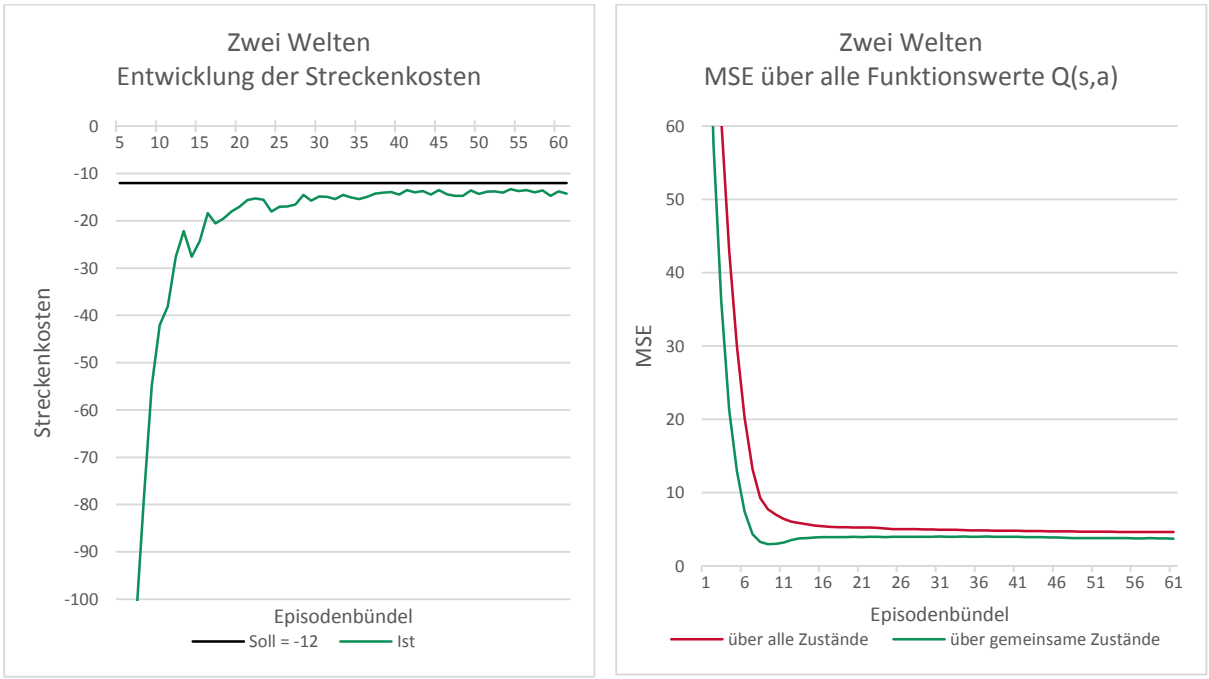


Abb. 6.7: Äquicausa Problem - Gesamtbewertung von Zuständen, die in unterschiedlichen Welten mit unterschiedlicher Einzelbewertung vorkommen.

Die Mittlere Quadratische Abweichung der Funktionswerte $Q(s,a)$ vom Optimum Q^* – den zuvor errechneten Sollwerten – konvergiert im Experiment gegen einen Wert zwischen drei und vier, sowohl über alle als auch über die gemeinsamen Zustände der beiden Welten (siehe **Abb. 6.7**). Das Experiment zeigt eine annähernd gleiche Entwicklung der Lernqualität, wie zuvor in einer einzigen Welt mit den optimierten Lernparametern (siehe **Abb. 6.6**). Die ein wenig langsamere Entwicklung zur Konvergenz, auch bei den Streckenkosten, rührt von der aus Agentensicht größeren Welt her – durch den Wechsel zwischen den beiden Karten sind fast doppelt so viele Zustände zu besuchen, wie in den Experimenten zuvor. Insbesondere bestätigt die Lernqualitätskurve, dass sich für gleiche Zustände bei variierendem Kontext ein Mittelwert für die Kostenschätzung einstellt und folglich, dass die Modellierung der Lernaufgabe geeignet ist, ohne Einbußen der Lernqualität mit multiplen Umwelten umzugehen. Bildlich gesprochen ist es egal, ob die Rasenfläche auf der man gerade steht zum Münchener Englischen Garten oder dem Hamburger Stadtpark gehört. Die gemachten Erfahrungen sind auf beide Orte anzuwenden.

6.2.1 Hypothese zum Äquicausa Problem für Laufroboter

Angenommen es gäbe eine in vielen verschiedenen Umwelten häufig vorkommende Umgebung mit einer stark streuenden optimalen Schätzung für die jeweilige Umwelt, ist die nützliche Verwertbarkeit eines Mittelmaßes mit hoher Wahrscheinlichkeit in Frage zu stellen. Bezogen auf das Zielszenario, die qualitative Einschätzung von Laufeigenschaften für ein bestimmtes Gelände, liegt aber die Vermutung nahe, dass auf bestimmte Bodentypen – beispielsweise ein englischer Rasen – ähnliche Typen folgen

oder zumindest eine begrenzte typische Variation von Bodenbeschaffenheiten zu erwarten sind. Vorausgesetzt, das nächste Etappenziel ist nicht kilometerweit entfernt, sondern beschränkt sich auf eine mittelbare Distanz, kann man von sich wiederholenden Geländecharakteristiken ausgehen, wodurch sich die Streuung der optimalen Einzelschätzungen im Rahmen hält.

6.3 Tiling für kontinuierliche Dimensionen

Das DampfrossContinuum Modell nutzt reellwertige Landschaftsmaße anstatt Landschaftstypen (siehe **5.4**). Mittels Tile Coding (**2.3.4**, **4.2**) wird der kontinuierliche Wertebereich diskretisiert. Dabei gibt es bezüglich der Auflösung und Anzahl solcher Segmentierungen mehrere Varianten, der Auswirkungen auf die Lernentwicklung hier untersucht werden.

6.3.1 1 Tiling

Der erste Test zu kontinuierlichen Dimensionsräumen soll vorerst nur die funktionale Eignung der Softwaremodellierung von MultiTilings für Sarsa(λ) (vgl. **4.2.1**) demonstrieren und überprüfen, ob sich wider Erwarten Unterschiede in der Lernentwicklung zum bisherigen tabellarischen RL-Modell zeigen. Dabei wurde streng genommen bereits im diskreten Dampfross® Szenario ein Tiling der Entfernung vorgenommen, indem größere Hexdistanzen in Intervalle zusammengefasst wurden. Der wesentliche Unterschied zum diskreten Modell und das eigentliche Tiling in diesem Experiment betrifft jedoch das reelle Landschaftsmaß je Sichtfeldkachel, das vergleichbar mit dem Unebenheitsmaß im AMEE-Szenario ist (vgl. **4.1.1**). Die Tilings-Organisation beschränkt sich hier auf ein 1-Tiling (Unterteilung des Wertebereichs) mit einer Auflösung von 0,1 und somit neun Intervallen, die in diesem Fall genau den verschiedenen Kachelausprägungen im diskreten Dampfross® entsprechen (vgl. **Tabelle 5.1**). Bezogen auf das Fehler! Verweisquelle konnte nicht gefunden werden. sind die Parametrisierung und das Ergebnis der Tilings-Organisation:

Parameter:

- Range $r = 0{,}9$
- Basis $b = 3$
- Exponenten $n = 2, m = 0$

Organisation:

- Anzahl Tilings $\#T = 3^0 = 1$
- Anzahl Tiles $\#t = 3^2 = 9$
- Auflösung $\delta = \frac{0{,}9}{3^2} = 0{,}1$

Die aus dem Tiling resultierenden Merkmalsdimensionen eines Sichtfeldes werden untereinander sowie mit der Zielrichtung und -distanz konjunktiv kombiniert (vgl. **4.2.2**), sodass letztlich jeder Zustand genau eines von $120 \cdot 9^6$ möglichen Merkmalen besitzt, was bewusst genau der Anzahl möglicher Zustände in der tabellarischen Modellierung entspricht.

Die Testergebnisse (**Abb. 6.8**) zeigen praktisch deckungsgleiche Lernkurven zwischen dem diskreten und kontinuierlichen Modell und damit das aufgrund der äquivalenten Modellierung erwartete Verhalten. Insbesondere demonstrieren die Resultate aber, dass die Lernaufgabe an sich, das Sarsa(λ) Lernverfahren, und das (Multi-)Tiling der RLLibrary auch für reellwertige Dimensionen prinzipiell geeignet sind.

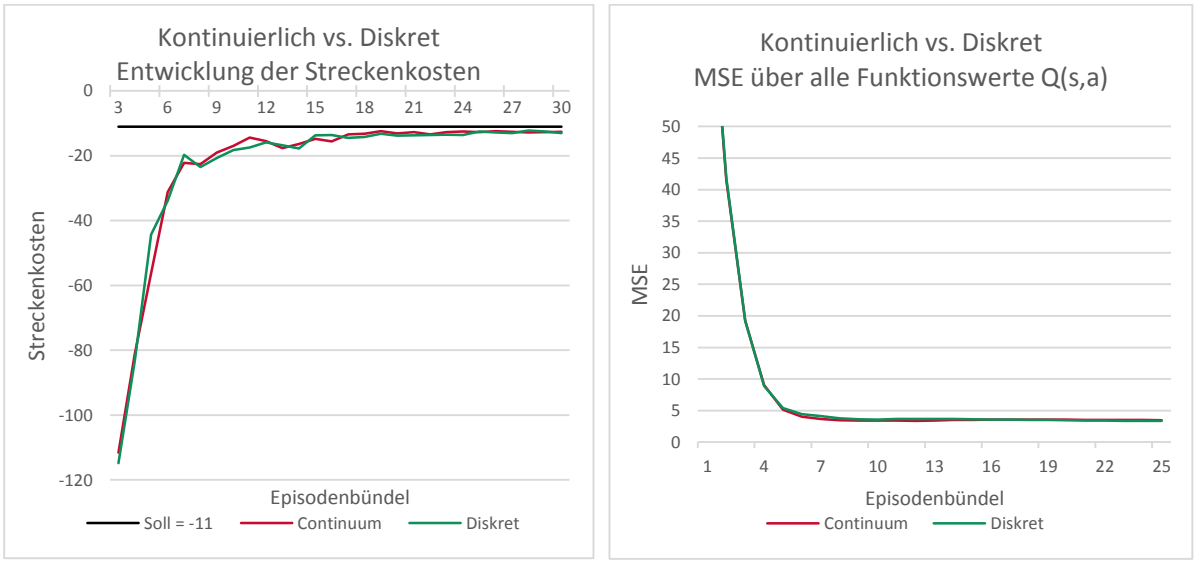


Abb. 6.8: Vergleich äquivalenter Zustandsmodellierungen für diskrete Zustände zum Einen und reellwertige Dimensionen zum Anderen.

6.3.2 Variation der Auflösung

Die Gesamtzahl möglicher Merkmale und somit der Ressourcenbedarf hängt bei der konjunktiven Dimensionsverknüpfung unter anderem von der Anzahl an Intervallen – der Tiles, in die der Wertebereich des Landschaftsmaßes aufgeteilt wird ab. Bei gleichbleibender Anzahl von Nachbarkacheln, Richtungen und Distanzen errechnet sie sich zu $120 \cdot \#t^6$ (vgl. **5.3**). Da die kontinuierliche Karte derart konstruiert ist, dass sie bei einer Aufteilung in neun Tiles mit einer resultierenden Auflösung von 0,1 für das Landschaftsmaß äquivalent zum diskreten Modell ist, stellt sich die Frage, wie sich das Lernverhalten bei einer gröberen Auflösung oder gar einer Überdefinition verhält. Das Delta-Experiment variiert hierfür die Auflösung des Wertebereichs zu drei zusätzlichen Tilings-Organisationen:

Tabelle 6.1: Auflösungen des Delta-Experiments

Auflösung	Anzahl Tiles	Merkmale gesamt
$\delta = \frac{0,9}{2^0} = 0,9$	$\#t = 2^0 =$ 1 Tile	7.680
$\delta = \frac{0,9}{3^1} = 0,3$	$\#t = 3^1 =$ 3 Tiles	87.480
$\delta = \frac{0,9}{3^2} = 0,100$	$\#t = 3^2 =$ 9 Tiles	63.772.920
$\delta = \frac{0,9}{2^4} = 0,05625$	$\#t = 2^4 =$ 16 Tiles	2.013.265.920

Abb. 6.9: HexMap14x16 für die MultiTiling Experimente

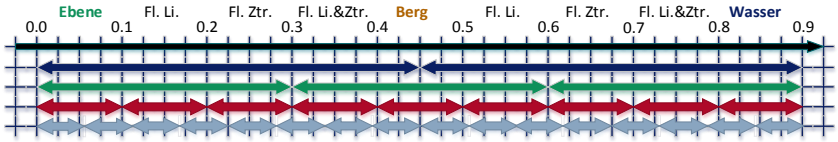


Abb. 6.10: 1-Tiling-Organisationen auf dem Zahlenstrahl. Rot: Die zum diskreten Modell äquivalente.

Auf der HexMap9x10-02 kommen im Verhältnis recht wenige Flussläufe vor, nur ca. 13% der Kanten sind Flüsse. Um die Wahrscheinlichkeit eines Flussvorkommens und somit dessen Einfluss auf die Kosten sowie die Auswirkung der Tilings-Organisationen zu erhöhen, wird das Delta-Experiment mit der dreifach größeren HexMap14x16, die etwa 26% Flusskanten hat, ausgeführt. Die Ergebnisse (**Abb. 6.11**) zeigen wenig überraschend, dass die überdefinierte Tilings-Organisation mit 16 Tiles nicht schneller oder genauer als mit neun Tiles zum Lernerfolg beiträgt. Tatsächlich ist sie auch nicht langsamer oder ungenauer, sondern verhält sich exakt gleich. Außer eines erhöhten Ressourcenbedarfs ergeben sich durch eine Überdefinition folglich keine Nachteile.

Bemerkenswert ist allerdings, dass selbst die erheblich gröbere Aufteilung mit drei Tiles keine signifikanten Unterschiede zur ursprünglichen 9-Tiles-Anordnung zeigt. Erst bei zwei Tiles – was praktisch nur noch eine Unterscheidung zwischen Ebene und Berg ohne Berücksichtigung der Flussläufe entspricht – konvergieren die Streckenkosten langsamer und weiter Entfernt vom Optimum, während von einer Lernqualität quasi nicht mehr die Rede sein kann.

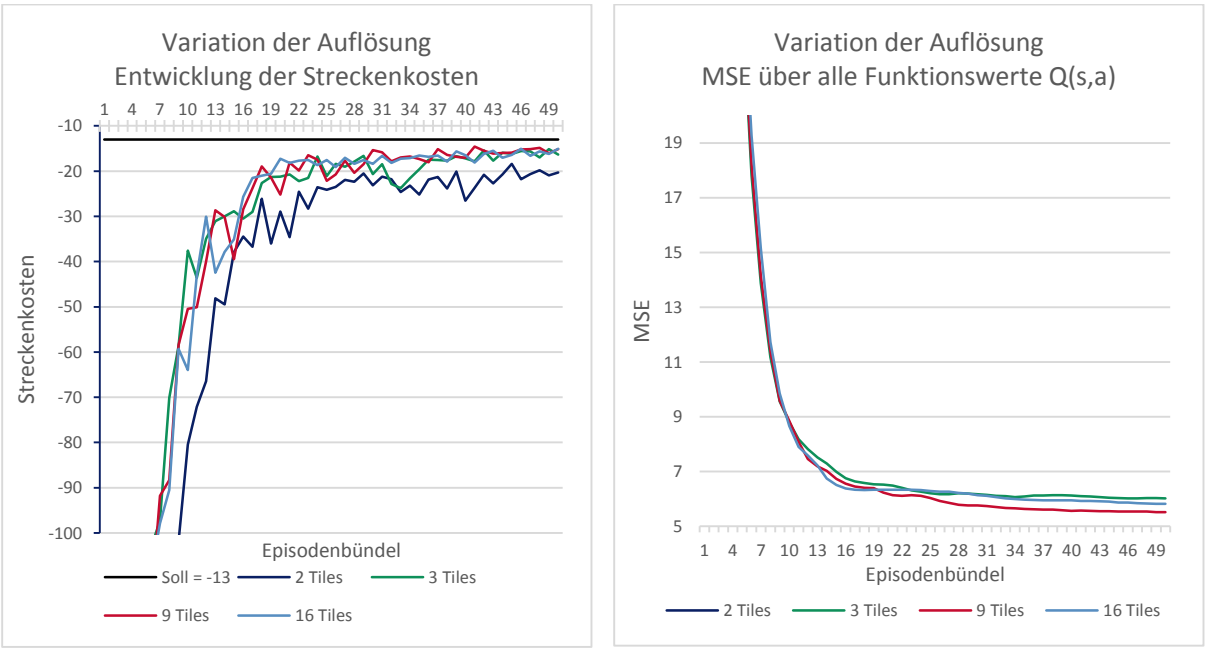


Abb. 6.11: Variation der Auflösung und implizit der Tiles-Anzahl bei 1 Tiling. Neun Tiles sind äquivalent zum diskreten Dampfross®-Modell. Die MSE-Kurve der 2-Tiles Anordnung liegt weit über dem hier abgebildeten Bereich.

6.3.3 MultiTiling

Tabelle 6.2: MultiTilings Organisationen mit Parametern nach Tiles-Verhältnis 3 Fehler! Verweisquelle konnte nicht gefunden werden.

MultiTiling	Parameter	Organisation
3x3 MultiTiling	Range $r = 0{,}9$ Basis $b = 3$ Exponenten $n = 2, m = 1$	Anzahl Tilings $\#T = 3^1 = 3$ Auflösung $\delta = \frac{0{,}9}{3^2} = 0{,}1$ Anzahl Tiles $\#t = 3^2 = 9$
2x4 MultiTiling	Range $r = 0{,}9$ Basis $b = 2$ Exponenten $n = 3, m = 1$	Anzahl Tilings $\#T = 2^1 = 2$ Auflösung $\delta = \frac{0{,}9}{2^8} = 0{,}1125$ Anzahl Tiles $\#t = 2^3 = 8$

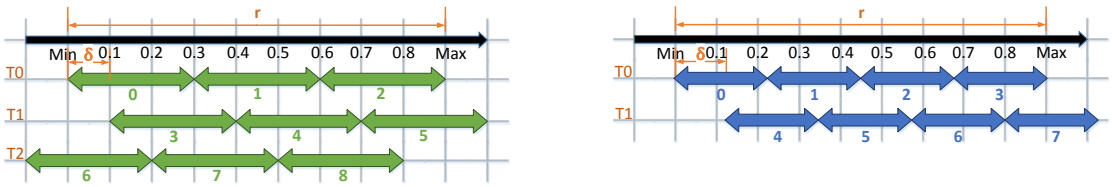


Abb. 6.12: MultiTilings Organisationen auf Zahlenstrahl abgebildet

Der in **2.3.4.3** und **4.2** beschriebene Vorteil von MultiTiling – der mehrfachen, leicht variierenden Unterteilung des Wertebereichs – liegt in einer höheren Generalisierung des Lernalgorithmus und damit einhergehend einer potentiellen Beschleunigung des Lernfortschritts in der Anfangsphase, allerdings auf Kosten der Approximationsgenauigkeit, was hier überprüft wird. Das MultiTilings Experiment vergleicht ein 2x4 und ein 3x3 Tiling mit dem Standard 1-Tiling. Die Organisationen (vgl. **Tabelle 6.2**) haben zwecks Vergleichbarkeit eine annähernd gleiche Auflösung und sind nach **Tiles-Verhältnis 3** parametrisiert, sodass vorsorglich die Bedingungen für ein adaptives MultiTiling (Sherstov et. al. [She05]) gegeben sind, das zwischen Tilings-Organisationen je nach Erfahrungsstand wechseln würde. Auf diese Weise lässt sich an diesem Experiment, ausgeführt mit der HexMap14x16 (**Abb. 6.9**), zeigen, in welchem Maße sich die Vor- und Nachteile des MultiTiling auf die hier betrachtete Lernaufgabe auswirken und ob ein adaptives Tiling lohnenswert wäre.

Jedoch zeigen die Ergebnisse nur in der aller frühesten Lernphase, die eh durch noch hohe Ungenauigkeiten geprägt ist, eine lediglich marginale Beschleunigung des Lernfortschritts, bei drei Tilings etwas stärker ausgeprägt als bei zwei Tilings. Umso signifikanter ist dafür die auf mittlere Sicht schlechtere Lernqualität (MSE-Kurve). Hier lässt sich an den Kurven gut erkennen, wie sich der anfängliche Vorteil ins Gegenteil kehrt und sich die Qualität mit der Zeit sogar verschlechtert. Der Qualitätsverlust ließe sich vielleicht mit einer adaptiven Tilings-Organisation ausgleichen. Allerdings lohnt sich der damit einhergehende Ressourcenmehraufwand nicht, sodass auf eine Implementierung eines adaptiven Verfahrens für die RLLibrary und weitere Experimente dazu gewissenhaft verzichtet werden kann.

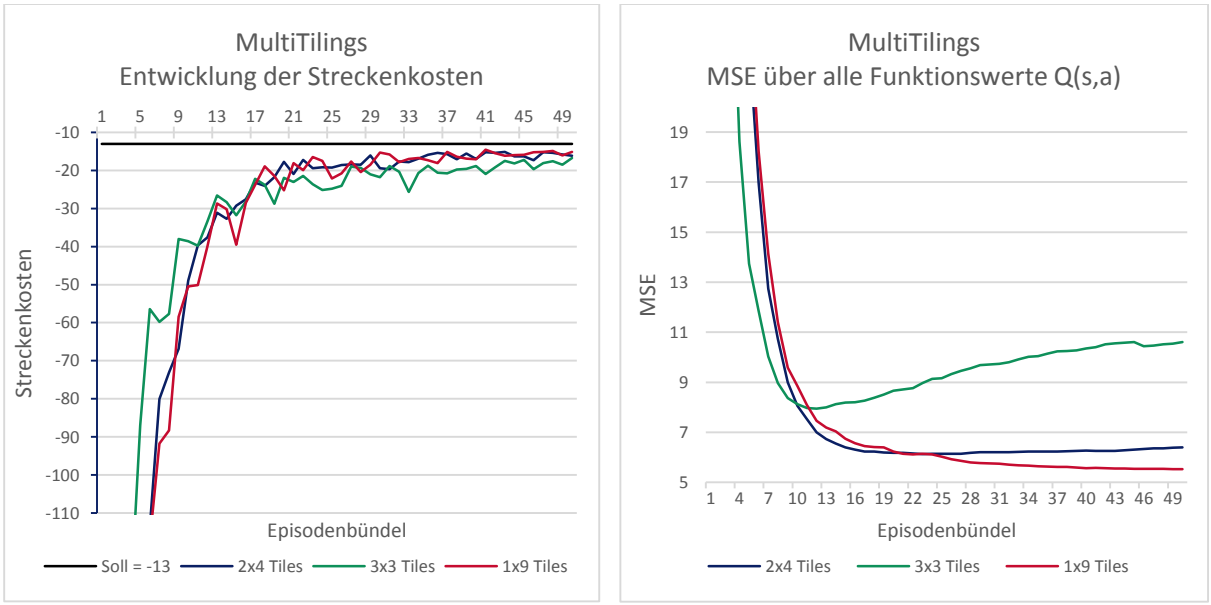


Abb. 6.13: Vergleich MultiTilings und 1-Tiling Organisationen bei je annähernd gleicher Auflösung

6.4 Merkmalskombinationen

Wie in **4.2.2** – Vektorgröße bei mehrdimensionalen Tilings – dargelegt, können die Zustandsdimensionen entweder konjunktiv oder disjunktiv miteinander verknüpft werden und **Behauptung 1** stellt die Behauptung auf, dass unter Umständen auch mit den stärker generalisierten disjunktiven Verknüpfungen ein gutes Lernergebnis zu erzielen ist. Daraufhin stellt Abschnitt **4.2.3** einige Kodierungsvarianten für die multidimensionale Tilings-Organisation des AMEE-Szenarios mit Ihrer Komplexität vor. Die Auswirkungen auf das Lernverhalten werden hier untersucht.

6.4.1 Kodierungsvarianten

Bei allen Kodierungsvarianten wird für die Landschaftsmaße der Nachbarkacheln ein 1-Tiling mit neun Tiles verwendet, wie es im **1-Tiling Experiment** eingeführt wurde. Insgesamt besitzt ein Dampfross®-Zustand neun Dimensionen: Sieben Kacheln inklusive des Zentrumfeldes im Blickfeld, die Zielrichtung und die Zieldistanz. In den vorangegangenen Experimenten wurden die Eingangsdimensionen als voneinander abhängig betrachtet und konjunktiv zu einem einzigen zustandsbeschreibenden Merkmal verknüpft. Man beachte den Unterschied zwischen dem MultiTiling und einer disjunktiven Dimensionsverknüpfung. Beim MultiTiling kann ein Zustand zwar auch mehrere Merkmale besitzen. Dies rührt aber von der mehrfach übereinander gelagerten Unterteilung des Wertebereichs der Dimensionen her. Dagegen wird hier die disjunktive Verknüpfung von Dimensionen eines mehrdimensionalen Zustandsraums betrachtet. Ein Zustand besitzt mehrere Merkmale, weil mehrere Dimensionen voneinander linear unabhängig sind.

Zur Angabe der Größe des Merkmalraums werden im Folgenden diese Größenangaben gemacht:

$\lvert ZMaß \rvert = 2$	Anzahl der Tiles für das Landschaftsmaß des zentralen Feldes.
$\lvert NMaß \rvert = 9$	Anzahl der Tiles für das Landschaftsmaß einer Nachbarkachel.
$\lvert Nb \rvert = 6$	Anzahl Nachbarkacheln.
$\lvert Dist \rvert = 10$	Anzahl unterscheidbarer Hex-Distanzen zum Ziel.
$\lvert Rtg \rvert = 6$	Anzahl unterscheidbarer Richtungen zum Ziel.

Es sollen folgende Kodierungen untersucht werden:

1. **Konjunktive Verknüpfung aller Dimensionen**.
   Als Vergleichsreferenz dient die im 1-Tiling Experiment eingeführte Standard Kodierung und Tilings-Organisation, die der **Kodierung 4** in **4.2.3** entspricht.
   Gesamtzahl der Merkmale: $|ZMaß| * |NMaß|^{|Nb|} * |Dist| * |Rtg| \approx 64$ Mio.
2. **Entkopplung von Geographie und Navigation**.
   Entspricht der **Kodierung 4b**, bei der vermutet wird, dass Landschaftsmerkmale und Zielangaben voneinander unabhängig sind.
   Gesamtzahl der Merkmale: $|ZMaß| * |NMaß|^{|Nb|} + |Dist| * |Rtg| \approx 1$ Mio.
3. **Zusätzliche Entkopplung des Zentrumfeldes**.
   Die Beschaffenheit des Zentrumfeldes hat vermutlich wenig Einfluss auf die Aktionswahl, da sie ein unveränderliches Faktum im aktuellen Zustand und deswegen eventuell unabhängig von den Merkmalen der Nachbarschaft ist.
   Gesamtzahl der Merkmale: $|ZMaß| + |NMaß|^{|Nb|} + |Dist| * |Rtg| = 531.503$
4. **Entkopplung der Nachbarkacheln untereinander**.
   Entsprechend der **Kodierung 3a** in **4.2.3** wird hier jede Nachbarkachel für sich betrachtet aber jeweils mit Zentrum, Richtung und Distanz konjunktiv verknüpft.
   Gesamtzahl der Merkmale: $|Nb| * (|ZMaß| * |NMaß| * |Rtg| * |Dist|) = 6.480$
5. **Aussparen der Distanz**.
   In **4.1.3** wurde ausführlich begründet, dass zur Bewältigung der Aufgabe, eine gute nächste Position zu finden, Richtungs- und Zielangaben unerlässlich sind, um ein Lernfortschritt zu erzielen. Die Begründung wurde schließlich bei der Erläuterung des Äquicausa Problems (**5.1.3**) nochmals herangezogen. In welchem Maße die navigatorischen Angaben ins Gewicht fallen, soll durch den Verzicht auf die Distanzangabe bei einer ansonsten konjunktiven Verknüpfung der Dimensionen untersucht werden.
   Gesamtzahl der Merkmale: $|ZMaß| * |NMaß|^{|Nb|} * |Rtg| \approx 6$ Mio.
6. **Verzicht auf Navigationsangaben**.
   Beim vollständigen Verzicht auf Zielrichtung und -distanz muss der Agent allein aufgrund der Landschaftsbeschaffenheit eine Strategie erlernen.
   Gesamtzahl der Merkmale: $|ZMaß| * |NMaß|^{|Nb|} \approx 1$ Mio.

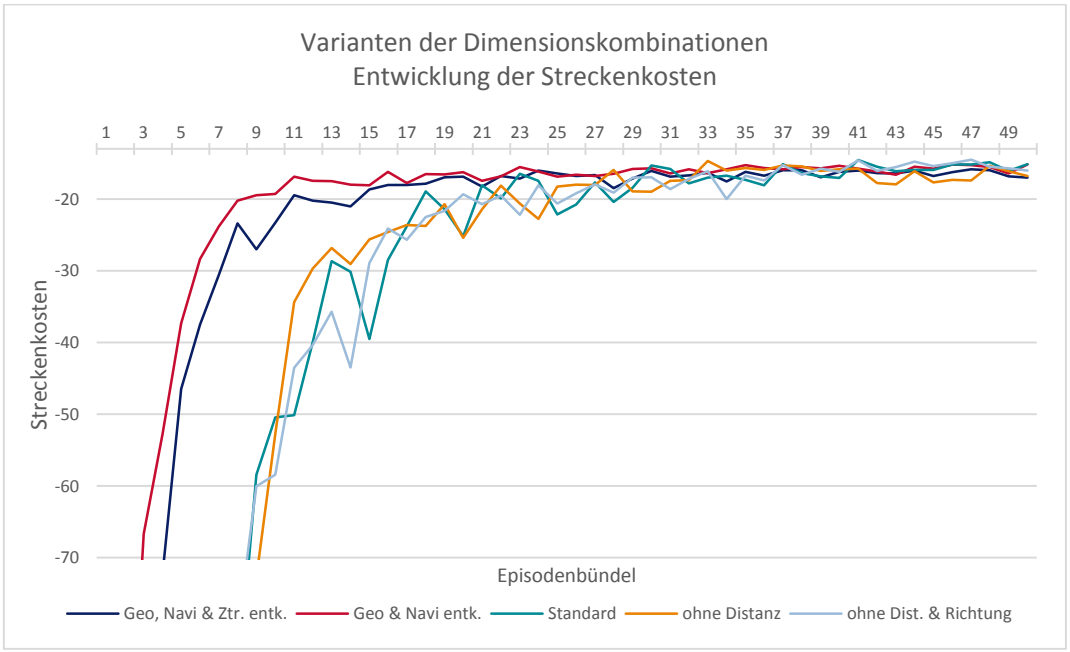


Abb. 6.14: Entwicklung einer Strategie bei unterschiedlichen Dimensionsverknüpfungen. Das Landschaftsmaß ist jeweils durch ein 1-Tiling mit einer Auflösung von 0,1 diskretisiert. Die Standard Lernparameter sind: $\alpha = 0.3, \gamma = 1.0, \lambda = 0.7, \varepsilon = 0.03$

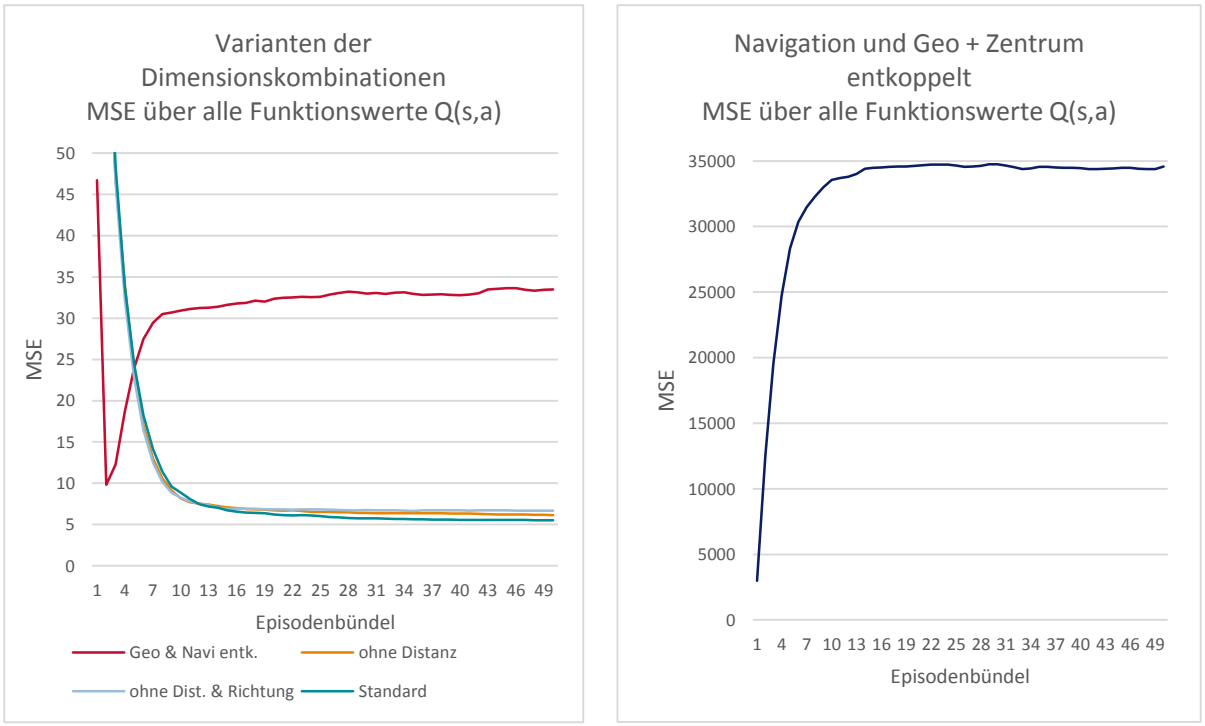


Abb. 6.15: Lernqualität bzw. Mittlere Abweichung der Einzelschätzungen für unterschiedliche Dimensionsverknüpfungen.

Die Experimente wurden auf der HexMap14x16 (**Abb. 6.9**) ausgeführt. Das erste bemerkenswerte Ergebnis ist, dass die Navigationsangaben scheinbar keinen signifikanten Einfluss auf Lernentwicklung und -qualität haben (**Abb. 6.14**, **Abb. 6.15**). Für die Strategiefindung ist hier die Beschaffenheit der Umgebung ungleich gewichtiger, als die Zielentfernung und -richtung. Ob dies allgemein der Fall ist, untersucht das folgende Experiment in einer homogenen Welt, die nur aus Ebenenfeldern besteht.

Die zweite Auffälligkeit betrifft die Entkopplung von Navigation und Geographie. Offensichtlich durch die höhere Generalisierung wird die Lernentwicklung wie vorhergesagt stark beschleunigt und konvergiert gegen nur unwesentlich höhere Streckenkosten, als bei der üblich konjunktiven Verknüpfung, was die **Behauptung 1** soweit bestätigt, wenn man allein die optimale Strategiefindung betrachtet. Allerdings ist für das AMEE-Szenario auch die Lernqualität – die korrekte Bewertung aller einzelnen Zustands-Aktions-Paare – von Belang. Hier zeigt sich an der MSE-Kurve eine Konvergenz gegen einen vergleichsweise hohen mittleren Fehler. Noch sehr viel extremer tritt dieser Effekt bei der zusätzlichen Entkopplung des Zentrumfeldes auf. Interessant hierbei ist es, dass anscheinend trotz einer groben Fehleinschätzung der einzelnen Situationen, in denen sich der Agent gerade befindet, sich eine vergleichsweise gute Strategie zur Zielfindung herausbilden kann. Bemerkenswert ist auch, dass sich die Fehlerkurve bei der entkoppelten Navigation und Geographie in den ersten etwa 20 Episoden (zwei Episodenbündel) noch überaus positiv und schneller als im Standard entwickelt, sich der anfängliche Vorteil der Generalisierung sich dann aber bezüglich der Qualität sehr deutlich ins Gegenteil kehrt. Ein Effekt, der dem MultiTiling zugesagt wird, hier aber sehr viel stärker ausgeprägt ist. Anstatt sich Gedanken über ein adaptives Tiling zu machen, scheint es nach diesen Ergebnissen vielversprechender zu sein, sich einer adaptiven Kodierung der Dimensionen des Zustandsraums zu widmen.

Die vierte zu untersuchende Kodierung, die Entkopplung der Nachbarkacheln untereinander, ist in den Abbildungen 6.14 und 6.15 nicht aufgeführt, da sich diese Kodierung als ungeeignet für eine Strategiefindung herausgestellt hat. Der Agent hat keinerlei verwertbare Erfahrungen gesammelt und fand das Ziel allenfalls zufällig innerhalb der vorgegebenen maximalen Schrittzahl (doppelte Anzahl von Land-Kacheln der Karte). Eine weitergehende Minimierung des Merkmalsraums, wie sie beispielsweise mit **Kodierung 3b** in **4.2.3** vorgeschlagen wird, braucht deswegen nicht untersucht werden.

6.4.2 Homogene Welt

Nach dem vorangegangen Experiment scheinen die navigatorischen Merkmale keinen Einfluss auf die Lernentwicklung und -qualität zu haben. Die Zielrichtung und -distanz wurde in **4.1.3** eingeführt, da neben den unmittelbaren Kosten auch gelernt werden soll, dass eine gute nächste Position den Agenten näher ans Ziel bringt, vor Allem aber, zur besseren Unterscheidung von Zuständen in gleichförmigen Welten. Die verwendete Welt ist allerding sehr heterogen: Die 158 Landkachel werden auf 157 Merkmale abgebildet (ein Zustand besitzt hier genau ein Merkmal) und sind somit gut unterscheidbar. Bevor also voreilig auf die Navigationsangaben verzichtet wird, soll ein Experiment in der homogenen Welt hexMap14x16-02, in der alle Landkacheln eben und ohne Flussübergänge sind, die

Notwendigkeit der Angaben evaluieren. Die Welt wird auch hier wieder durch Wasserfelder am Kartenrand begrenzt. In der Standard-Kodierung mit einer konjunktiven Verknüpfung aller Zustandsdimensionen werden die 167 Landfelder auf 62 Merkmale abgebildet. Der Agent erkennt mit seinem beschränkten Sichtfeld folglich nur 62 unterscheidbare Zustände. Verzichtet man auf die Distanz, reduziert sich die Zahl auf 25 und ohne jegliche Navigationsangaben auf magere elf Zustände, anhand derer die Lernaufgabe bewältigt werden soll.

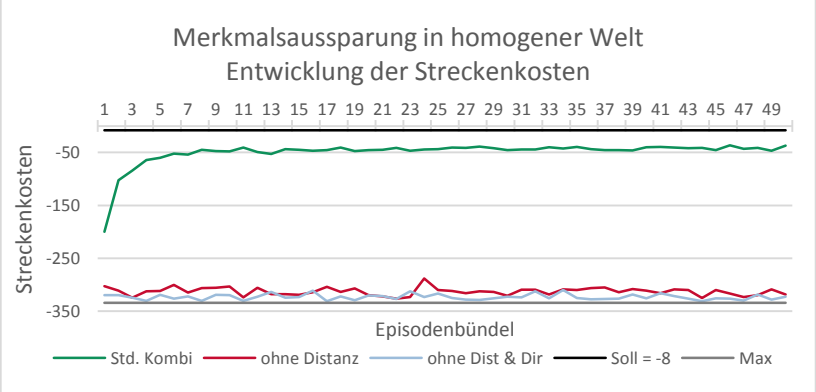
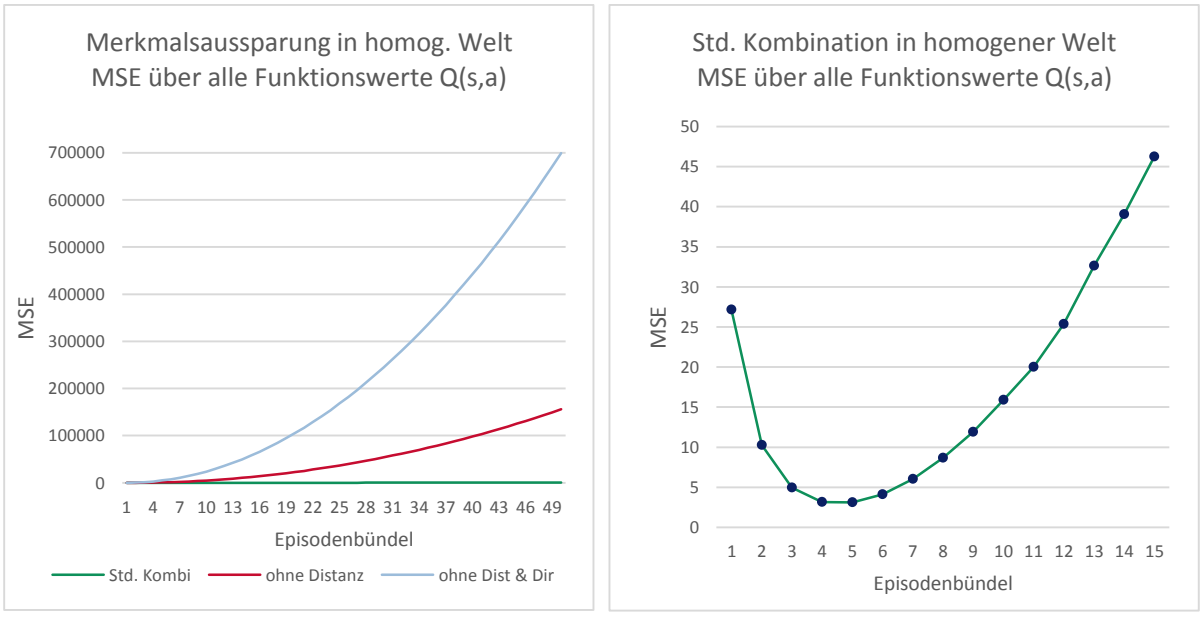


Abb. 6.16: Lernentwicklung in einer homogenen Welt mit und ohne navigatorische Zustandsangaben – verwendet ein 1-Tiling mit 9 Tiles für Landsch.maß.



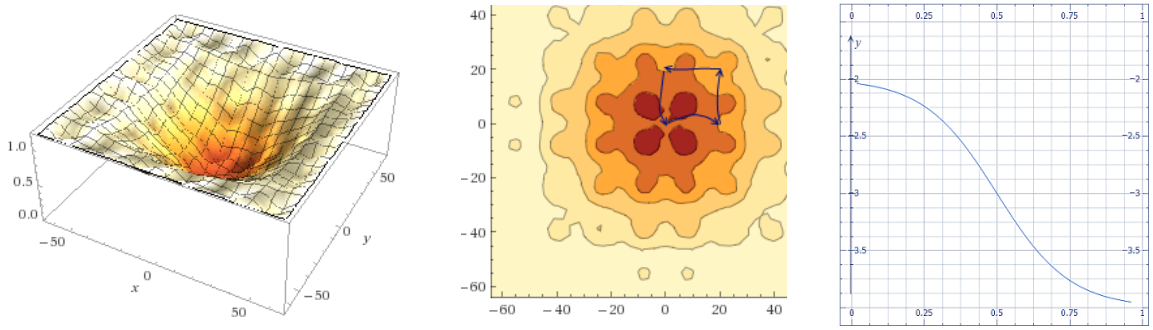
(a) (b)

Abb. 6.17: Fehlerentwicklung / Lernqualität in einer homogenen Welt. (a) Exponentiell umso stärker steigender Fehler, je weniger Dimensionen berücksichtigt werden. (b) Vergrößerung der Entwicklung für die Standard Kodierung.

Es zeigt sich, dass in der homogenen Welt nur unter Berücksichtigung aller navigatorischen Zustandsdimensionen überhaupt eine Strategie gefunden werden kann (**Abb. 6.16**), wenn auch mit

durchschnittlichen Kosten von etwa 45 statt acht in der Konvergenz eine recht schlechte. Die Fehlerentwicklung bezüglich der Lernqualität (**Abb. 6.17**) zeigt ebenfalls, dass sich eine eintönige Umwelt äußerst negativ auf das Lernverhalten auswirkt. Für die Standard-Kodierung kommt es erneut zu dem Effekt, dass sich eine anfänglich positive Entwicklung der Lernqualität dramatisch umkehrt, so wie es in der heterogenen Welt auf die unvollständigen Kodierungen zutraf. Es ist umso frappierender, dass sich hier sogar trotz steigender Fehleinschätzung im Einzelnen eine Strategie im Gesamten ergibt.

6.5 Modell AMEE



(a) (b) (c)

Abb. 6.18: 3D und 2D Plot der Welt als Unebenheitskarte. Dunkelrot = sehr eben, Ocker = sehr rau. (b) In blau gekennzeichnet die intuitiv geschätzten Optimalpfade im Experiment. Start und Ziel sind immer 20 Meter entfernt. (c) Simulation der Schrittanalyse durch Vergeltungsfunktion in Abhängigkeit von der Unebenheit auf der Trittposition: Schrittkosten (y-Achse) von -2 bis -4.

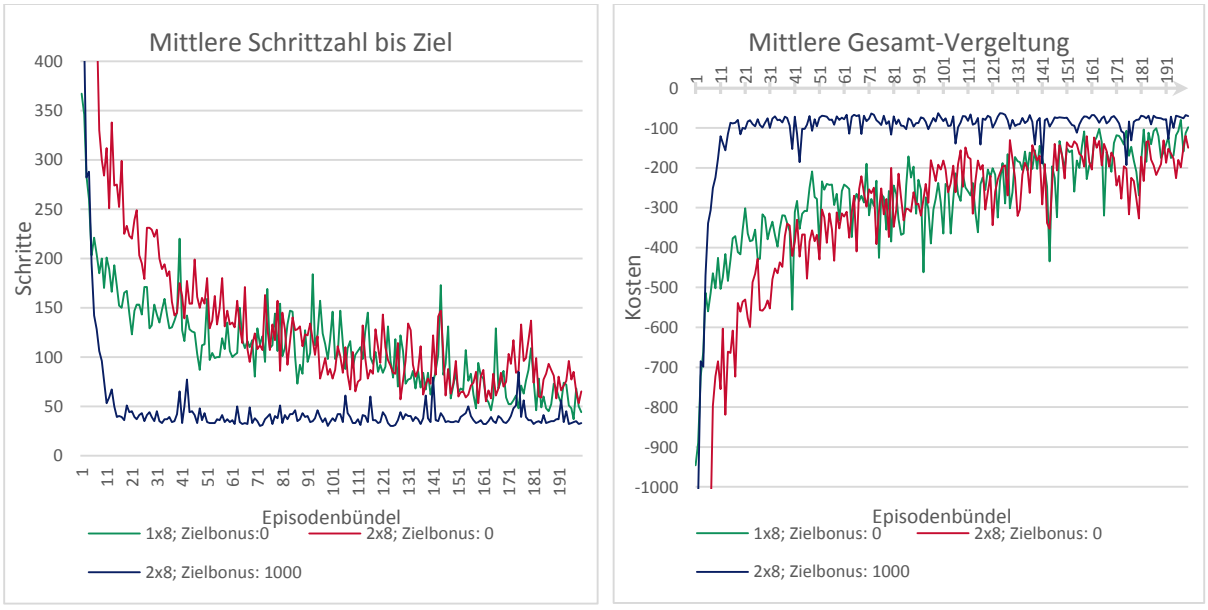
Mit den gewonnenen Erkenntnissen aus den vorigen Experimenten wird eine Versuchsreihe für das eigentliche AMEE-Szenario (siehe **4.1**) und dessen radialer Unebenheitskarte durchgeführt. Für das Unebenheitsmaß wird ein 1-Tiling verwendet, das die Maße von 0,0 bis 1,0 in acht Segmente unterteilt, zehn Entfernungsintervalle beschreiben die Distanz zum Ziel. Die Karte besteht aus acht Sektoren und einer, beziehungsweise zwei Spuren, sodass acht und 16 Abschnitte entstehen. Eine „Welt“-Funktion über (x,y)-Koordinaten simuliert die Messergebnisse entsprechend **Abb. 6.18 (a)/(b)** (vgl. **4.1.3** und **A.3 Weltfunktionen**). Eine RewardFunction simuliert die Schrittkosten in Abhängigkeit von der Unebenheit auf der Schrittposition. Ein Schritt kosten mindestens zwei Einheiten auf ebenem Untergrund und bis zu vier Einheiten auf zunehmend raueren Gelände. Für die Lernparameter werden die Ergebnisse aus **6.1.5** verwendet: $\alpha=0.3$, $\gamma=1$, $\lambda=0.7$, $\varepsilon=0.03$. Im Experiment werden vier Koordinaten als Start und Ziel, die immer 20 Einheiten (z.B. Meter) auseinanderliegen, permutiert.

Da das Weltmodell, die möglichen Aktionen, Pfade und Kosten hier sehr viel komplexer sind als im Dampfross®-Szenario, wird auf die Ermittlung von optimalen Sollkosten und einer Untersuchung der Lernqualität in Form des mittleren Fehlers wie in den vorangegangenen Experimenten verzichtet. Die wichtigsten Einflüsse bezüglich Modellierung und Parametrisierung sind durch die Dampfross®

Versuche bereits ausführlich analysiert. Daher wird hier vordergründig die Eignung des AMEE-Lernmodells demonstriert und darüber hinaus noch der Einfluss von komplexeren Sichtfeld-organisationen und die Verwendung einer Zielbelohnung untersucht.

Im ersten Experiment ist die Umgebungskarte in acht Sektoren und einer zwei Meter breiten Spur organisiert, woraus sich acht Abschnitte beziehungsweise Aktionsalternativen mit einer Schrittweite von einem Meter ergeben. Es ist zwar deutlich ein Lernfortschritt zu erkennen (**Abb. 6.19**), es deutet sich eine Konvergenz der minimalen Schrittzahl und einhergehend der mittleren Kosten je Episode an, doch entwickeln sich die Kurven verhältnismäßig langsam und mit einer starken Streuung. Dies könnte an der im Vergleich zu den Dampfross-Karten größeren Testwelt liegen. Eventuell ist die Welt auch zumindest im Radius des Sichtfeldes noch zu homogen für eine optimale Lernentwicklung. Erhöht man die Komplexität der Umgebungskarte von einer Spur auf zwei je ein Meter breite Spuren, mit Schrittweiten von 0,5 und 1,5 Metern, so verlangsamt sich die Lernentwicklung noch ein wenig aufgrund der insgesamt höheren Modell-Komplexität.

Es ist bekannt, dass bei episodischen Aufgaben eine Zielbelohnung, die deutlich über den durch-schnittlichen Wegkosten liegt, die Strategiefindung beschleunigt. Auch bei der 2x8 Umgebungskarte ist dies sehr deutlich zu erkennen, wenn der Agent eine Belohnung von 1.000 Einheiten bei Erreichen des Etappenziels erhält. Wie sich das auf die Entwicklung der Lernqualität auswirkt, könnte eventuell noch näher Untersucht werden. Auf jeden Fall würden die situationsindividuellen Einzelabschätzungen nicht mehr die tatsächlichen Kosten wiederspiegeln, sondern „nur“ ein noch weiter abstrahiertes Qualitätsmaß.



(a) (b)

Abb. 6.19: Entwicklung der mittleren Anzahl von Schritten bis zu Zielkoordinate (a) und der erzielten Vergeltung (b). Die blaue 2x8; Zielbonus: 1000 Kurve ist durch Abzug der 1.000 Belohnungspunkte von der Gesamtvergeltung normiert.

6.6 Zusammenfassung

Mittels der Experimente wurde sukzessive die Parametrisierung, Tilings-Organisation und Merkmalskombination für die bestehende Lernaufgabe optimiert. Dabei galt es zwei teilweise widersprüchliche Anforderungen zu berücksichtigen. Einerseits die schnelle Entwicklung einer guten Strategie und andererseits eine hohe Genauigkeit auch für die Bewertung einzelner Zustands-Aktionspaare. Es zeigt sich deutlich, dass der Fokus von Reinforcement Learning in der Strategiefindung liegt: Es soll möglichst effizient das Ziel erreicht oder das Spiel gewonnen werden. Dabei kann die Einschätzung der jeweils aktuellen Optionen innerhalb einer Episode durchaus grob von der Realität abweichen, solange sich daraus eine übergeordnete Strategie herleiten lässt. Die vorhandene Modellierung der Lernaufgabe als RL-Problem bevorzugt offensichtlich ebenfalls das Erreichen einer Etappenkoordinate, da sowohl Geographie als auch Navigation eine Rolle spielen und letztlich eine Koordinate oder Zielkachel eben das Episodenende bezeichnet. Der Anspruch, auch genaue Einzelabschätzungen zu erlernen wird bei schlechter Parametrisierung schnell verfehlt. Anhand des diskreten Dampfross®-Modells wurden die Lernparameter $\alpha = 0.3$, $\gamma = 1.0$, $\lambda = 0.7$ und $\varepsilon = 0{,}03$ für die geeignetsten Kompromisswerte befunden.

Für das Äquicausa Problem wurde die **Behauptung 2** weitgehend bestätigt: Sofern die Lernumgebung nicht zu homogen ist, stellen sich globalgültige Mittelwerte für gleiche Zustände mit unterschiedlichen Realkosten ein. Hier zeigt sich aber auch eine weitere Schwäche des Lernverfahrens, beziehungsweise seiner Modellierung. In gleichförmigen Landschaften entwickelt sich zwar auch eine minder gute Strategie, aber die Fehler der Einzelbewertungen steigen mit jeder Lernepisode exponentiell an.

Auch die **Behauptung 1**, dass unter Umständen disjunktive Merkmalskombinationen ebenfalls zu adäquaten Lernerfolgen führen, konnte ansatzweise bestätigt werden. Die Trennung von geographischen und navigatorischen Merkmalen beschleunigte die Lernentwicklung erheblich, allerdings auf Kosten des Mittleren Fehlers der Einzelbewertungen, die auf einen etwa sechsfach höheren Wert konvergierten als bei der rein konjunktiven Verknüpfung. Ein Effekt, der für das MultiTiling vorhergesagt wurde, aber nur sehr viel schwächer ausgeprägt nachgewiesen werden konnte. Daher ist von einem MultiTiling an sich und von dem Mehraufwand für ein adaptives Tiling hier eher abzuraten. Auch die disjunktive Merkmalskodierung sollte für unsere Lernaufgabe nur angewandt werden, wenn sich die Ersparnis von Speicherressourcen lohnt. Bezüglich der Auflösung einer Tilings-Organisation haben sich keine klaren Tendenzen herausgebildet, außer dass es anscheinend einigen Spielraum in Richtung grober Auflösung gibt.

Schließlich wurde das eigentliche AMEE-Szenario getestet und es zeigte sich neben dem generellen Erfolg des Modells, dass gegebenenfalls eine Zielbelohnung entscheidend zur positiven Lernentwicklung beitrüge.

7 Weiterführende Konzepte

Im Laufe der Modellierung und Untersuchung zum Lernverfahren für die Wahl sicherer Schrittpositionen sind einige Herausforderungen hervorgetreten, zu denen es auch Ideen gibt, die in dieser Arbeit aber nicht untersucht werden konnten. Die wichtigsten beiden betreffen die Optimierung des Lernverfahrens hinsichtlich Geschwindigkeit (**7.1**) und Ressourcenverbrauch (**7.2**). Wie die bisher theoretische Betrachtung des Lernverfahrens in den von Ruhnke [Ruh14] ausgearbeiteten Software Kontroller eingebettet werden kann, wird in **7.3** beleuchtet. Unmittelbar ausstehende Arbeitspakete des Projekts AMEE listet Abschnitt **7.4** auf.

7.1 Adaptive Merkmalskombination

Im Experiment **6.4.1** wurde festgestellt, dass die semantische Trennung von Navigations- und Geographiemerkmalen in Form einer disjunktiven Merkmalskombination zu einer erheblichen Beschleunigung der Lernentwicklung führt, sich dieser Vorteil bei fortschreitendem Lernen aber ins Gegenteil kehrt, insbesondere bezüglich des mittleren Fehlers der Einzelbewertungen. Inspiriert durch Ansätze zum adaptiven Tiling soll hier eine Methode zur adaptiven Merkmalskodierung skizziert werden.

Die Herausforderung bei der Umkodierung des Merkmalraums von einer disjunktiven Verknüpfung hin zu einer konjunktiven liegt darin, Merkmalsgewichte, die das erlernte Wissen repräsentieren, an die neue Kodierung anzupassen, ohne die erzielten Erfahrungen zu verfälschen. Dabei ist es sicherlich ungefährlicher, die Anzahl der Merkmale zu erhöhen als zu verkleinern. Es soll hier aber gerade der anfängliche Vorteil der disjunktiven Kombination, mit weniger Merkmalen, in den späteren Vorteil einer konjunktiven Kombination, mit mehr Merkmalen, überführt werden. Angenommen, Distanz und Zielrichtung sind konjunktiv verknüpft und die Unebenheit als unabhängiges zusätzliches Merkmal modelliert, also eine unabhängige Betrachtung von Navigation und Geographie vorliegt. Hat der aktuelle Zustand beispielsweise das Zielmerkmal f_2 und das Unebenheitsmerkmal f_0', so errechnet sich die Bewertung aus der Summe der zugehörigen Gewichte: $V(s) = w_2 + w_0'$. Genau dies müsste dann die Abbildungsregel sein, um von der disjunktiven zur vollständig konjunktiven Verknüpfung zu wechseln. Denn der Zustand, der zuvor die beiden Merkmale (f_2, f_0') besitzt, hat in der neuen Kodierung die Kombination $f_{20} = f_2 \times f_0'$ mit dem Gewicht $w_{20} = w_2 + w_0'$ (vgl. **Abb. 7.1**).

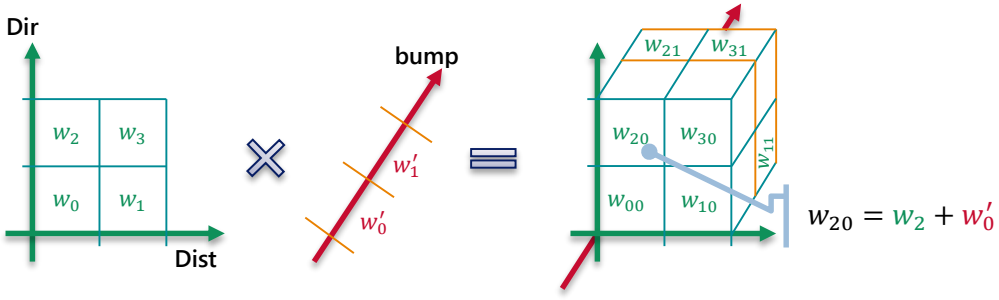


Abb. 7.1: Unabhängige Betrachtung der Kombination Distance X Direction und des Merkmals bumpiness (Unebenheit), sowie die Abbildung auf die Konjunktion aller Merkmale. *w* bezeichnet das Merkmalsgewicht.

Es stellt sich die Frage nach dem Zeitpunkt des Kodierungswechsels. Ein Maß könnte der in Sarsa(λ) errechnete Schätzfehler $\delta = r + Q(s^+, a^+) - Q(s, a)$ sein, der bei der Unterschreitung eines Minimalwerts zur Umschaltung führt. Doch kann dieser Fehler durchaus vorerst auf ein lokales Minimum konvergieren und dann wieder steigen, wie es bei der Entkopplung von Geographie und Navigation im Experiment **6.4.1** beobachtet wurde (**Abb. 6.15**). Wenn der minimale Fehler zu niedrig gewählt ist, verpasst man den Zeitpunkt für den Wechsel und verbleibt bei der dann nachteiligen ersten Kodierung. Ein anderer Ansatz wäre es, auf die anfänglich gute Lernentwicklung zu vertrauen und den Wechselzeitpunkt an der Häufigkeit festzumachen, wie oft ein Zustand besucht wurde. Hierzu könnte ein Zustand oder Merkmal mit einem zusätzlichen Vertraulichkeitsmaß versehen werden, dass mit jedem Vorkommen erhöht wird und gegen Eins konvergiert, beispielsweise mit der Gleichung $c_n = 1 - b^n \mid n \in \mathbb{N}, b\ in\ [0,1]_{\mathbb{R}}$. Ist das Vertrauen in die Korrektheit des Gewichts über einem gewissen Punkt, wird die Kodierung gewechselt.

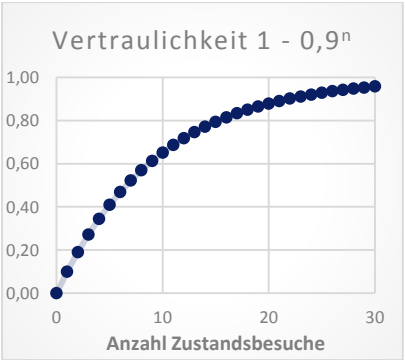


Abb. 7.2: Entwicklung eines Vertraulichkeitsmaßes, das angibt, wie weit sich ein Merkmal wahrscheinlich zum Optimum entwickelt hat und zur Wahl des Zeitpunkts herangezogen werden kann, wann von einer disjunktiven zur einer konjunktiven Merkmalskombination gewechselt werden kann.

7.2 Fluch der Dimensionalität

Um dem Fluch der Dimensionalität beim Zustandsraum zu begegnen, wurde in Abschnitt **4.2** eine unabhängige Betrachtung der Dimensionen vorgeschlagen, die sich in einer disjunktiven Verknüpfung der Merkmale äußert. Aus $dim = |Unebenh.|^{|Abschn.|} \cdot |Richtg.| \cdot |Dist.|$ wurde beispielsweise bei der Entkopplung von Geographie und Navigation $dim = |Unebenh.|^{|Abschn.|} + |Richtg.| \cdot |Dist.|$. Zum einen haben allerdings die Untersuchungen in **6.4.1** unbestreitbar gezeigt, dass die übliche rein konjunktive Verknüpfung überlegen ist und zum anderen bleibt der Knackpunkt die exponentielle Komplexität durch $|Unebenh.|^{|Abschn.|}$.

Ein alternativer oder zusätzlicher Ansatz wäre eine gestaffelte Betrachtung des RL-Zustands in Form der Unebenheitskarte (vgl. **4.1**). Im ersten Schritt wird anhand einer grob untergliederten Karte gemäß der Erfahrung und einer Greedy-Strategie der beste Abschnittskanditat ausgewählt. Doch anstatt diesen als empfohlene Aktion weiterzugeben, wird der Abschnitt nochmals feiner in ebenso viele Unterabschnitte untergliedert, wie in der Gesamtkarte vorhanden sind. Die feinere Teilansicht dient dann als eigentlicher Zustand für die Aktionswahl und RL-Aktualisierungen der Wissensbasis. Dadurch kann im Endeffekt die Anzahl der Abschnitte vervielfacht werden, ohne die Gesamtzahl der Merkmale des RL-Algorithmus zu erhöhen. Es stellt sich jedoch die Frage, inwiefern das erlernte Wissen für die Subkacheln auf die gröbere Gesamtansicht anwendbar ist.

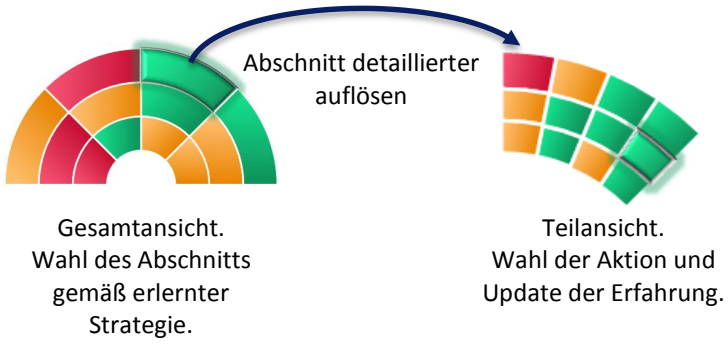


Abb. 7.3: Gestaffelte Betrachtung der Unebenheitskarte. Die Farben Grün, Orange, Rot symbolisieren hier Unebenheiten von glatt bis rau.

Ein ebenfalls näher zu beleuchtender Ansatz, wäre der von Firouzi et. al. [Fir12]. Auch hier ist man auf den Gedanken gekommen, die Zustandsdimensionen getrennt zu beobachten. Allerdings nicht in Form einer disjunktiven Merkmalskombination, sondern indem für jede Dimension ein eigener Lernagent (tiny agent) angesetzt wird. Jeder Agent wählt die Aktion gemäß seiner Erfahrung, die speziell auf eine Eingangsdimension beruht. Ein „Aktionsverschmelzer“ (Action Fuser) wählt dann aus allen Vorschlägen

die letztlich auszuführende aus. Die Vergeltung von der RL-Umweltkomponente geht an alle Agenten, die daraufhin ihre spezielle Erfahrung aktualisieren. Die Wahl der auszuführenden Aktion beruht auf einer Expertenregel, die diejenige Aktion des im aktuellen Kontext erfahrensten Agenten nimmt. Obwohl die publizierten Untersuchungen zu diesem Ansatz vielversprechend sind, bestehen auch hier noch Unsicherheiten, ob sich auch bei stark voneinander abhängigen Eingangsmerkmalen ein guter Lernerfolg einstellt, insbesondere da sich der Action Fuser immer für genau einen Experten entscheidet, anstatt die Vorschläge vielleicht zu verrechnen.

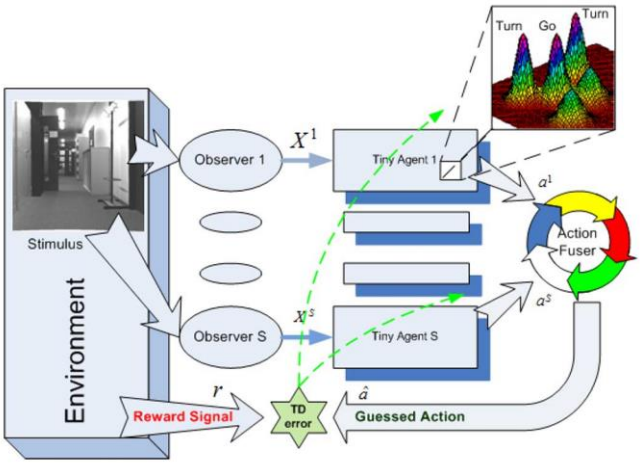


Abb. 7.4: Framework-Schema von Firouzi et.al. [Fir12] zur Partitionierung des Zustandsraums nach Dimensionen. X^i bezeichnet den Eingangsvektor des i'ten Sensors

7.3 Schnittstelle zum Lauf-Kontroller

In der Arbeit von Ruhnke [Ruh14] wird beschrieben, wie innerhalb des Deliberativ Layers eine dynamische, iterative Laufrichtungs- und Trittpositionsplanung erfolgt. Die elegante, regelbasierte Lösung berücksichtigt auf Basis eines Umgebungsscans Höhenunterschiede, gewünschte Laufrichtung, Trittreichweite und weitere Faktoren auf der Suche nach möglichst ebenen Trittpositionen. Die Laufrichtungsplanung erzeugt dabei einen auf Sichtweite ausgelegten groben Pfad für die Maschine als Ganzes, während die Trittpositionsplanung dies zu Positionierungskandidaten einzelner Füße verfeinert.

Das Lernverfahren dieser Thesis, zur Auswahl sicherer Trittpositionen, soll durch die propriozeptäre Rückmeldung zusätzliche, indirekte Faktoren zur Laufeigenschaft des Bodens berücksichtigen. Bestandteil des Algorithmus ist die Bewertung von Zustands-Aktions-Paaren, wobei der Zustand durch eine in Abschnitte unterteilte Umgebungskarte ist und eine Aktion der Fuß- oder Roboterpositionierung auf eines dieser Abschnitte entspricht. Insofern gibt es eine offensichtliche Gemeinsamkeit beider Ansätze: Die Unterteilung einer Umgebungskarte in Trittpositionskandidaten.

Kostenkarte zu Qualitätsmatrix

Genau diese Gemeinsamkeit kann ausgenutzt werden, um beide Ansätze und vor allem ihre jeweiligen Vorteile miteinander zu kombinieren. Dazu muss die Segmentierung der Umgebungskarte aufeinander abgestimmt werden und die jeweilige Kostenschätzungen der Lernkomponente für die aktuelle Umgebung und alle Aktionen in eine Qualitätsmatrix überführt werden. Diese Matrix kann entweder in die Laufrichtungs- oder Trittpositionsplanung des Deliberative Layers integriert werden. Beispielhaft soll das an der Laufrichtungsplanung verdeutlicht werden (siehe **Abb. 7.5**). Hier wird der Umgebungsscan zu einer Höhenkarte, beziehungsweise -matrix, verrechnet und mit einer Entfernungs- und Richtungsmatrix aufsummiert. Jede Matrix ist mit einem Faktor gewichtet, um Prioritäten zu steuern. Die einzelnen Matrizenelemente sind wiederum von Null bis Hundert aufsteigend priorisiert. Die Qualitätsmatrix des Lern-Moduls wird nun als zusätzlich zu summierende und eigens gewichtete Matrix in diese Berechnung aufgenommen. Das Lernmodul erstellt dazu die Qualitätsmatrix aus den Kostenschätzungen zum aktuellen Umgebungsscan (siehe **Abb. 7.6**), beginnend mit der Unebenheitskarte. Diese im Lern-Kontext der aktuelle Zustand. Jeder Abschnitt (jede Kachel) ist eine mögliche Aktion und zu jedem Zustands-Aktions-Paar wurde eine Kostenschätzung Q(s,a) erlernt. Alle zusammen ergeben eine Kostenkarte, welche für die Laufrichtungsplanung nur noch in dessen Wertebereich von 0 bis 100 normiert werden muss und dann als Qualitätsmatrix übergeben werden kann.

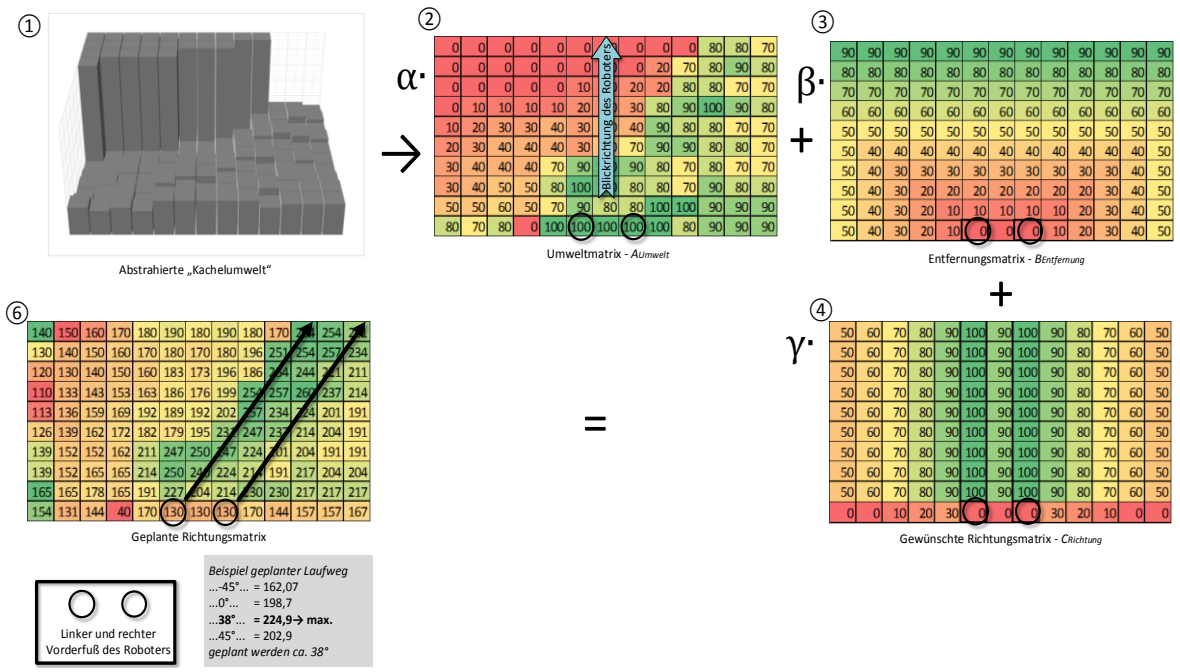


Abb. 7.5: Kalkulation der Laufrichtungsplanung im Deliberative Layer der Robotersteuerung (Quelle: [Ruh14]). Je höher der Wert, desto besser die Trittposition. Bei der Höhenkarte A wird die Höhe der aktuellen Auflage mit 100 als optimal bewertet, weite Schritte werden in der Entfernungsmatrix B bevorzugt, die Richtungsmatrix C priorisiert die gewünschte Richtung. Aus der gewichteten Summe wird die geplante Laufrichtung extrapoliert.

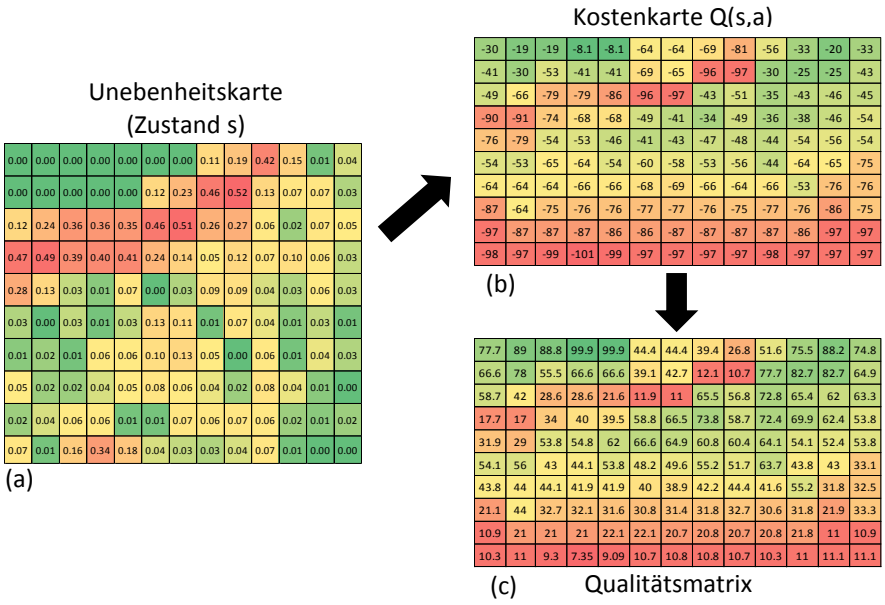


Abb. 7.6: (a) Unebenheitskarte zu Abb. 7.3 ① als Zustand des RL-Algorithmus. (b) Fiktive Kostenkarte, bestehend aus den RL-Schätzwerten zum Zustands s – Aktion = Tritt auf Kachel. (c) Normierung der Kostenkarte entsprechend des Zahlensystems der Laufrichtungsplanung zu einer Qualitätsmatrix.

Erkundungsstrategie

Eine weitere Schnittstelle, die der Lernalgorithmus benötigt, ist die Strategie zur Aktionswahl. In den simulierten Testreihen des **Abschnitts 6** wurde eine ε-Greedy Strategie genutzt, die im realen System im Zusammenspiel mit dem Deliberative Layer entfällt. Denn letztlich entscheidet die Laufkontrolle über die tatsächliche Trittposition. Während in der ε-Greedy Strategie der Wert für Epsilon die Wahrscheinlichkeit einer Abweichung von der vermeintlich besten Aktion bestimmt, um so „Neues" auszuprobieren, wird dies im Zusammenspiel indirekt durch die Gewichtung der Qualitätsmatrix im Deliberative Layer übernommen. Je höher die Qualitätsgewichtung im Verhältnis zu den anderen Matrixgewichten ist, desto eher wird der erfahrungsbasierten Empfehlung des Lernmoduls gefolgt. Die tastsächliche Trittpositionswahl muss dann dem Lernmodul mitgeteilt werden, wodurch die Laufkontrolle dem Lernmodul gegenüber quasi als Erkundungsstrategie auftritt, anstelle von ε-Greedy.

Vergeltung (Reward)

Die bisher simulierte Vergeltung muss ebenfalls von externen Komponenten, vor allem dem Lauf-Kontroller bereitgestellt werden. Fürs Erste würde ein Schritt pauschal mit einer Kosteneinheit versehen werden. Hinzu sollten Fehler in der Schrittausführung, die durch die verteilten Beinsteuereinheiten mitgeteilt werden (siehe [Ruh14]) zusätzlich deutlich bestraft werden, beispielsweise mit zehn Kosteneinheiten. Darüber hinaus können Analysen zur Stromaufnahme oder Entwicklung der Drehmomente während der Schrittausführung herangezogen werden (vgl. [Kim10]).

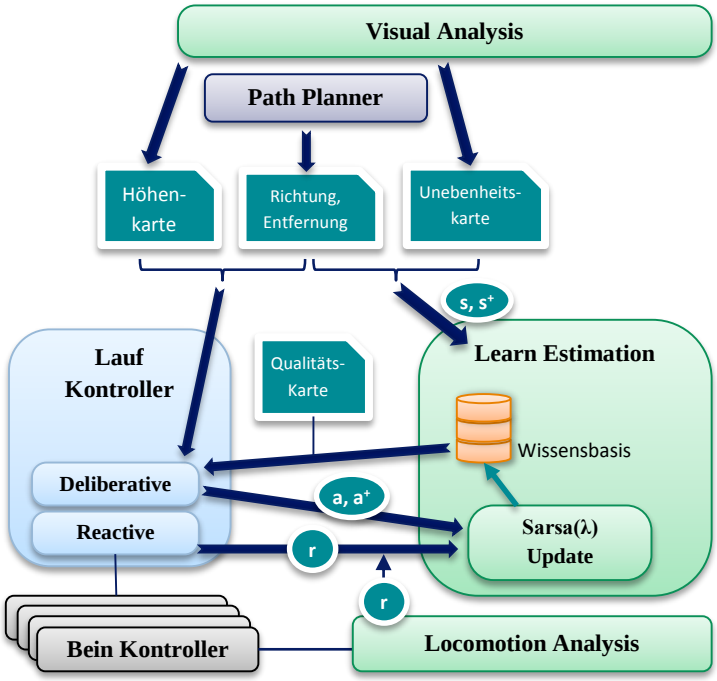


Abb. 7.7: Schematisches Zusammenspiel der erfahrungsbasierten Trittwahl und der Lauf-Steuerung.

7.4 Weitere Herausforderungen

Robot Vision

Ansätze zur Umgebungserfassung wurden in 3.2 demonstriert. Diese müssen noch ausgearbeitet werden, damit sowohl der Laufkontroller als auch das Lernverfahren mit Daten versorgt werden.

Propriozeption

Im Zuge der Zusammenführung von Lernmodul und Lauf-Kontoller muss festgelegt werden, welche Rückmeldungen der Lernalgorithmus erhält und wie sie zu einem Vergeltungswert verrechnet werden.

Simulationsmodell von AMEE

Abgesehen von einigen isolierten Simulation zu speziellen Fragestellungen in der AMEE-Softwareentwicklung gibt es keine physische Simulationsumgebung des Roboters. Diese ist dringen notwendig, um die Weiterentwicklung zu beschleunigen und ohne Gefährdung der Maschine Tests ausführen zu können.

Modulare Software Architektur / Alternative zu DSS

Neben der grundsätzlichen Laufkontrolle soll das hier vorgestellte Lernmodul der Roboteranwendung hinzugefügt werden. Mittelfristig werden zahlreiche weitere Module hinzukommen, die mehr oder weniger eigenständige Aufgaben bearbeiten. Die Wahl einer geeigneten Softwarearchitektur wird dadurch immer wichtiger. Die Vorteile einer verteilten Roboteranwendung mit dem DSS[34]-Framework wurde in [Bet10] und **3.1** gezeigt. Angesichts dessen, dass die Entwicklung an DSS eingestellt wurde, müssen gegebenenfalls Alternativen gefunden werden.

Feldversuche

Der Roboter ist zwar Einsatzbereit, wurde aus Sicherheitsgründen bisher aber nur in einer Traverse hängend gestartet. Ein Feldversuch steht noch aus, anhand der auch noch einige Kontroversen zur Konstruktion und zu Gangarten diskutiert werden müssen.

Elektromechanische Konstruktion

Es sind noch konstruktive Verbesserungen bezüglich Energiemanagement und Busleitungen empfehlenswert.

[34] Decentralized Software Services des Microsoft® Robotics Studio

Von Ruhnke [Ruh14] bereits identifizierte Aufgaben:

- Odometrie
  Behandlung von Schlupf, etc.
- Pfadplanung
  Auf mittlere Sicht und unter Aspekten von Such- und Erkundungsaufgaben.
- Drehen auf der Stelle
  Noch nicht unterstützt.
- Adaptive Gewichtung
  Lern- und Regelungsverfahren zum Parameter Tuning für den Deliberative Layer.
- Personenerkennung
  Der Roboter soll sich gegenüber Gestrüpp durchsetzen, aber nicht gegenüber Menschen.
- Userinterface und Visualisierung
  Roboter soll einem Operator berichten und Aufträge entgegennehmen.

8 Resümee

Das Laufen in schwierigem, unstrukturiertem Gelände erfordert eine sorgfältige Abwägung der Trittpositionen. Mittels der Methode des Reinforcement Learnings wurde ein Lernverfahren entworfen, das die wahrgenommene unmittelbare Umgebung bezüglich der Laufqualität bewertet und durch die Erfahrung der Schrittausführung die Bewertung verbessert. Dies ermöglicht dem System, ohne detailliertes Vorwissen auch in unbekanntem Terrain sicher voranzuschreiten. Kern des vorgestellten Verfahrens war dabei die Modellierung der RL-Aufgabe basierend auf einer Umgebungskarte mit Landschaftsmerkmalen und dem eigens entwickelten Programmiermodell für RL. Die von der RL-Umwelt-Komponente gekapselte Außen- und Eigenwahrnehmung wurde konzeptionell untersucht und für das Lernverfahren simuliert. In einem weiter abstrahierten, methodisch äquivalentem Simulationsmodell, basierend auf dem Brettspiel Dampfross®, wurden die Parameter und Modellierungsvarianten überprüft und verbessert, wobei stets ein Kompromiss zwischen der übergeordneten Strategie zum kosteneffizienten Erreichen des Etappenziels und dem Anspruch einer möglichsten exakten Bewertung der jeweils aktuellen Trittpositionskandidaten für eine Kostenkarte der momentanen Begebenheiten gesucht wurde. Ein finales Experiment mit dem AMEE-Lernmodell zeigte die generelle Eignung des Verfahrens. Abschließend wurden Konzepte zu Integration in das Zielsystem und Weiterentwicklung des Verfahrens vorgestellt.

Die Schrittpositionswahl mittels bestärkenden Lernens ist durchaus in der Lage, ihre Aufgabe zu erfüllen und den laufenden Roboter auch über schwieriges Gelände zu führen. Es ist aber nicht zu empfehlen, das Lernmodul als alleinentscheidende Komponente für sich stehen zu lassen. Vielmehr sollte die Schrittwahl als Empfehlung in eine Kombination von Auswahlmechanismen einfließen. Es gibt simplere Regeln anhand derer man Trittkandidaten festmachen kann, wie Gleichgewicht der Maschine, Kollisionsfreiheit der Beine untereinander, ebene Teilflächen, die errechnet oder geregelt werden können und nicht erst gelernt werden müssen. Einfache Regeln und die abstrakteren Erfahrungen können dabei durchaus auseinanderlaufen. Kombiniert man die Empfehlungen unterschiedlicher Module, nicht nur priorisiert sondern eventuell mit einem probabilistischem Sicherheitsmaß („Lernmodul, wie sicher bist Du Dir?“), erzielt man gewiss die besseren Resultate.

Der Rough Terrain quadruped Robot kann – sofern er sicher läuft – vielfältig eingesetzt werden und zwar dort, wo ein erhöhtes Maß an Mobilität gefordert ist. Sei es im Wald, an Hängen, auf Baustellen, in Gebäuden oder selbst im urbanen Umfeld zwischen parkenden Autos durch und über Treppenstufen hinweg. Die Hauptaufgabe bleibt dabei in aller Regel, den Menschen zu unterstützen, ihm Arbeit abzunehmen. Während heutige Roboter so gut wie immer eine spezialisierte Aufgabe erfüllen, muss

ein echter Begleiter aber flexibler sein. In der direkten Interaktion mit dem Menschen müssen Befehle nicht nur entgegengenommen, sondern im Kontext der Umgebung interpretiert werden. Ein Führhund-Roboter müsste nicht nur aus Gestik, Körperhaltung und verbalem Kommando erkennen, dass eine sichere Straßenüberquerung gewünscht ist, sondern auch erfassen, dass erst ein Schalter aktiviert und selbst bei grün erst der vorbeirasende Fahrradkurier abgewartet werden muss. Der schreitende Roboter, so herausfordernd er ist, ist auf jeden Fall erst der Anfang, die Plattform für eine Maschine, die mit uns kommuniziert, interagiert, uns begleitet.

„*Egal wie weit der Weg ist, man muss den ersten Schritt tun.*“ (Mao Tse-tung)[35]

[35] Der Autor distanziert sich ausdrücklich von den sozio-politischen Meinungen Maos.

Literaturverzeichnis

[Bet10] **Bettzüche, Björn. 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.

[Bis06] **Bishop, Christopher M. 2006.** *Pattern Recognition and Machine Learning.* s.l. : Springer, 2006. ISBN 978-0-387-31073-2.

[Byl08] **Byl, Katie. 2008.** *Metastable Legged-Robot Locomotion, Thesis (Ph. D.).* Cambridge, MA : MIT, 2008.

[Efr99] **Efros, A. A. und K., Leung T. 1999.** Texture synthesis by non-parametric. *Computer Vision. Proceedings. IEEE International Conference on.* 1999.

[Est05] **Estremera, Joaquin und Gonzalez de Santos, Pablo. 2005.** *Generating continuous free crab gaits for quadruped robots on irregular terrain.* s.l. : IEEE Transactions on Robotics, 2005. 21(6):1067 – 1076.

[Hil06] **Hilljegerdes, Jens, Spenneberg, Dirk und Kirchner, Frank. 2006.** The Construction of the Four Legged Prototype Robot ARAMIES. [Buchverf.] M. O. Tokhi, G. S. Virk und M. A. Hossain. *Climbing and Walking Robots.* Berlin Heidelberg : Springer, 2006.

[Hon07] **Honda Motor Co., Ltd. September 2007.** *ASIMO Technical Information.* [PDF] s.l. : Honda Motor Co., Ltd., September 2007. kA.

Interactive Learning in Continuous Multimodal Space: A Bayesian Approach to Action-Based Soft Partitioning and Learning. [Fir12] **Firouzi, H., et al. 2012.** s.l. : IEEE, 2012. Autonomous Mental Development, IEEE Transactions on. Bde. Volume:4 , Issue: 2, S. 124-138. ISSN :1943-0604.

[iSt13] **2013.** iStruct: Intelligent Structures for Mobile Robots. *DFKI - Robotics Innovation Center.* [Online] Deutsches Forschungszentrum für künstliche Intelligenz, August 2013. [Zitat vom: 29. September 2014.] http://robotik.dfki-bremen.de/de/forschung/projekte/istruct.html.

[Kal11] **Kalakrishnan, M., et al. 2011.** learning, planning and control for quadruped locomotion over challenging terrain. *International Journal of Robotics Research.* 2011, 30.

[Kim10] **Kim, Kisung, et al. 2010.** Performance Comparison between Neural Network and SVM for Terrain Classification of Legged Robot. *SICE Annual Conference.* 2010.

[Kol09] **Kolter, J. Z., Y., Kim und Y., Ng A. 2009.** Stereo Vision and Terrain Modeling for Quadruped Robots. *IEEE ICRA 2009.* Mai 2009, S. 1557-1564.

[LuL09] **Lu, Liang, et al. 2009.** Terrain Suface Classification for Autonomous Ground Vehicles Using a 2D Laser Stripe-Based Structured Light Sensor. *IEEE/RSJ IROS.* 2009.

[Rai08] **Raibert, Marc, et al. 2008.** *BigDog, the Rough-Terrain Quaduped Robot.* Waltham, MA : Boston Dynamics, 2008.

[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.

[Ruh13] **Ruhnke, Jan. 2013.** *A Controller for Quadruped Locomotion over Rough Terrain - Thesis Outline.* Hamburg : HAW Hamburg, 2013.

[Ruh12] **—. 2012.** *A Walksystem for a Quadruped Rough Terrain Robot, Controller Concepts (Studienarbeit).* Hamburg : Hochschule für Angewandte Wissenschaften, 2012.

[Ruh14] **—. 2014.** *Ein vierbeiniger Roboter für unebenes Gelände.* Fachbereich Infomatik, Hochschule für Angewandte Wissenschaften. Hamburg : s.n., 2014. Master Thesis.

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

[She05] **Sherstov, Alexander A. und Stone, Peter. 2005.** Function Approximation via Tile Coding: Automating Parameter Choice. [Buchverf.] Jean-Daniel Zucker und Lorenza Saitta. *Abstraction, Reformulation and Approximation.* Berlin : Springer Berlin Heidelberg, 2005, S. 194-205.

[Sut98] **Sutton, Richard S. und Barto, Andrew C. 1998.** *Reinforcement Learning: An Introduction.* Cambridge : A Bradford Book, 1998. ISBN 978-0262193986.

[Sut12] **Sutton, Richard S. und Barto, Andrew G. 2012.** *Reinforcement Learning: An Introduction.* Cambridge : MIT Press, 2012. ISBN 978-0262193981.

[Tan09] **Tanner, Brian und White, Adam. 2009.** RL-Glue : Language-Independent Software for Reinforcement-Learning Experiments. *Journal of Machine Learning Research.* September 2009, Bd. 10, S. 2133-2136.

[Wol08] **Wolter, Anne. 2008.** *Reinforcement Learning in der Roboternavigation.* Saarbrücken : VDM Verlag Dr. Müller, 2008. ISBN 978-3639047028.

Anhang A

A.1 Implementierung

Die Umsetzung des Lernverfahrens zur Auswahl sicherer Schrittpositionen für den vierbeinigen Roboter AMEE erfolgte auf Basis des entwickelten Frameworks für Reinfocement Learning, der RLLibrary.NET. Die folgenden Abschnitte erläutern das Softwaredesign des Lernmoduls und demonstrieren gleichzeitig eine Verwendung des Frameworks.

Die Grundkomponenten von Sarsa(λ) bestehen aus dem Agenten und der Umwelt. Die gesamte simulierte Sensorik zur Umgebungserfassung und der Trittanalyse als Rückmeldung ist Bestandteil der Umwelt-Komponente. Der Agent verbessert seine Strategie in Form der Q-Funktion (State-Value-Function) aufgrund der Rückmeldungen von der Umwelt.

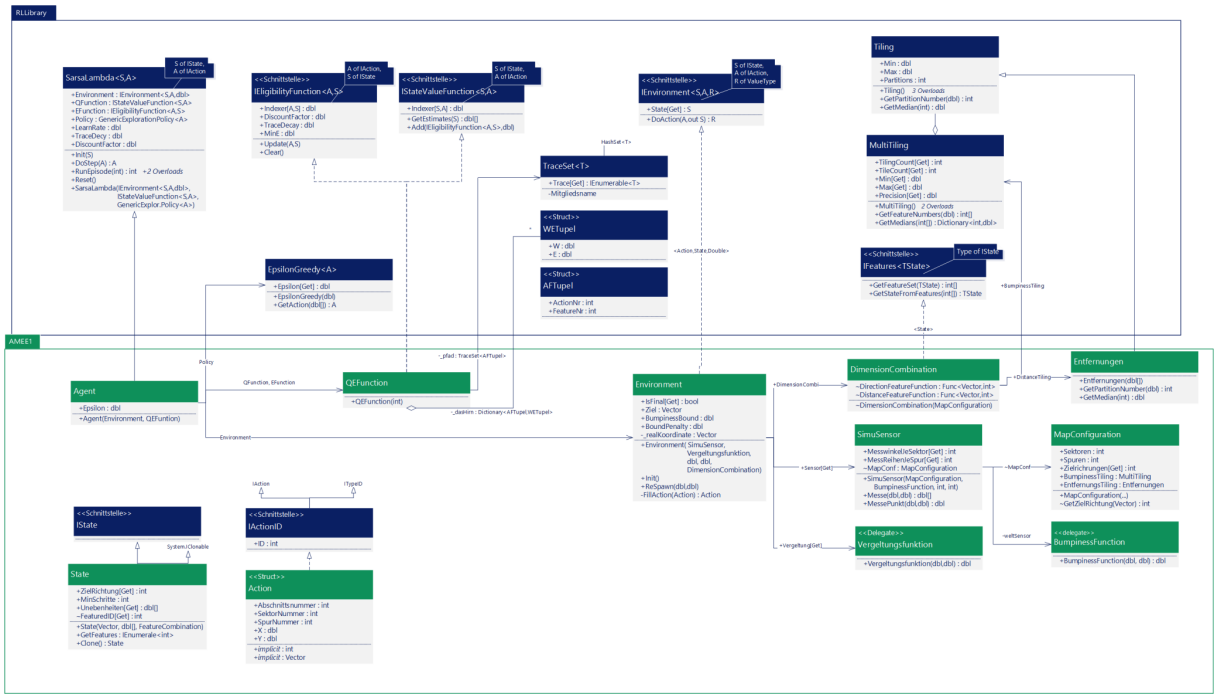


Abb. A 1: Klassendiagramm – Gesamtübersicht der AMEE-Implementierung des Lernverfahrens.

A.1.1 Sensordaten Vorverarbeitung

Die erste Segmentierung eines kontinuierlichen Zustandsraums betrifft die Umgebungssicht. Ausgehend von einem radialen Scan erfolgt die Unterteilung der zweidimensionalen Oberfläche nach Winkel und Radius, sodass klar voneinander abgegrenzte Abschnitte entstehen (siehe **Abb. A 2**). Durch die Winkeleinteilung entstehen Tortenstücke, die als Sektoren bezeichnet werden und die Ringe, die durch die Radien als Grenzen entstehen, werden als Spuren bezeichnet, womit ein Abschnitt dem Schnittbereich einer Spur und eines Sektors entspricht. Diese grundsätzliche und fixe Unterteilung in Spuren und Sektoren wird initial in der Klasse **`MapConfiguration`** (siehe **Abb. A 3**) definiert.

Im Experiment wird der Sensor, der die Umgebung erfasst, durch **`SimuSensor`** simuliert. Seine prinzipielle Aufgabe ist es, beispielsweise aus einem Tiefenbild des Untergrunds, Maße zur geographischen Beschaffenheit der einzelnen Abschnitte wiederzugeben. Die Experimente arbeiten hier mit lediglich einem fiktiven Maß der Unebenheit, wobei null für spiegelglatt und eins für extrem rau oder unpassierbar steht. Die simulierte Messung und Datenvorverarbeitung erfolgt hier durch die Definition mehrerer Messpunkten innerhalb eines jeden Abschnitts (**Abb. A 2**). Zu jeder Messkoordinate gibt eine **`BumpinessFunction`** (vgl. **A.3**) ein Unebenheitsmaß, die alle zusammen gemittelt die Unebenheit des Abschnitts ergeben. Die Abschnittsunebenheiten sind dann ein wesentlicher Bestandteil des RL-Zustands. Eine mögliche Ausprägung einer BumpinessFunction ist in **Abb. A 4** abgebildet.

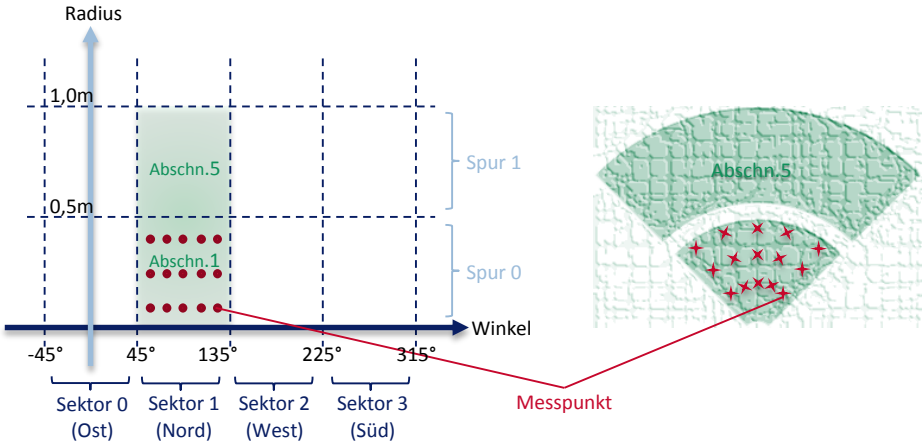


Abb. A 2: Unterteilung des umgebenden Untergrunds in Sektoren und Spuren, die wiederum die einzelnen Abschnitte definieren. Der SimuSensor definiert zu jedem Abschnitt eine Reihe von Messpunkten, anhand derer und eine BumpinessFunction die Abschnittsunebenheit ermittelt wird.

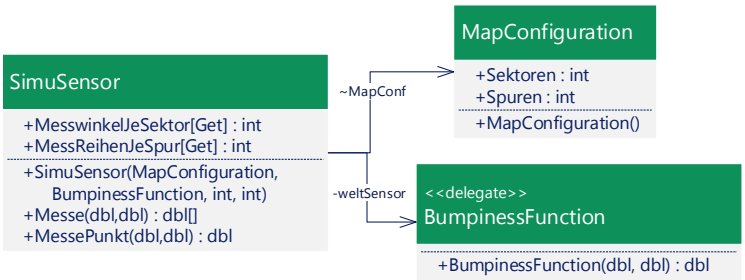
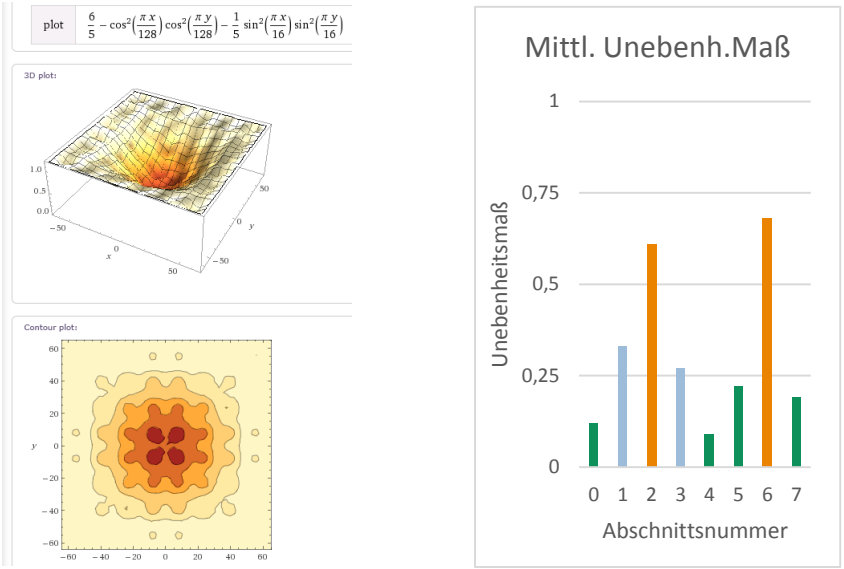


Abb. A 3: Klassendiagramm des SimuSensor, zur simulierten Umgebungsanalyse. Dem Konstruktor muss die MapConfiguration, welche die Unterteilung in Sektoren und Spuren definiert, eine BumpinessFunction, welche eine Unebenheitsanalyse simuliert, sowie die Anzahl der Messpunkte je Sektor und Spur, woraus sich die Messpunkt-Koordinaten je Abschnitt errechnen, übergeben werden.



(a) (b)

Abb. A 4: (a) Eine BumpinessFunction - Simulierte Messresultate der „Realwelt“ als Funktion von (x,y)-Koordinaten. (b) Beispielhafte Unebenheitsmaße je Abschnitt, die sich aus dem Mittelwert der BumpinessFunction Ergebnisse zu den Messpunkt-Koordinaten eines Abschnitts ergeben. Diese sind wesentlicher Bestandteil des RL-Zustands.

A.1.2 Segmentierungen (Tiling)

Jede Dimension des Zustandsraums – Richtung, Entfernung, Unebenheiten – ist reellwertig und muss diskretisiert werden. Die MapConfiguration hält die Definitionen zur Diskretisierung.

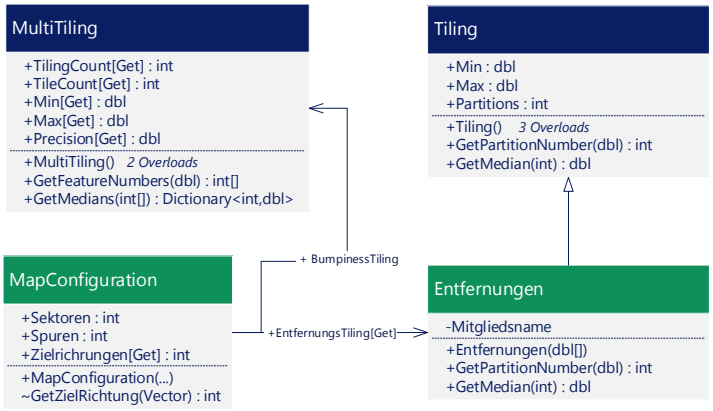


Abb. A 5: Klassendiagramm zur MapConfiguration, erweitert um Diskretisierungsdefinitionen. Blau: Klassen aus RLLibrary.NET

Die Richtung

Der RL-Zustand beinhaltet eine grobe Zielrichtung, die den Sektoren entspricht. Das heißt, das Etappenziel befindet sich irgendwo in dem Winkelbereich, der durch den entsprechenden Sektor definiert wird. Die erste Implementierung des Zustands $\boldsymbol{s}$ orientiert dabei den ersten Sektor immer entlang der X-Achse, die der Himmelsrichtung Ost entspricht und teilt dann die Radialkarte in gleichwinklige „Tortenstücke“, von *Null* bis *#Sektoren-1* nummeriert, auf. Die Zielrichtung ist auf die Sektornummer abstrahiert, in dessen Sektor der Vektor Agent→Ziel liegt.

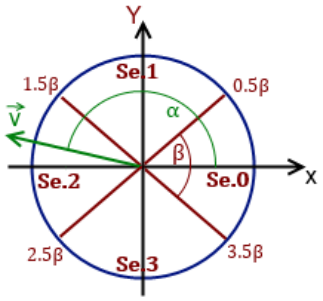
Abbildung Zielvektor auf Sektornummer:

Sektorbreite $\beta = \frac{2\pi}{\#Sekt.}$

Agent→Ziel[36] $\vec{v}_{AgentZiel} = \vec{v}_{Ziel} - \vec{v}_{Agent}$

Winkel α von $\vec{v}_{AgentZiel}$: $\text{atan2}\left(\frac{y}{x}\right) | + 2\pi \; wenn \; \alpha < 0$

SektorNr = $\left\lfloor \frac{\alpha}{\beta} + 0{,}5 \right\rfloor$ (kaufm. Rundung)



Diese Abbildung ist als Hilfsfunktion in MapConfiguration implementiert, da hier auch die Unterteilung in Sektoren und Spuren definiert ist. Der Vektor $\vec{v}_{AgentZiel}$ wird als Parameter übergeben.

Distanz zum Ziel

Auf eine Verwendung von (Multi-)Tiling für die Zielentfernung, die die Entfernung zum Etappenziel als minimale Schrittzahl angibt, wurde unter anderem aus Gründen der Vereinfachung vorerst verzichtet. Darüber hinaus ist das MultTiling der RLLibrary.NET bisher nur auf eine gleichmäßige Partitionierung eines Wertebereichs ausgelegt. Für die Entfernung wäre aber eine logarithmische Skalierung wahrscheinlich sinnvoller. Daher erbt die Klasse **`Entfernungen`** von der einfachen Bibliotheksklasse `Tiling` und überschreibt die Methoden `GetPartitionNumber` und `GetMedian`. Dem Konstruktor teilt man die einschließlichen Obergrenzen der Distanzbereiche mit. Das Entfernungen-Tiling ist dann Bestandteil der `MapConfiguration`.

Abschnittsunebenheit

Jeder Abschnitt ist mit einem reellwertigen Unebenheitsmaß zwischen 0,0 und 1,0 versehen, dessen Diskretisierung durch ein MultiTiling aus der RLLibrary.NET erfolgt.

[36] $\vec{v}_{AgentZiel}$ liegt im Koordinatensystem der „Realwelt“ und wird gedanklich in das Radialsystem transponiert.

A.1.3 Dimensionsverknüpfung und der Zustand

Die Art und Weise der Verknüpfung aller Dimensionen des Zustandsraums, in der ersten Implementierung rein konjunktiv, wird in der Klasse **`DimensionCombination`** umgesetzt. Als Basis dient die RLLibrary-Schnittstelle `IFeatures`, dessen Methode `GetFeatureSet` eine Zustandsinstanz übergeben wird und die zugehörigen Merkmalsnummern zurückgibt. `DimensionCombination` vereint hierzu die zuvor beschriebenen Segmentierungen, beziehungsweise Funktionen zur Berechnung der jeweiligen Merkmalsnummer, die bei der Instanziierung über die `MapConfiguration` übergeben werden. Jedes Tiling einer Dimension gibt eigene von null beginnende Indizes, die in der Implementierung von GetFeatureSet miteinander zu der letztlichen Merkmalsnummer verrechnet werden. Würden nur einfache 1-Tilings verwendet, gäbe eine konjunktive Verknüpfung ein eindeutiges Merkmal zu einem Zustand. Da für die Unebenheiten aber ein MultiTiling vorgesehen ist, ist die maximale Anzahl der Merkmale entsprechend der Anzahl (versetzter) Tilings der Unebenheit. Ist ein einzelner Tiling-Index `Tiling.OutOfBounds`, so ist die gesamte Verknüpfung außerhalb und wird nicht gezählt. Die Methode `GetFeatureSet` ist virtuell[37], sodass andere Kombinationen durch Vererbung umgesetzt werden können.

Der Zustand hält die „rohen" Eigenschaften, wobei Richtung und Distanz von vorn herein entsprechend ihrer Segmentierung zur Verfügung gestellt werden. Bei der Instanziierung – später durch die Umwelt-Komponente – werden aus den ursprünglichen Abschnittsunebenheiten und einem 2D-Vektor, der die Richtung und Entfernung des Agenten zur Zielkoordinate definiert, mittels der übergebenen `FeatureCombination` die Merkmale berechnet, die später mittels `GetFeatures()` abgefragt werden können. `FeatureID` ist ein aus den Merkmalsnummern generierter Hash-Code, der den Vergleich zweier Zustände vereinfacht.

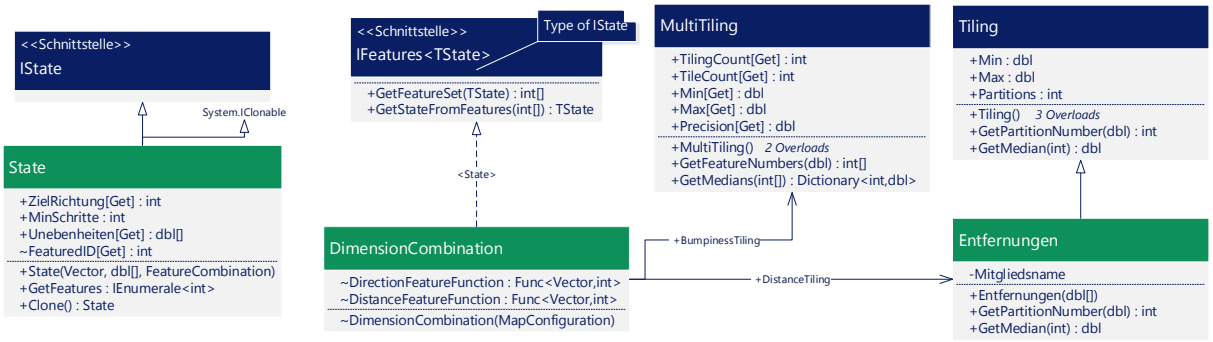


Abb. A 6: Klassendiagramm des Zustand-Objekts und der Dimensionsverknüpfung – eine konjunktive Verknüpfung der Zustandsdimensionen, die aus den Tilings die zu einem Zustand gehörigen Merkmalsnummern errechnet.

[37] Im C#-Sprachjargon kennzeichnet das Schlüsselwort virtual Methoden, die von Derivaten überschrieben werden können. (C# Referenz, zuletzt Besucht: 06. Okt. 2014)

A.1.4 Environment

Eine visuelle Außenwahrnehmung mit Datenvorverarbeitung wird bereits mit `SimuSensor` simuliert. Die Eigenwahrnehmung (Propriozeption) aus der die Vergeltung eines Schrittes ermittelt wird, ist durch eine **`Vergeltungsfunktion`** simuliert, die Zielnäherung und Unebenheit auf den Trittpunkt zu Kosten verrechnet (vgl. Fehler! Verweisquelle konnte nicht gefunden werden.). Außen- und Eigenwahrnehmung sind Bestandteil der Umweltkomponente **`Environment`** (siehe **Abb. A 7**). Sie kennzeichnet sich durch die Schnittstellenmethode `DoAction` aus, die zu einer aus-zuführenden Aktion den Folgezustand und die Vergeltung wiedergibt.

Ein Aufruf von `DoAction` erfolgt üblicherweise vom Agenten aus mit einer Aktion, die durch die `ExplorationPolicy` vorgegeben wurde und die bis dahin normalerweise nur eine ID besitzt, die der Abschnittsnummer entspricht. Daher wird das `Action`-Objekt intern mit den weiteren zugehörigen Daten, wie relative Koordinaten zur aktuellen Position, Sektor- und Spurnummer aufgefüllt. Abhängig davon, ob die Unebenheit an der Aktionskoordinate nicht zu stark ist, was mit `SimuSensor.MessePunkt` ermittelt wird, folgt die Schrittausführung, indem zunächst die aktuelle Position aktualisiert wird. Anschließend gibt wieder der `SimuSensor` die gemittelten Abschnittsunebenheiten, die zusammen mit dem Vektor von der neuen Position zum Ziel und der `DimensionCombination` zur Erzeugung des Folgezustands genutzt werden. Letztlich werden der Vergeltungsfunktion noch die bereits erfasste Unebenheit an der neuen Position und die Zielnäherung, die sich aus einer Vektoroperation mit alter und neuer Position ergibt, übergeben, um den Vergeltungswert festzulegen. Das Sequenzdiagramm in **Abb. A 8** stellt den Ablauf nochmals schematisch dar.

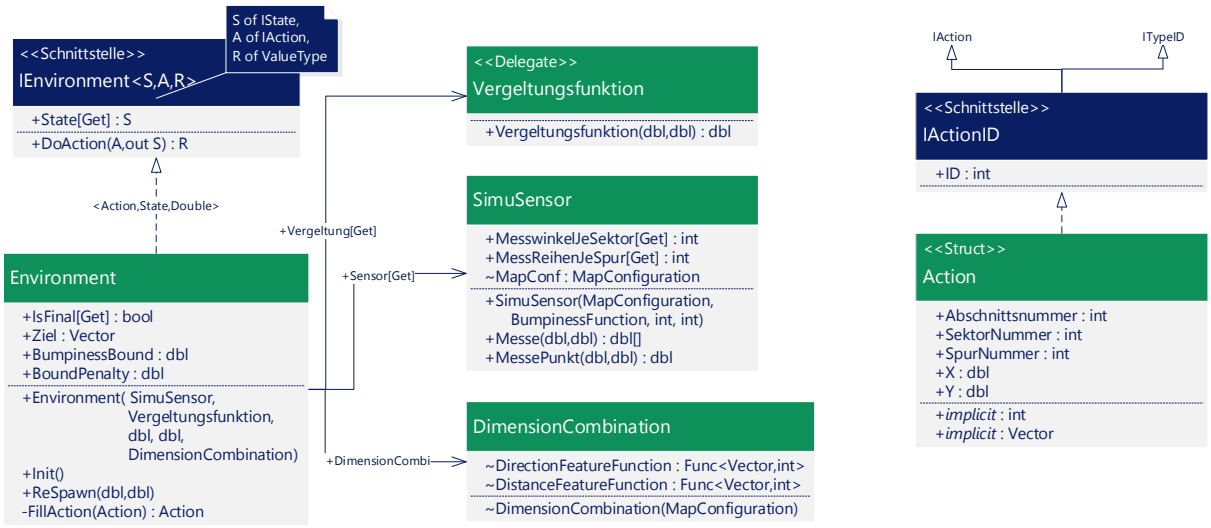


Abb. A 7: Klassendiagramm zur Umwelt-Komponente. Sie vereint die simulierte Sensorik zur Umgebungserfassung und Propriozeption. In DoAction werden Aktionen ausgeführt und Vergeltung sowie Folgezustand generiert.

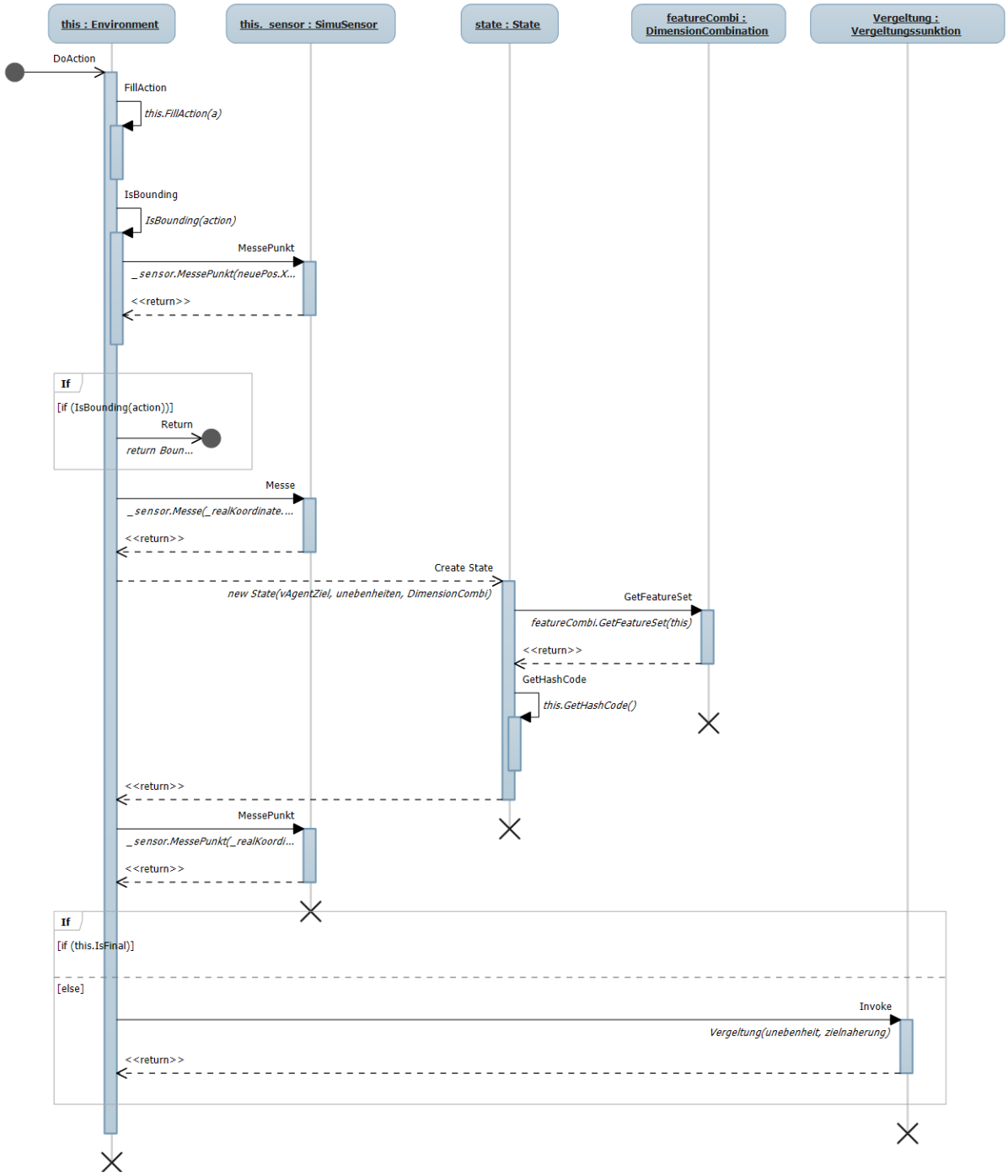


Abb. A 8: Sequenzdiagramm zur Environment-Methode DoAction, wo eine Aktion ausgeführt sowie zugehöriger Folgezustand und Reward (Vergeltung) erzeugt werden.

A.1.5 State-Value Function, Eligibility, Agent

Wesentlicher Bestandteil des Sarsa(λ) Algorithmus sind die Schätzwerte der Funktion Q zu Aktion und Zustand sowie die zugehörigen Verantwortlichkeitsfaktoren der Funktion E. Die RLLibray.NET definiert hierfür jeweils die Schnittstelle `IStateValueFunction` und `IEligibilityFunction` (vgl. Kap. **3.3**), die gemeinsam von der Klasse **`QEFunction`** implementiert werden. Sie verwaltet das „Wissen" in Form der Merkmalsgewichte. Aus Performance optimierenden Gründen (siehe **A.2**) wird keine komplette Matrix von Q- und E-Werten sondern eine Key-Value-Kollektion und ein Pfad genutzt. Letzter hält alle Aktions-Merkmals-Tupel, deren E-Wert ungleich null sind und somit bei der nächsten Aktualisierung die entsprechenden Gewichte und Verantwortlichkeiten (`WETupel`) modifiziert werden müssen.

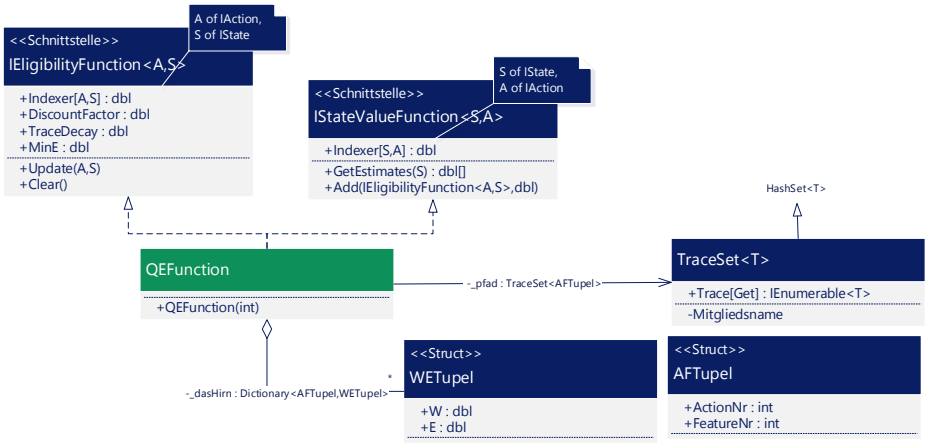


Abb. A 9: Die QEFunction implementiert die State-Value-Function und EligibilityFunction. Die Kollektion _dasHirn halt die Merkmalsgewichte – das "Wissen".

Der Agent

Neben der Verknüpfung zur `Environment`, um Aktionen auszuführen, benötigt der **`Agent`** einer Referenz auf die `QEFunction` und die Definition der Lernparameter. Ansonsten wird von der in der RLLibray vorimplementierten Klasse `SarsaLambda` Gebrauch gemacht, von der einfach geerbt wird. Der Umgang mit kontinuierlichen Dimensionsräumen ist in der Umwelt und der QE-Funktion gekapselt, sodass die standardisierten Methoden, die `SarsaLambda` nutzt, vollständig ausreichen.

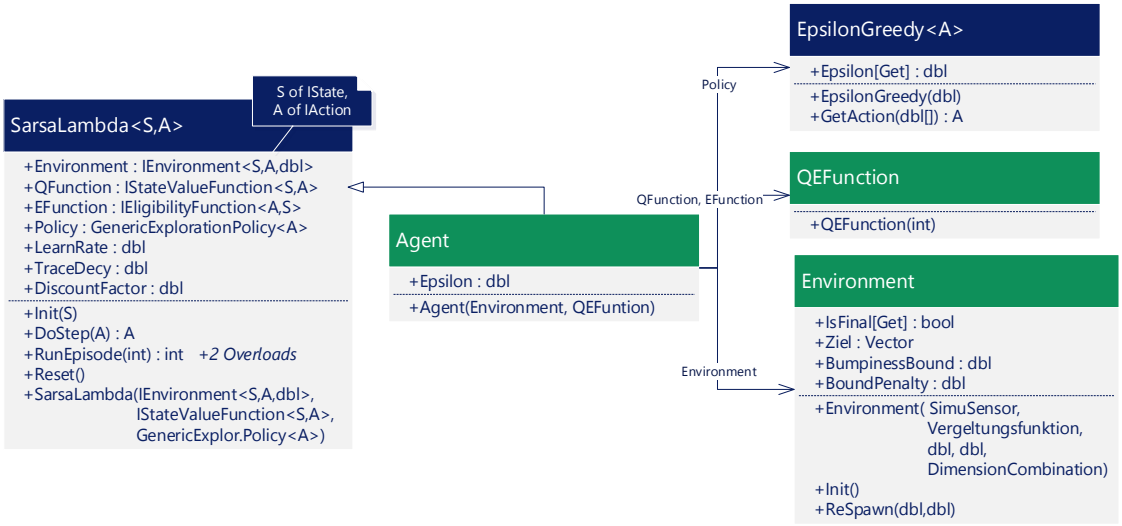


Abb. A 10: Klassendiagramm des Agenten, der von der Standardimplementierung für SarsaLambda erbt.

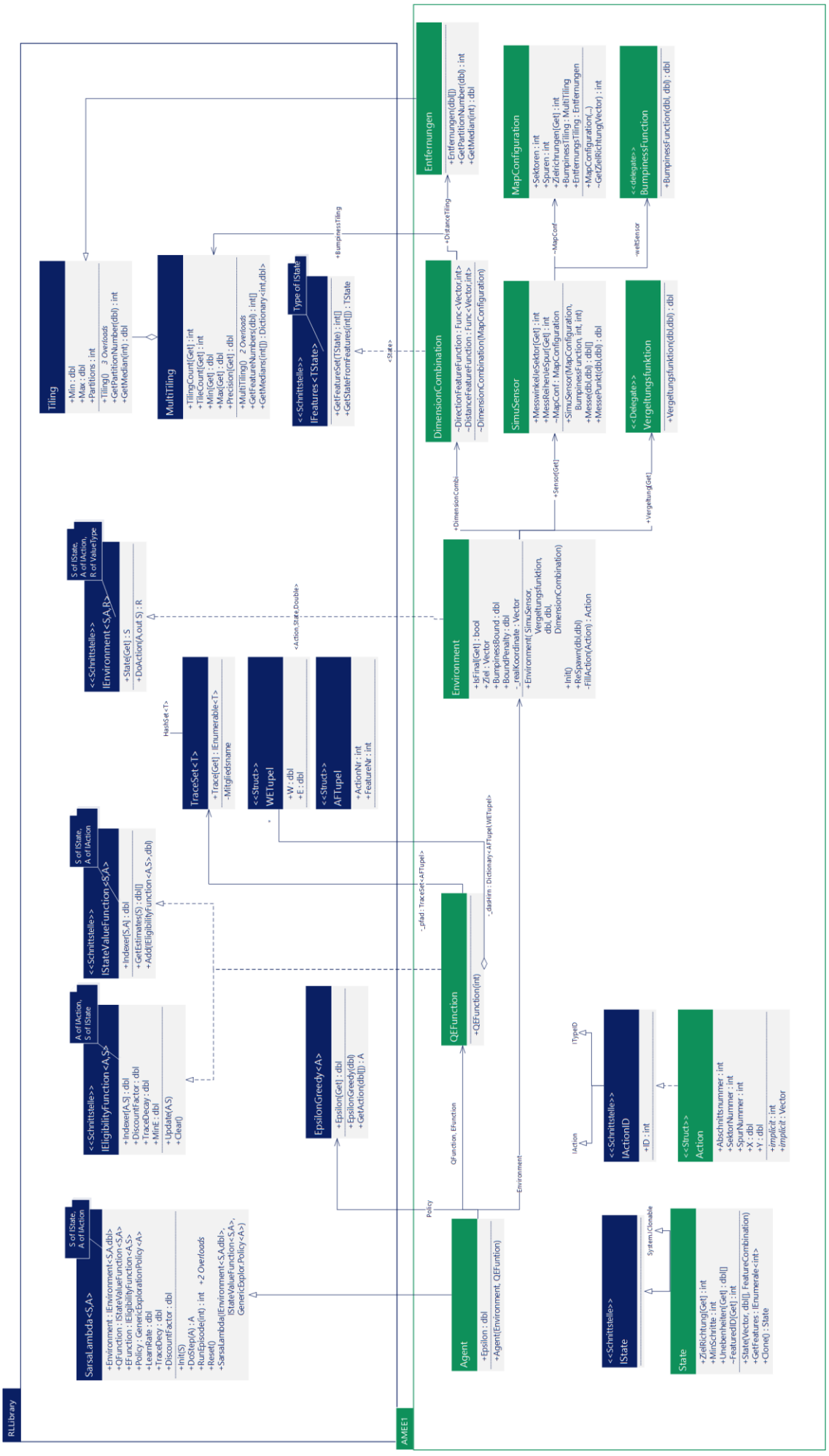


Abb. A 11: Klassendiagramm – Gesamtübersicht der AMEE-Implementierung des Lernverfahrens

A.2 RL-Update – Performance-Optimierung

Die tabellarische (diskrete) Zustandsmodellierung des Dampfross® Testszenarios (siehe **5.3**) besitzt im Normalfall etwa 64 Millionen Merkmale und Eligibility-Werte je Aktion, die bei jedem Lernschritt aktualisiert werden müssen. Zur Erinnerung, die beiden wesentlichen Aktualisierungen sind:

$$\boldsymbol{E} \leftarrow \gamma\lambda\boldsymbol{E} \text{ und } \boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\delta\boldsymbol{E}$$

Bei der diskreten Implementierung entspricht W der $Q(s,a)$-Matrix und bei der kontinuierlichen Umsetzung der Matrix von Merkmalsgewichten. Allerdings sind die e-Werte aus der Eligibility-Matrix E überwiegend Null, was eine Aktualisierung der korrespondierenden Gewichte überflüssig macht, da sie sich nicht ändern. Um die Berechnung zu beschleunigen, ist es daher sinnvoll, nur diejenigen Werte zu aktualisieren, deren e-Wert ungleich Null ist. Dafür kann eine Listenstruktur genutzt werden, die den zurückgelegten Zustands-Aktions-Pfad speichert. Da bei Linear Gradient Descent Sarsa(λ) die Zustände in Merkmale überführt werden, speichert die Pfad-Struktur entsprechend Merkmals-Aktions-Tupel. Da jedes Tupel genau ein Matrix-Element indiziert, sowohl in E als auch in W, bietet sich ein HashSet für eine entsprechende Tupelstruktur an. Die Tupelstruktur sollte selbstverständlich die Hash-Methode, basierend auf der Zustands- beziehungsweise Merkmalsnummer und Aktions-ID, implementieren. Die erstellte RLLibrary.NET (siehe **3.3**) bietet die generische Klasse TraceSet an, die die native HashSet um einen Enumerator erweitert, mit dem die Kollektion vom neuesten zum ältesten Element durchlaufen werden kann. Bei jedem Lernzyklus wird aus dem gerade aktuellem Zustand und der aktuelle Aktion ein neues Tupel erzeugt und dem TraceSet-Pfad hinzugefügt. Für die Matrix-Aktualisierungen können dann genau und nur die Elemente durchlaufen werden, die sich im Pfad befinden. Unterschreitet dabei ein e-Wert das vorgesehene Minimum und wird deswegen auf null gesetzt, ist das Zustands-Aktions-Paar nicht mehr relevant und wird aus dem Pfad entfernt.

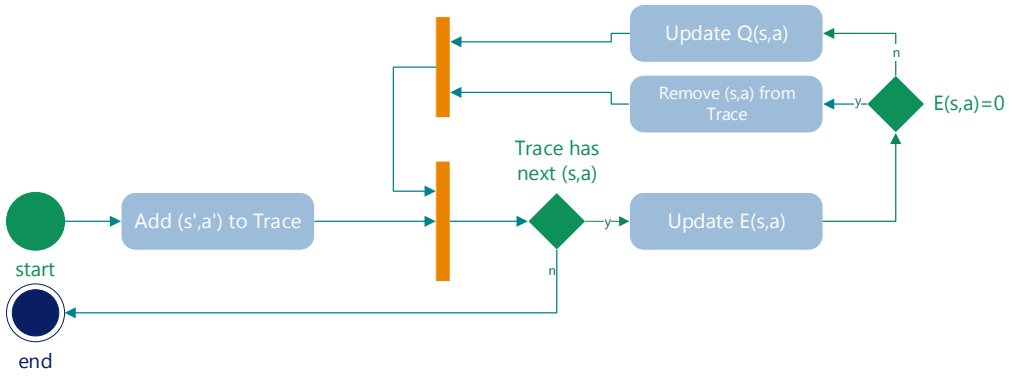


Abb. A 12: Aktivitätsdiagramm zur Performance-Optimierung der Matrix-Aktualisierungen mit einem TraceSet (Zustands-Aktions-Pfad). Zu Beginn wird das gerade aktuelle Zustands-Aktions-Paar dem Pfad hinzugefügt. Danach bezeichnet (s,a) ein beliebiges Tupel im Pfad.

Speicheroptimierung

Für die Experimente in Abschnitt **6** wurde eine zusätzliche Optimierung der Speicherverbrauch vorgenommen, indem statt je einer W- und E-Matrix ein Key-Value Kollektion genutzt wurde (Dictionary in C#, äquivalent zu Map in Java). Über ein Merkmals-Aktions-Tupel indiziert, hält die Kollektion Tupel-Strukturen mit Gewicht und Verantwortlichkeitswert (eligibility), WETupel genannt. Die Kollektion wird leer initialisiert und mit jedem Lernschritt sukzessive gefüllt, sodass nur Elemente gespeichert werden, die ein Gewicht ungleich Null haben. Für Merkmals-Aktions-Tupel, die bisher nicht vorkamen, wird mit dem Standard-WETupel gerechnet, dessen beiden Werte Null sind.

A.3 Weltfunktionen

In Abschnitt **4.1.3** wird ein analytischer Umgebungsscan durch eine Funktion $welt\colon \mathbb{R}^2 \rightarrow [0;1] \subset \mathbb{R}$ simuliert, die zu einer Weltkoordinate $(\mathrm{x},\ \mathrm{y})$ ein Unebenheitsmaß u liefert. Bei den folgenden beispielhaften Ausprägungen, die für Tests verwendet wurden, wurde stets darauf Wert gelegt, dass die Welt durch einen Bereich maximaler Unebenheit begrenzt wird. Die Zeichnungen wurden mit Wolfram Alpha erstellt. Man kann sich die Weltfunktion als analytisches Ergebnis der Sensordaten vorstellen, das in der robotereigenen Karte, der Agentenwelt, festgehalten und weiterverarbeitet wird. Die Abbildungen der Weltfunktion 1-4 zeigen zunehmend komplexere Beispiele auf denen der RL-Algorithmus getestet wird. Die Funktionen sind derart konzipiert, dass sie in der Nähe des Ebenenursprungs eine möglichst glatte $(u \rightarrow 0)$ und nach außen hin eine sehr unebene Oberfläche $(u \rightarrow 1)$ widergeben. Das Ziel ist es, dass der Agent zu gegeben Start- und Zielkoordinaten einen möglichst kurzen aber auch ebenen Pfad erlernt.

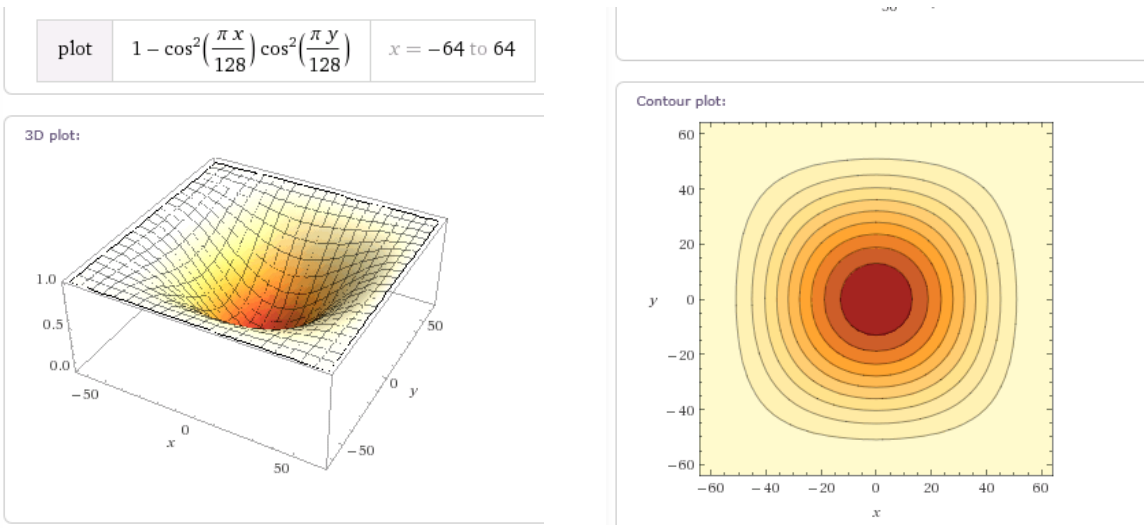


Abb. A 13: Weltfunktion 1 - 3D und Kontur Ansicht. Dunkel für glatte, hell für raue Oberfläche

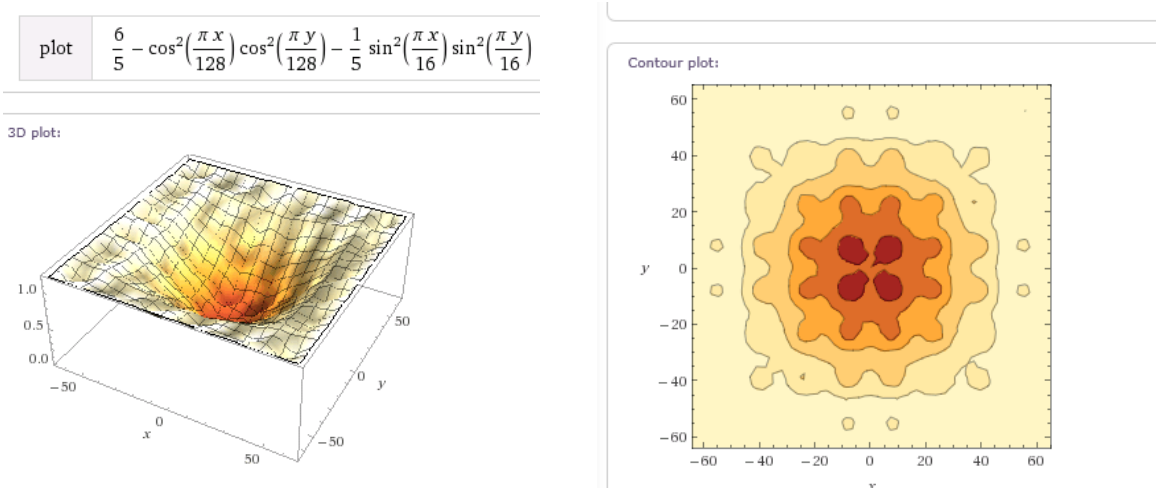


Abb. A 14: Weltfunktion 2

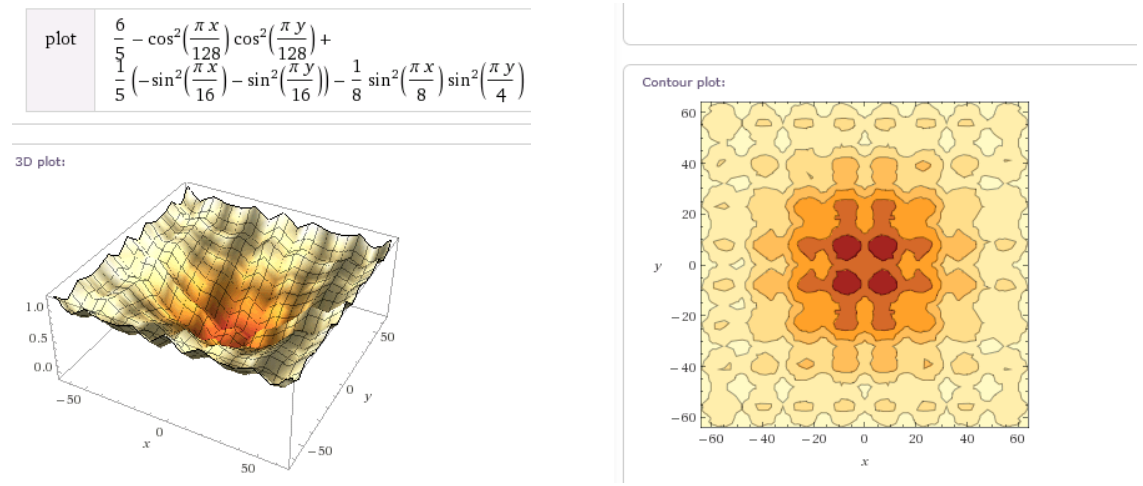


Abb. A 15: Weltfunktion 3 – 3D und Kontur Ansicht. Dunkel für glatte, hell für raue Oberfläche

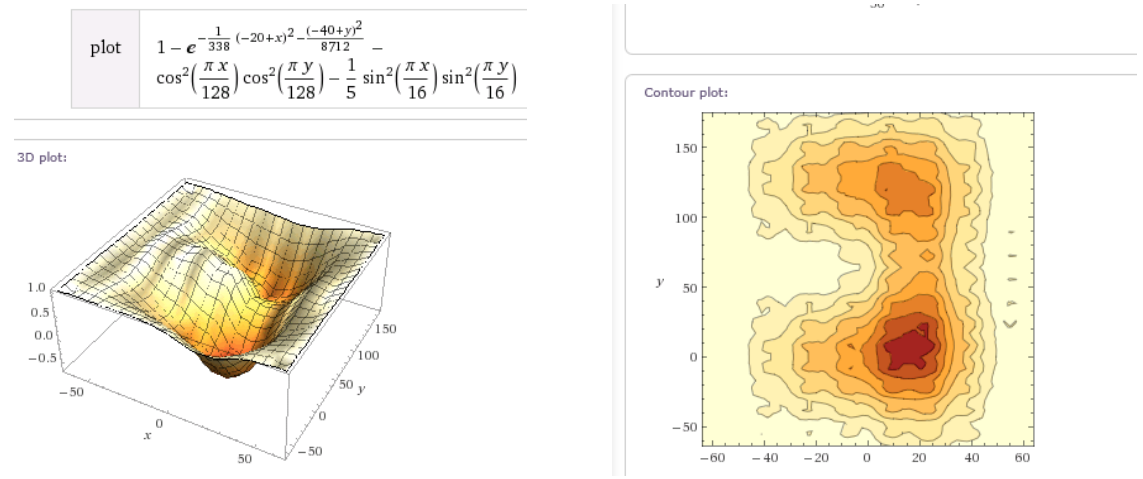
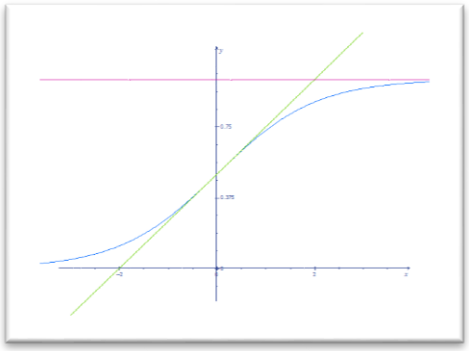


Abb. A 16: Weltfunktion 4

A.4 Vergeltung- / Sigmoidfunktion

In **4.1.2** wird eine Reward-Funktion zur Bewertung eines Roboterschritts auf Basis von Unebenheit u vorgestellt. Für die Simulation bietet sich unteranderem eine Sigmoidfunktion an. Eine Standardvariante ist:

$$sig(x) = \frac{1}{1+e^{-x}}$$



Eine Sigmoidfunktion von -1 bis +1:

$$sig(x) = \frac{2}{1+e^{-Tx}} - 1$$

Tangente am Nullpunkt:

$$Tng_0(x) = \frac{T}{2}x \;\Rightarrow\; Tng_0\left(^{+}/_{-}\frac{2}{T}\right) = {}^{+}/_{-}1$$

„Wachstums“-Parameter$^{+}/_{-}$ T:

Es gilt $sig\left(^{+}/_{-}\frac{2}{T}\right) \cong {}^{+}/_{-}76\%$

Allgemeiner eine Sigmoidfunktion von -N bis +N:

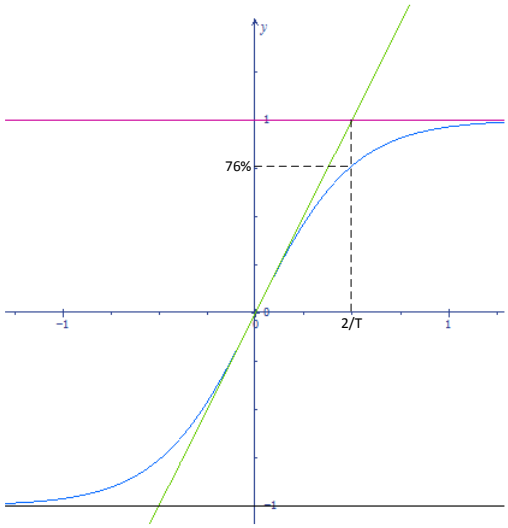
$$sig(x) = N \cdot \left(\frac{2}{1+e^{-Tx}} - 1\right)$$

Tangente am Nullpunkt:

$$Tng_0(x) = \mathrm{N} \cdot \frac{T}{2}x \;\Rightarrow\; Tng_0\left(^{+}/_{-}\frac{2}{N\cdot T}\right) = {}^{+}/_{-}1$$

„Wachstums“-Parameter T:

Es gilt $sig\left(^{+}/_{-}\frac{2}{N\cdot T}\right) \cong {}^{+}/_{-}76\% \cdot N$



A.4.1 Performancesteigerung durch Näherung von e^x

Per Definition gilt:

$$e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \text{ bzw. } e^{-Tx} = \lim_{n\to\infty}\left(1+\frac{-Tx}{n}\right)^n$$

Für z.B. n = 2^4 = 16 ist die Näherung schon ganz ordentlich. Ersetzt man in der Sigmoidfunktion das e^x durch die Näherung mit n= 16 erhält man:

$$fsig(x) = \frac{2}{1+\left(1-\frac{Tx}{16}\right)^{16}} - 1$$

Für die Näherung eine Zweierpotenz für n zu nehmen erleichtert und beschleunigt zusätzlich die Berechnung im Programm:

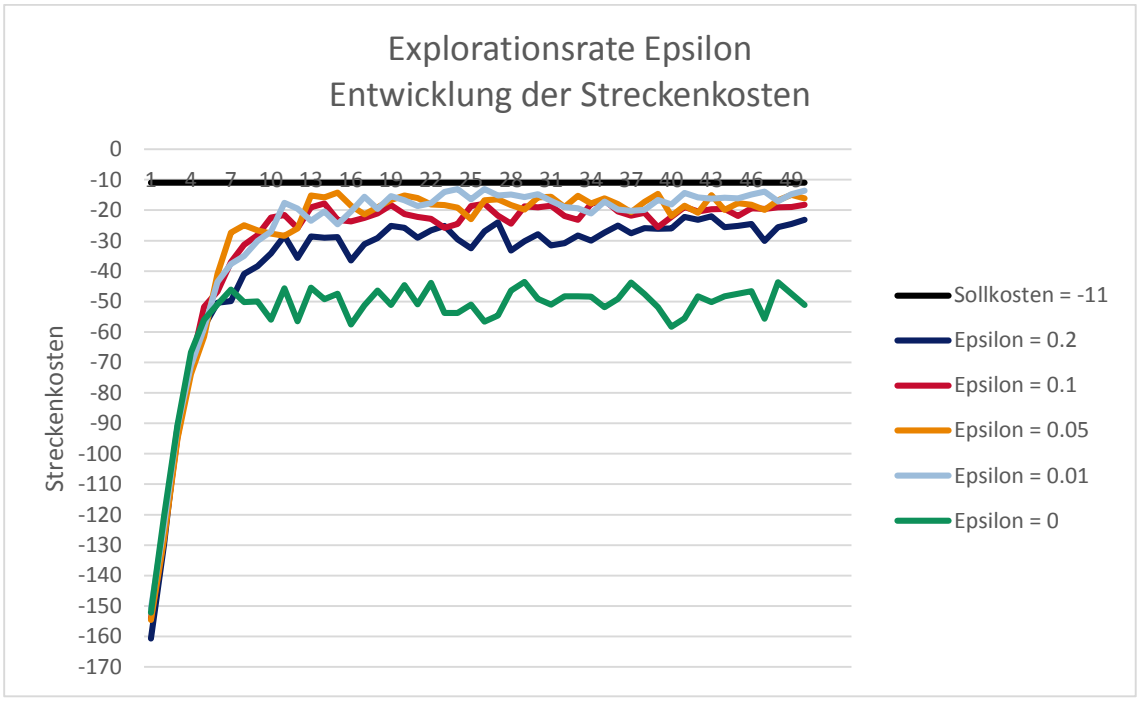
```
static double T = 5.0;
static double Sigmoid(double x) {
     return (2.0 / (1.0 + Math.Pow(Math.E, (-T * x)))) - 1.0;
}
static double FastSigmoid(double x) {
     double temp = 1.0 - (T * x / 16.0);
     temp *= temp; // hoch 2
     temp *= temp; // hoch 4
     temp *= temp; // hoch 8
     temp *= temp; // hoch 16
     return (2.0 / (1.0 + temp)) - 1.0;
}
```

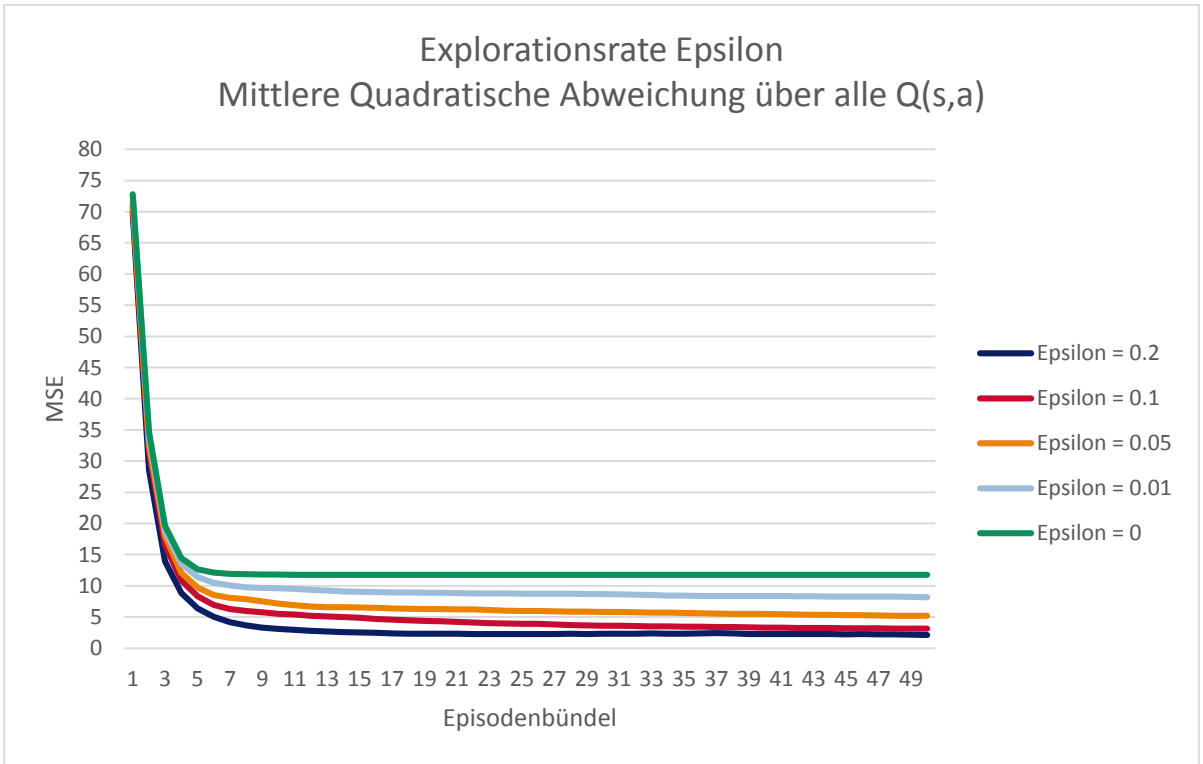
Tests mit Zufallszahlen zwischen -1 und +1 zeigen eine Beschleunigung von etwa 280% bei einer mittleren quadratischen Abweichung von etwa 0,37 ‰. Bei den zufällig gleichverteilten Eingangswerten und Aufgrund der Symmetrieeigenschaften beider Sigmoidfunktionen ist ein durchschnittliches Rechenergebnis von 0,0 zu erwarten. Bei `Sigmoid` ist das Durchschnittsergebnis etwa 0,0001, was auf eine hohe Rechengenauigkeit hindeutet. Dagegen zeigt `FastSigmoid` Anzeichen für höhere Rundungsfehler, denn hier liegt das Durchschnittsergebnis bei etwa 0,0174.

Anhang B Diagramme

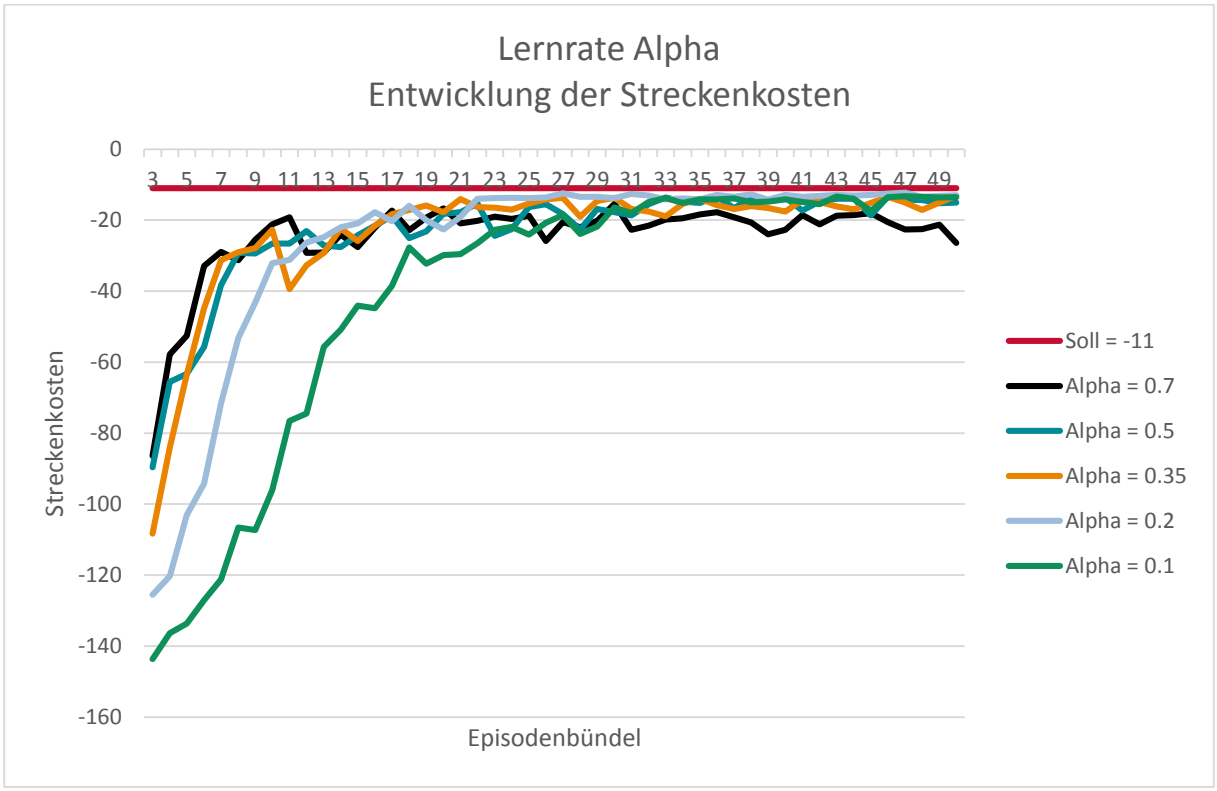
B.1 Versuchsergebnisse RL-Parameter

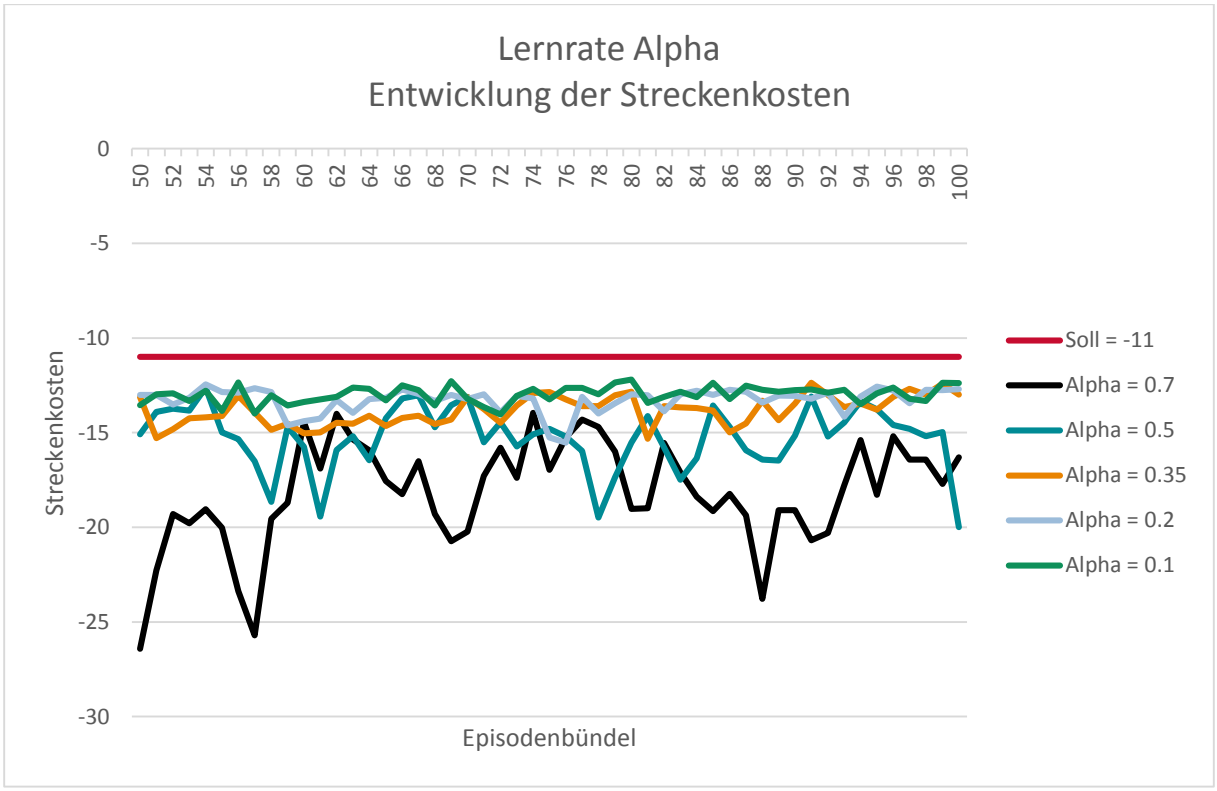
Explorationsrate ε

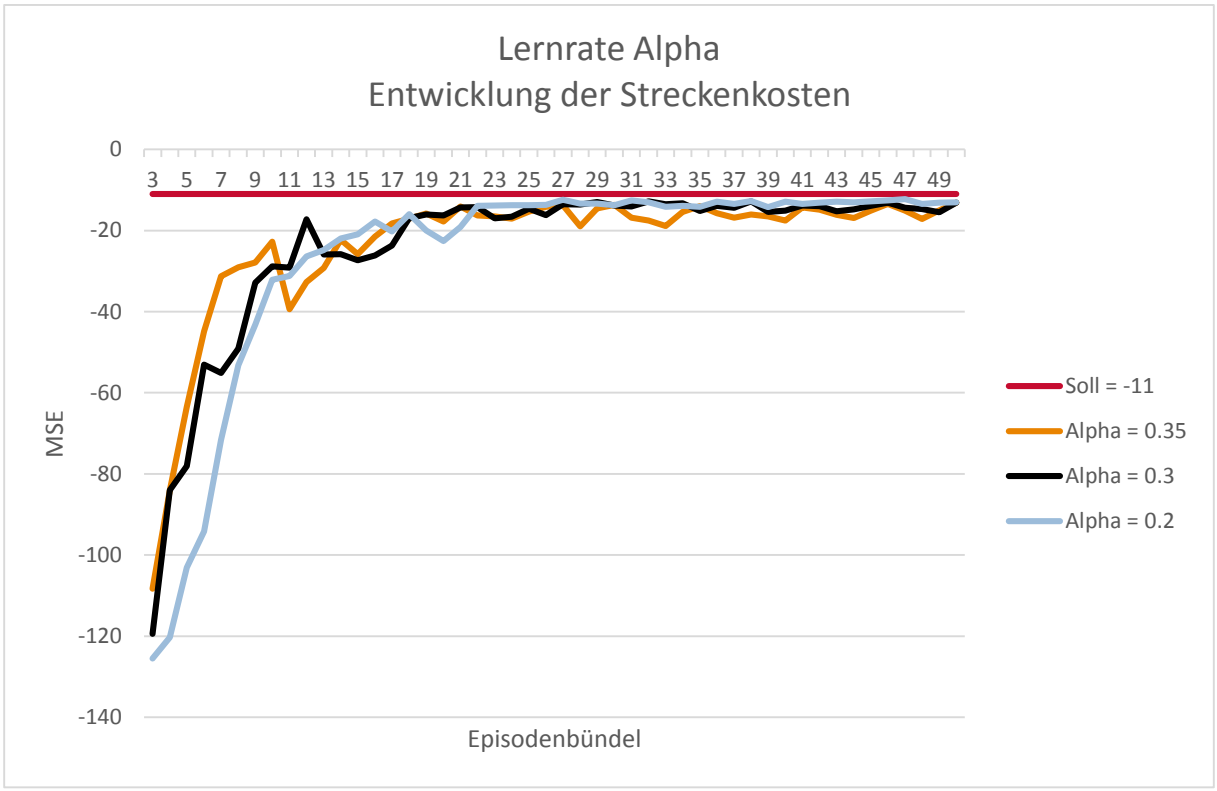


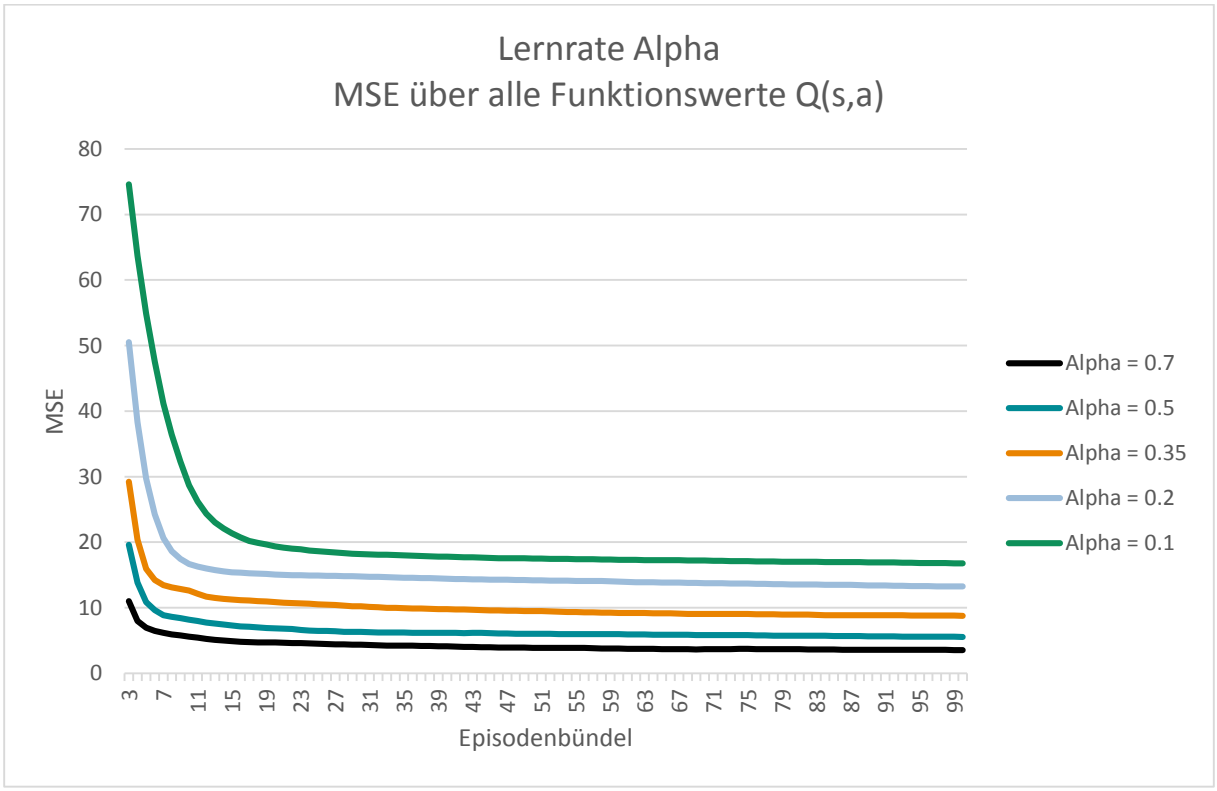


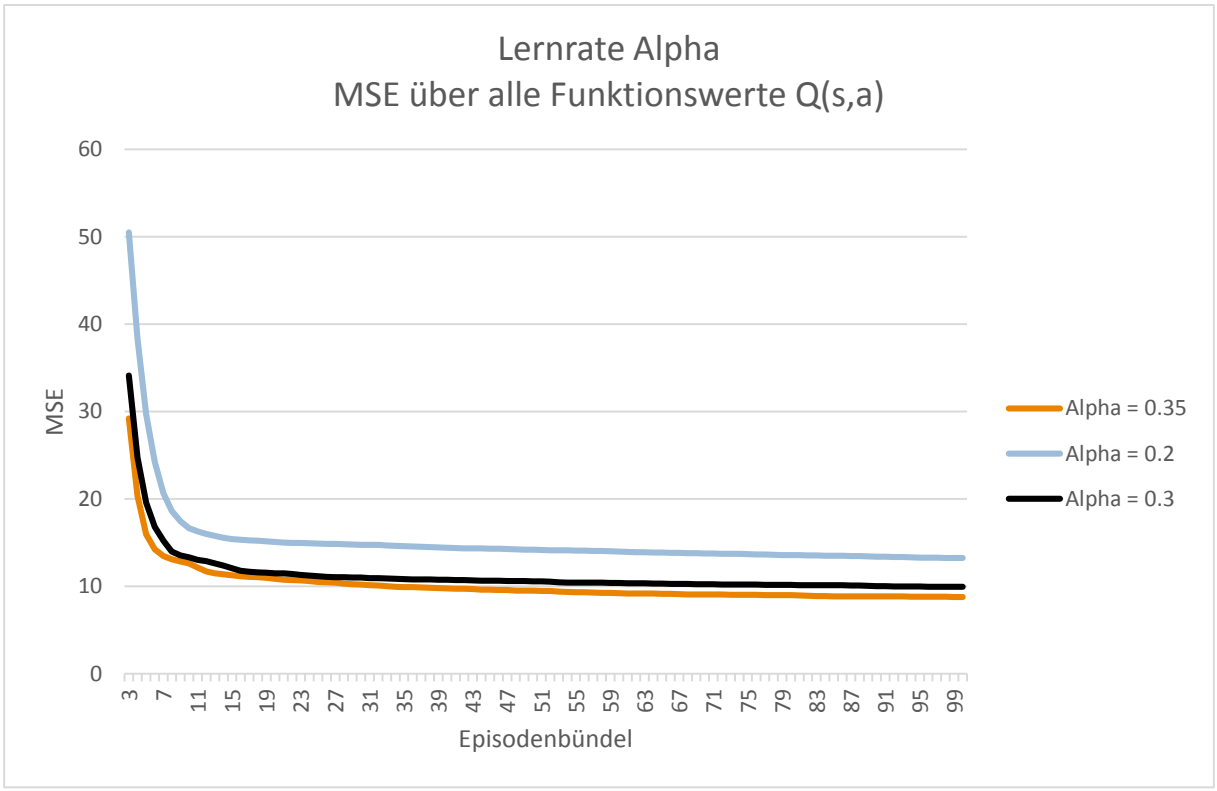
Lernrate α


Lernrate Alpha
Entwicklung der Streckenkosten
0
-20
-40
-60
-80
-100
-120
-140
-160
3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Streckenkosten
Episodenbündel
Soll = -11
Alpha = 0.7
Alpha = 0.5
Alpha = 0.35
Alpha = 0.2
Alpha = 0.1

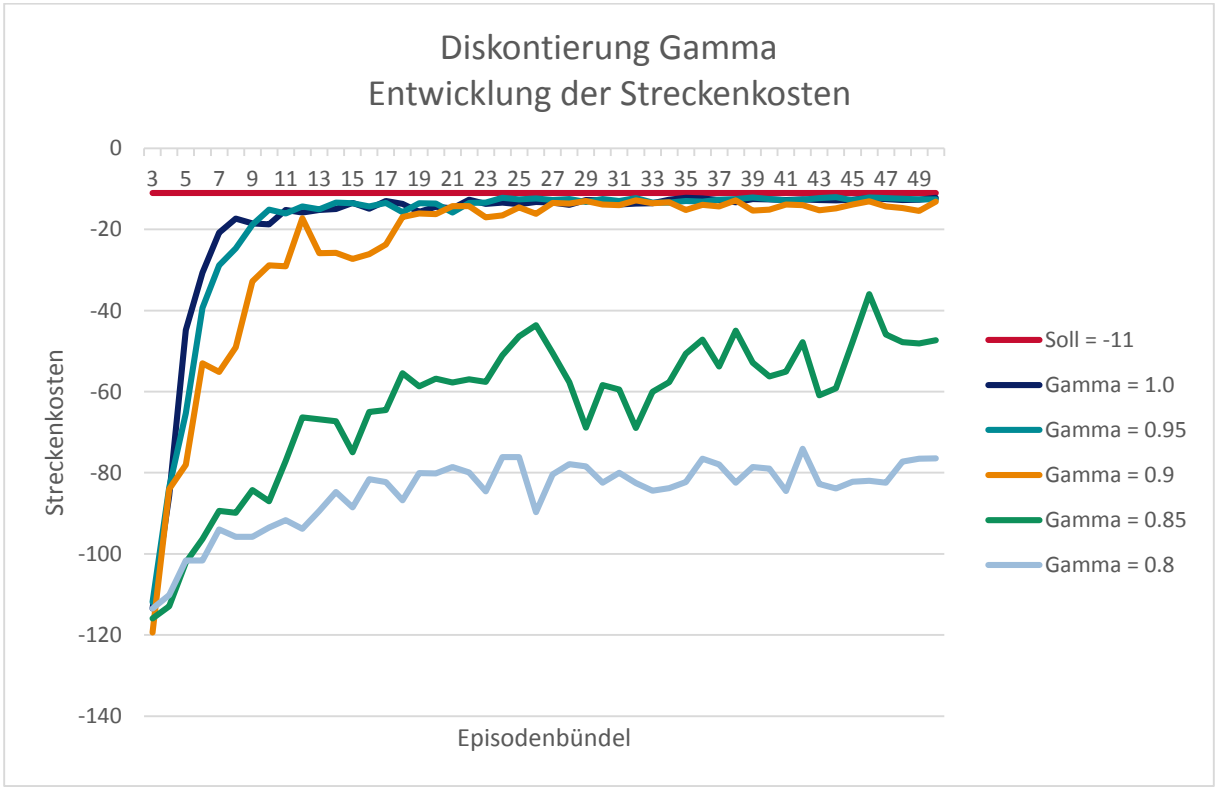

Lernrate Alpha
Entwicklung der Streckenkosten
0
-5
-10
-15
-20
-25
-30
50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100
Streckenkosten
Episodenbündel
Soll = -11
Alpha = 0.7
Alpha = 0.5
Alpha = 0.35
Alpha = 0.2
Alpha = 0.1

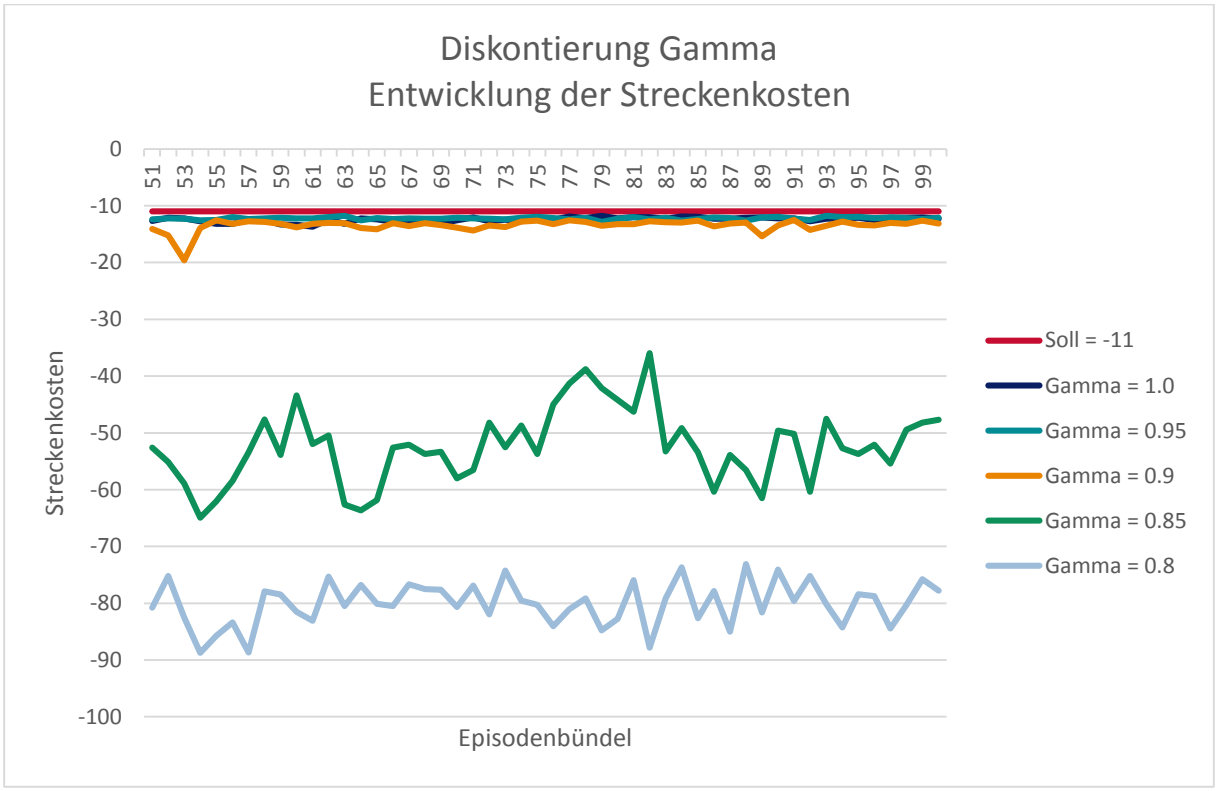

Lernrate Alpha
Entwicklung der Streckenkosten
0
-20
-40
-60
-80
-100
-120
-140
MSE
3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Episodenbündel
Soll = -11
Alpha = 0.35
Alpha = 0.3
Alpha = 0.2

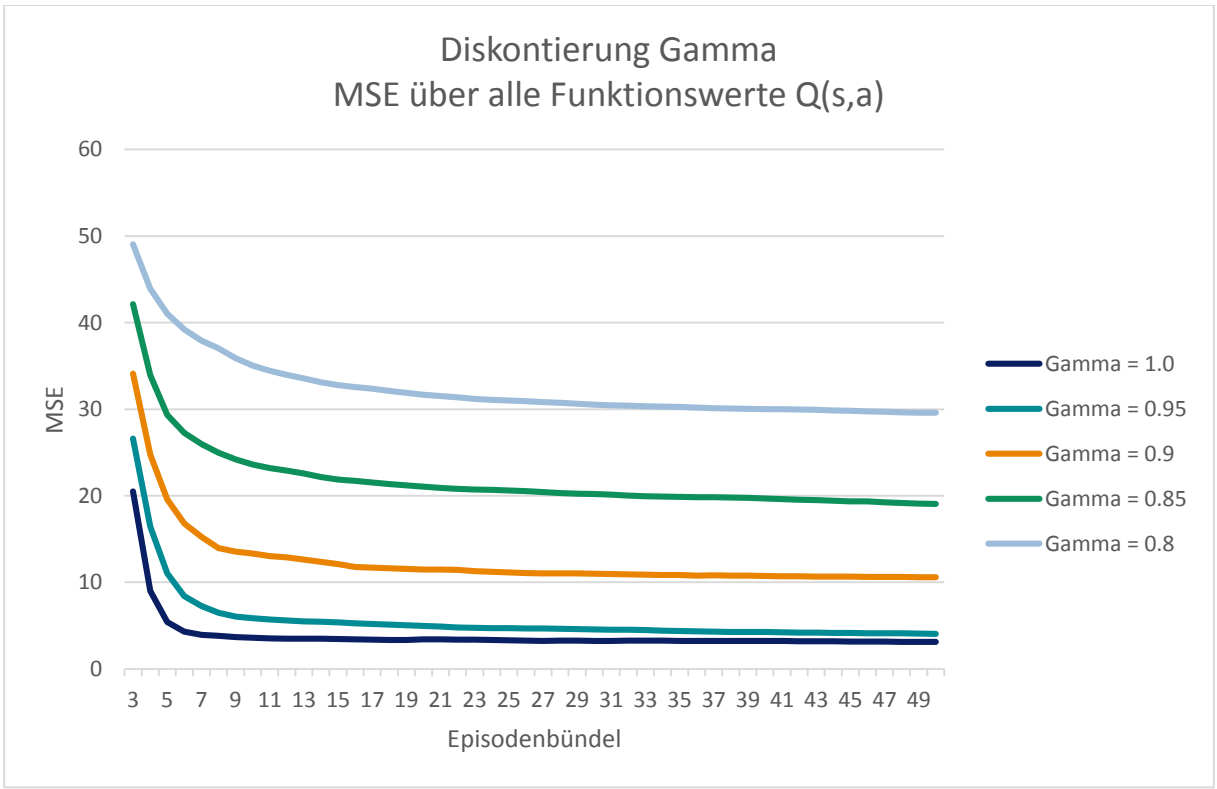

Lernrate Alpha
MSE über alle Funktionswerte Q(s,a)
80
70
60
50
40
30
20
10
0
MSE
3 7 11 15 19 23 27 31 35 39 43 47 51 55 59 63 67 71 75 79 83 87 91 95 99
Episodenbündel
Alpha = 0.7
Alpha = 0.5
Alpha = 0.35
Alpha = 0.2
Alpha = 0.1



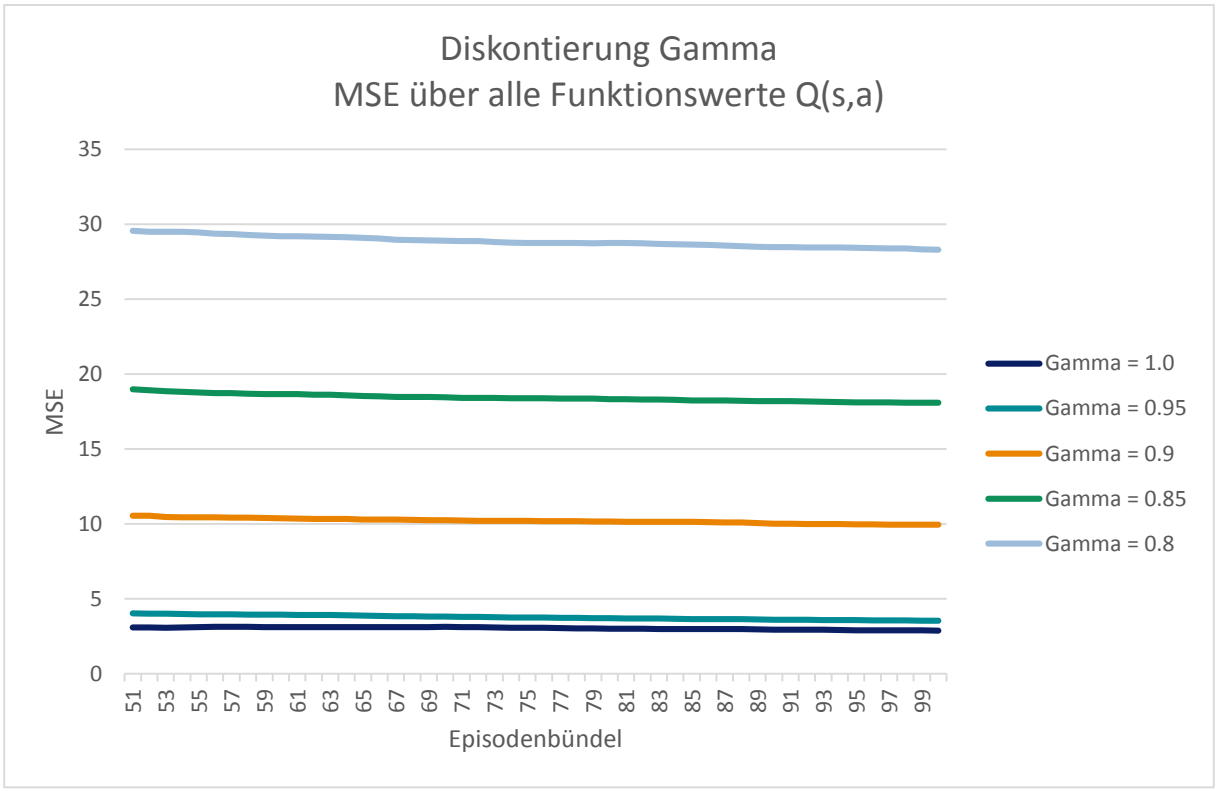
Diskontierung γ



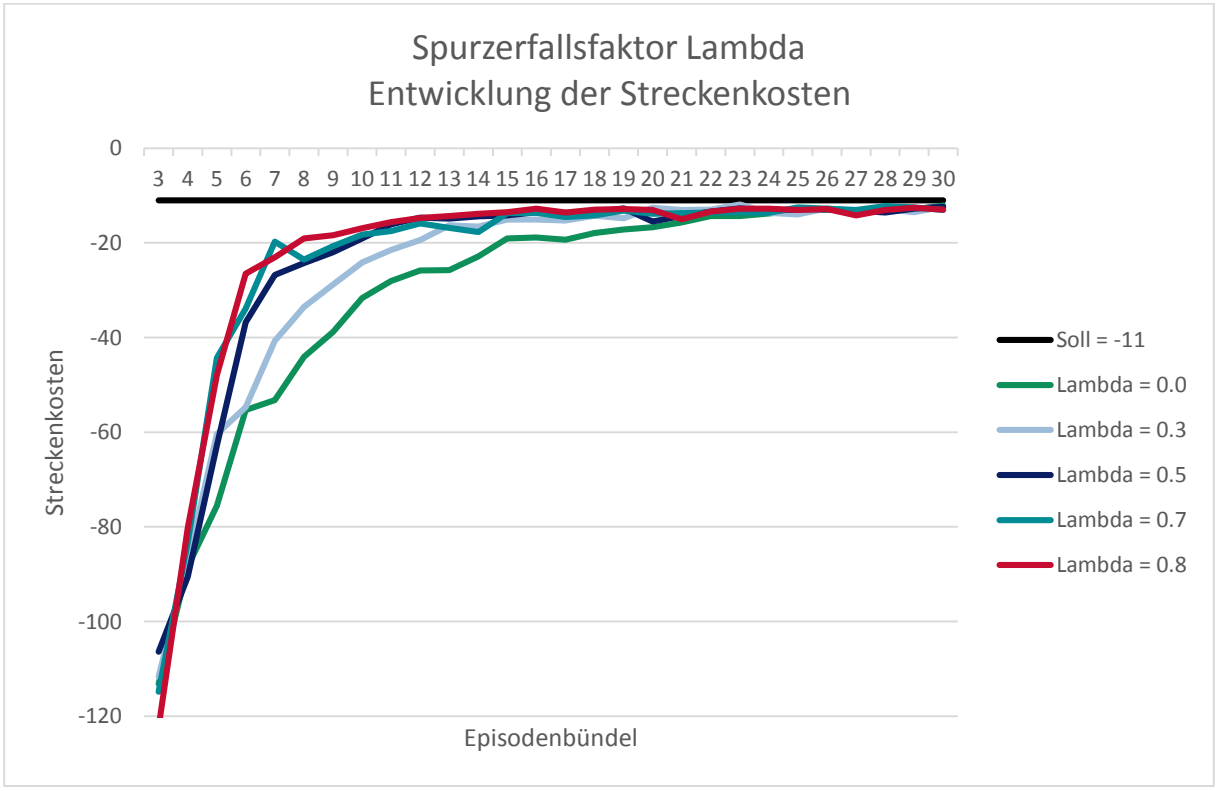

Diskontierung Gamma
Entwicklung der Streckenkosten
Streckenkosten
Episodenbündel
Soll = -11
Gamma = 1.0
Gamma = 0.95
Gamma = 0.9
Gamma = 0.85
Gamma = 0.8

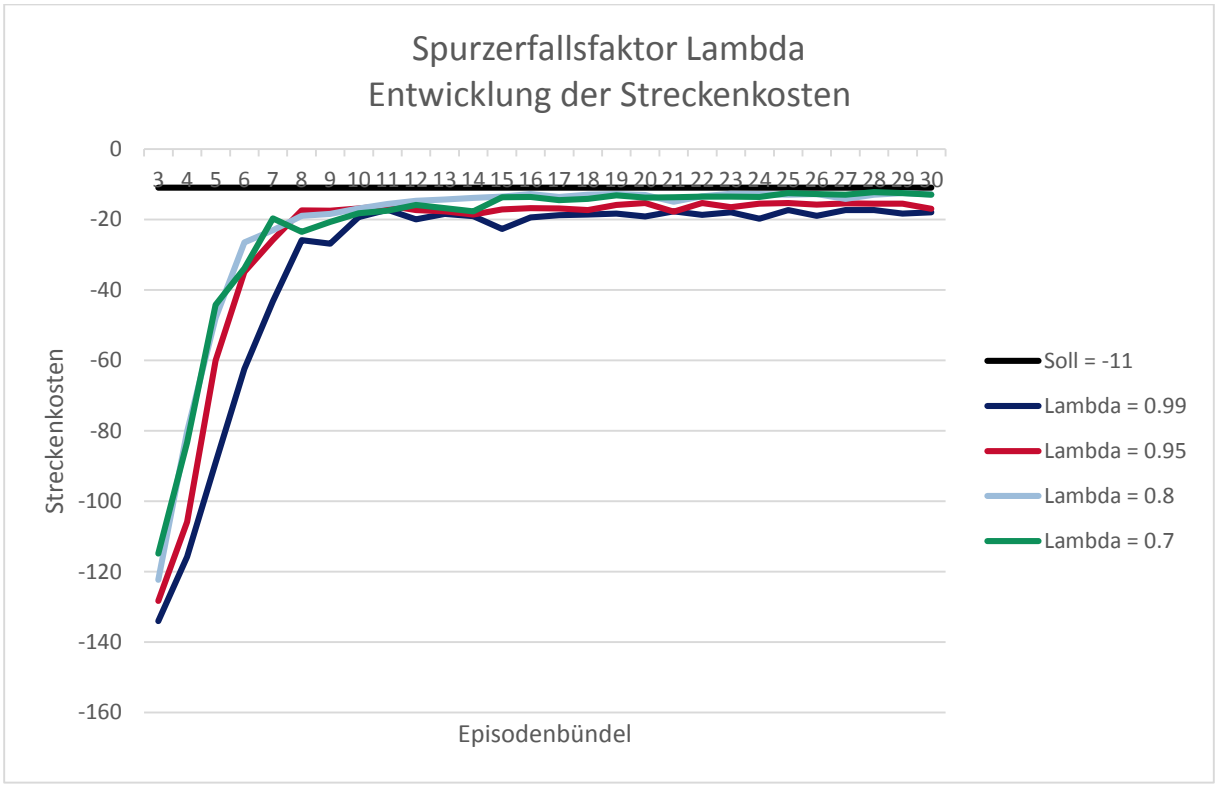

Diskontierung Gamma
MSE über alle Funktionswerte Q(s,a)
MSE
Episodenbündel
Gamma = 1.0
Gamma = 0.95
Gamma = 0.9
Gamma = 0.85
Gamma = 0.8

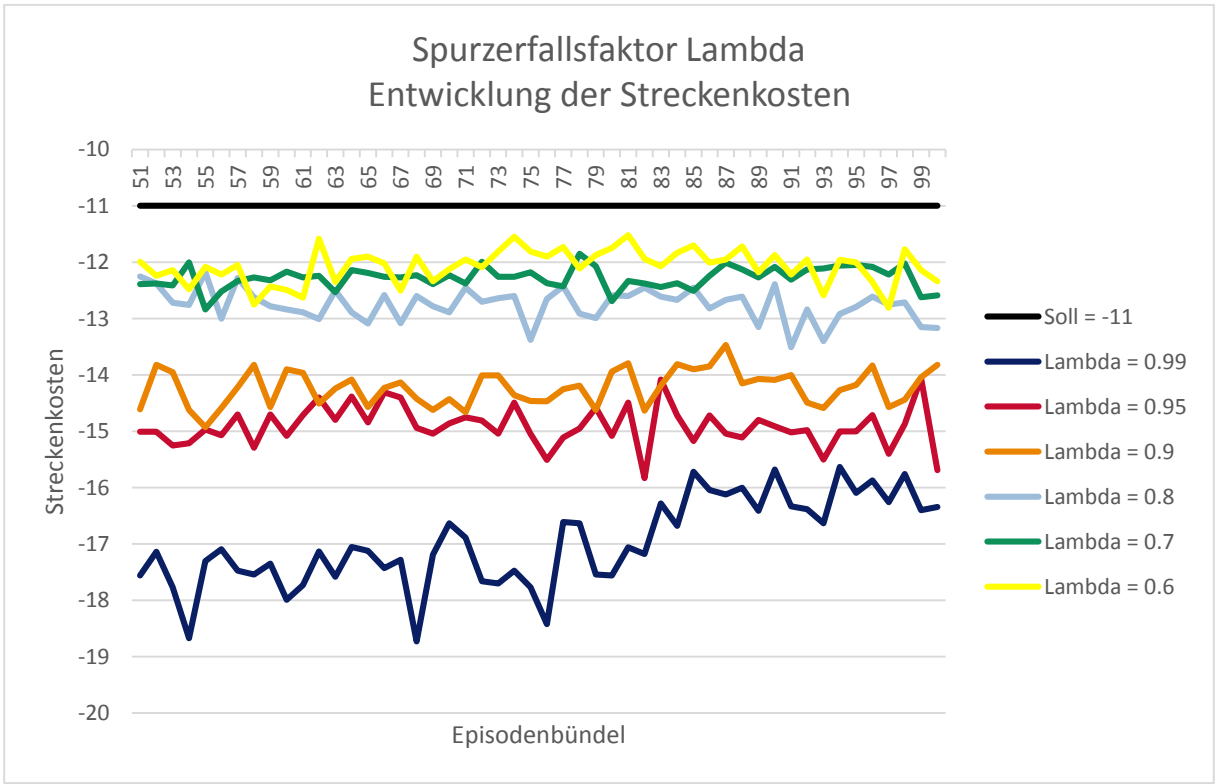
xx

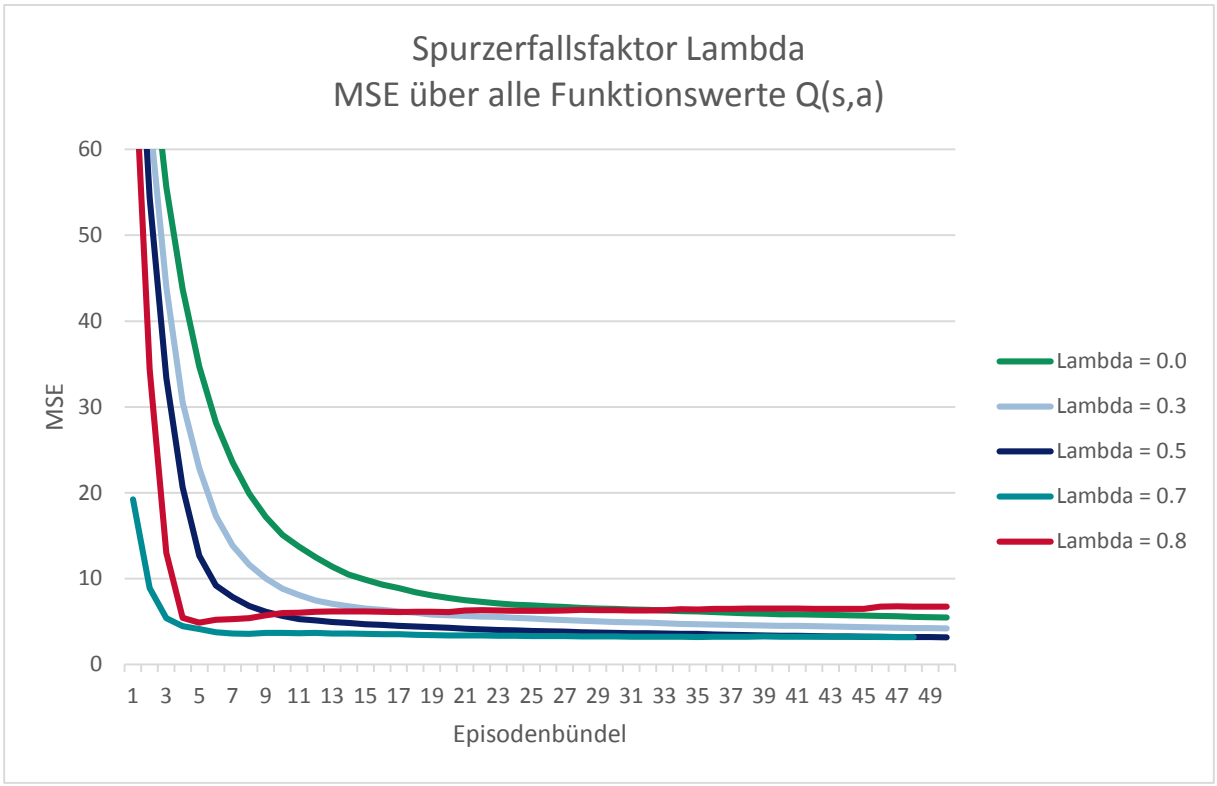


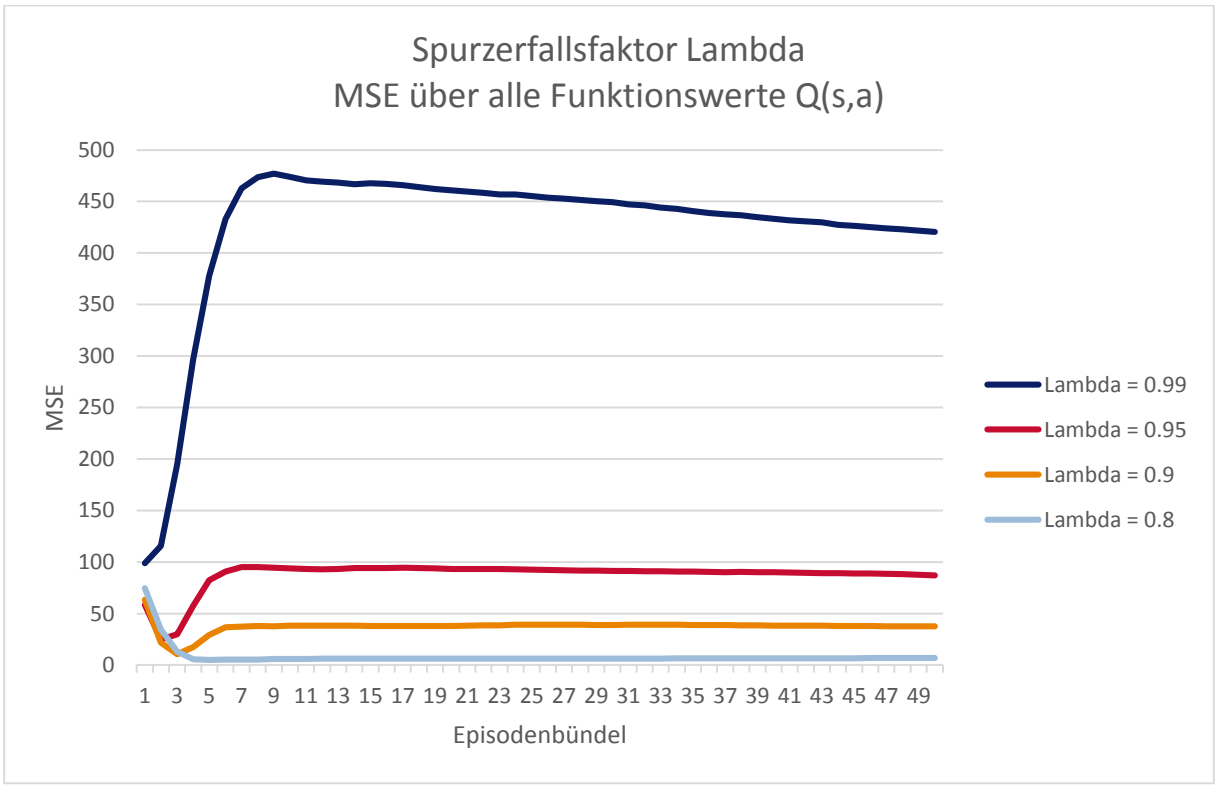
Spurzerfall λ



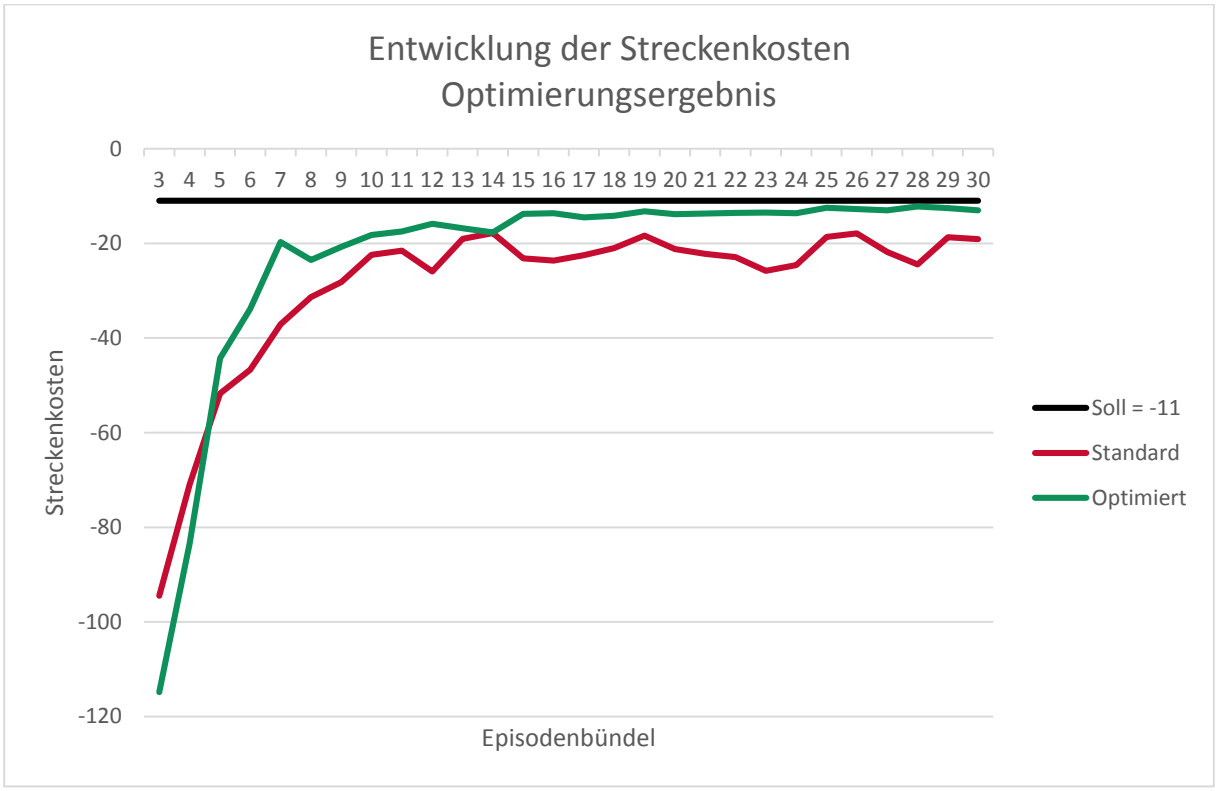

Spurzerfallsfaktor Lambda
Entwicklung der Streckenkosten
Streckenkosten
Episodenbündel
Soll = -11
Lambda = 0.99
Lambda = 0.95
Lambda = 0.8
Lambda = 0.7

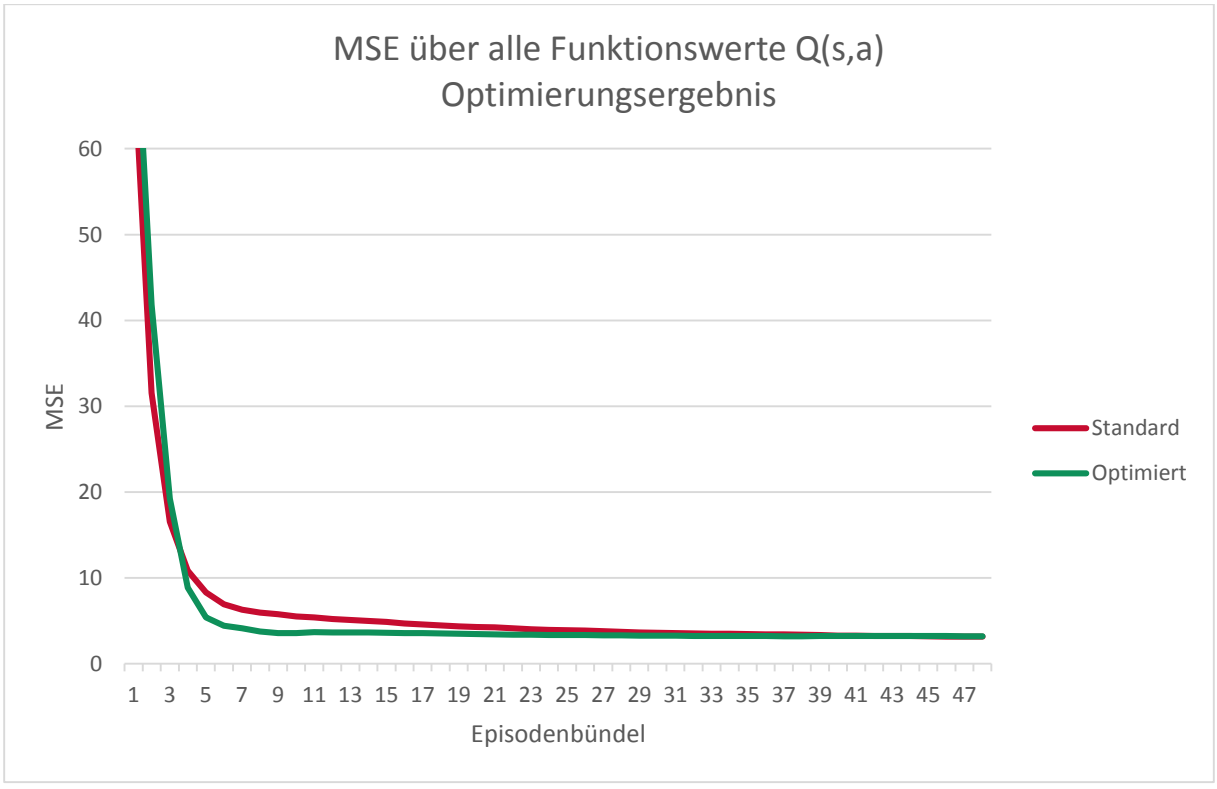

Spurzerfallsfaktor Lambda
Entwicklung der Streckenkosten
Streckenkosten
Episodenbündel
Soll = -11
Lambda = 0.99
Lambda = 0.95
Lambda = 0.9
Lambda = 0.8
Lambda = 0.7
Lambda = 0.6


Spurzerfallsfaktor Lambda
MSE über alle Funktionswerte Q(s,a)
MSE
Episodenbündel
Lambda = 0.0
Lambda = 0.3
Lambda = 0.5
Lambda = 0.7
Lambda = 0.8

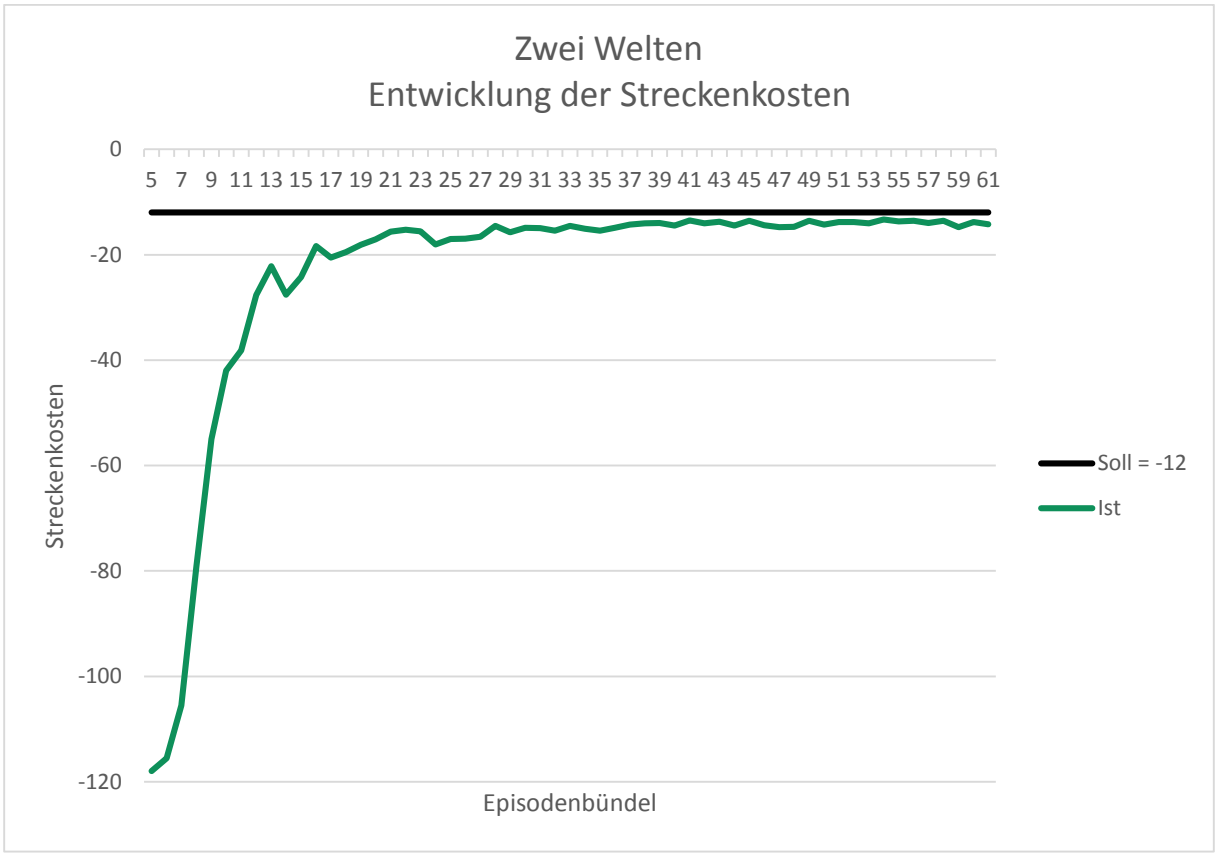

Spurzerfallsfaktor Lambda
MSE über alle Funktionswerte Q(s,a)
MSE
Episodenbündel
Lambda = 0.99
Lambda = 0.95
Lambda = 0.9
Lambda = 0.8

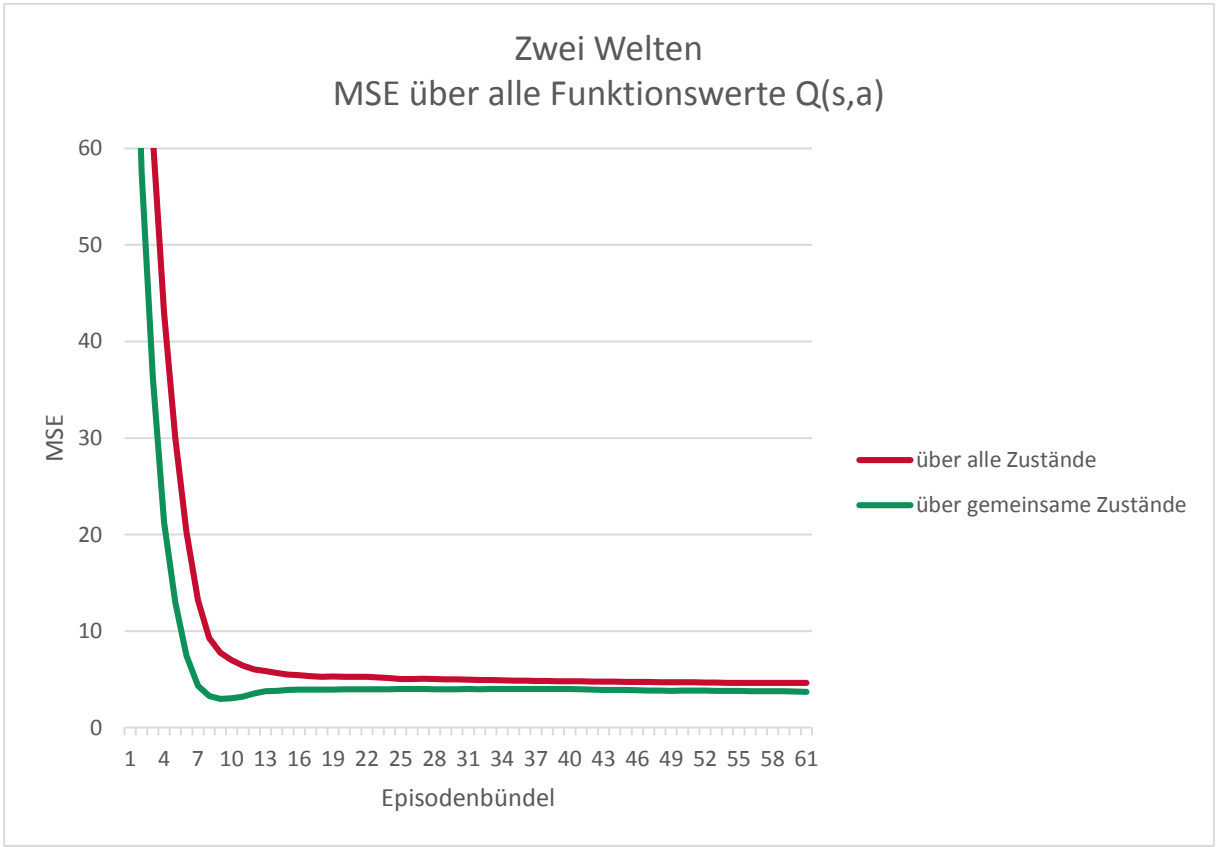
Optimierungsergebnis





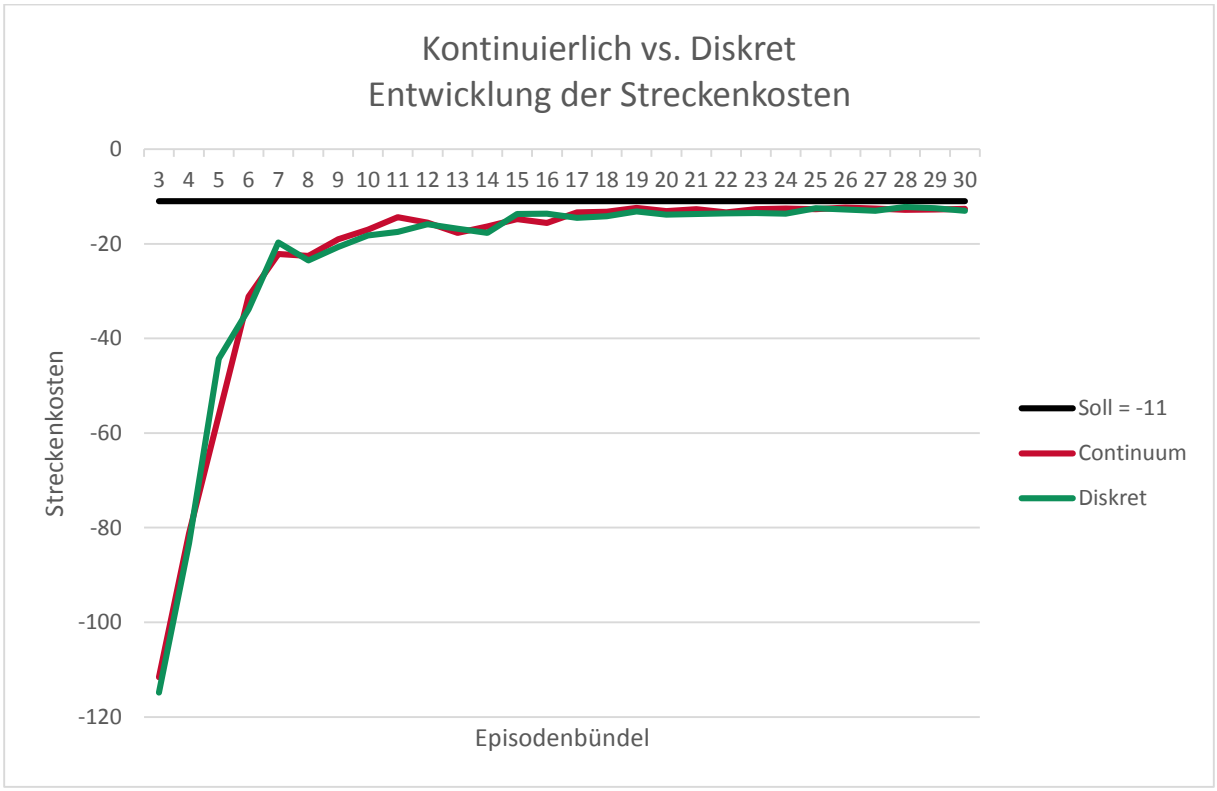
B.2 Versuchsergebnis Äquicausa Problem

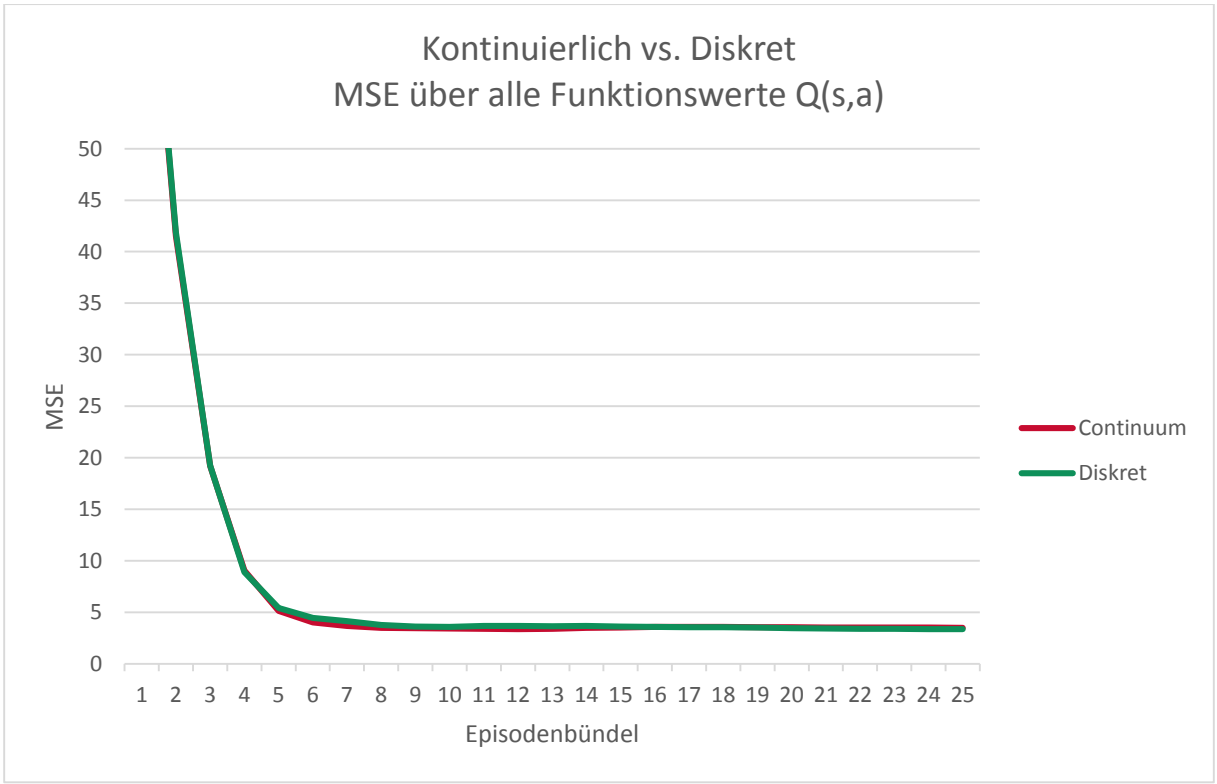




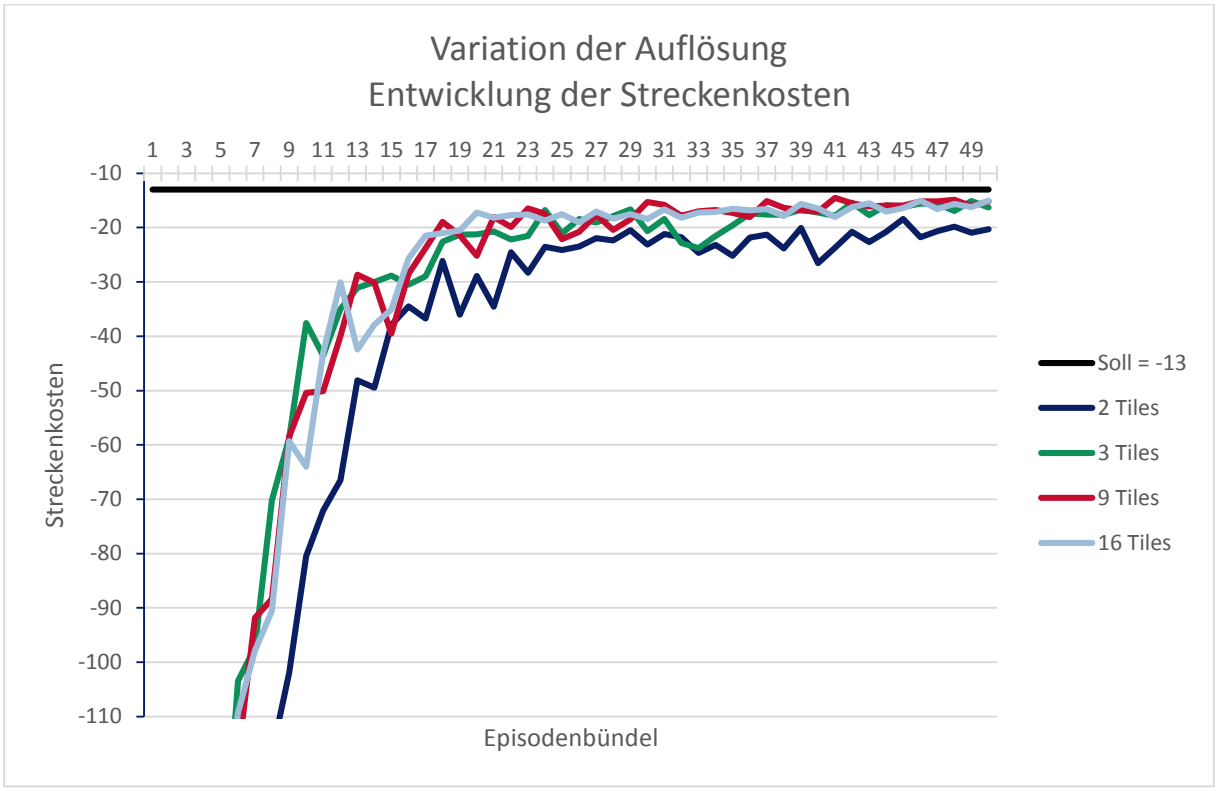
B.3 Versuchsergebnis Tiling

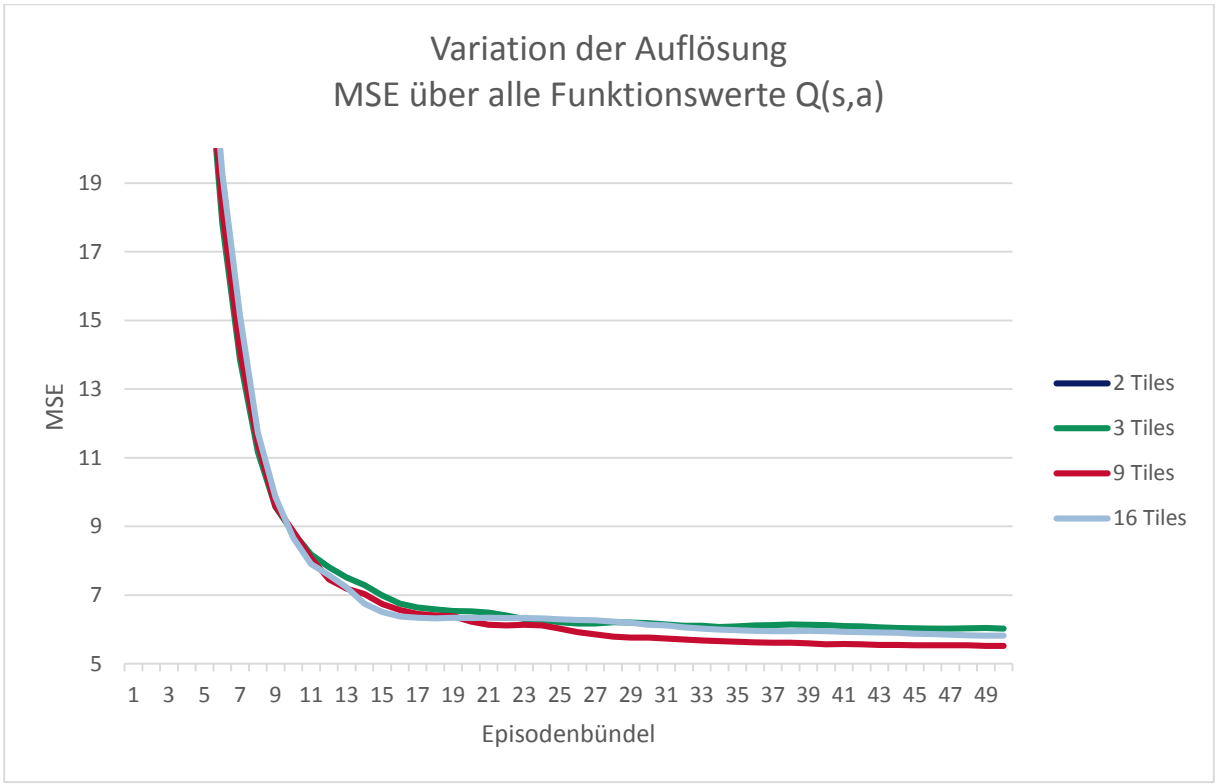
1 Tiling



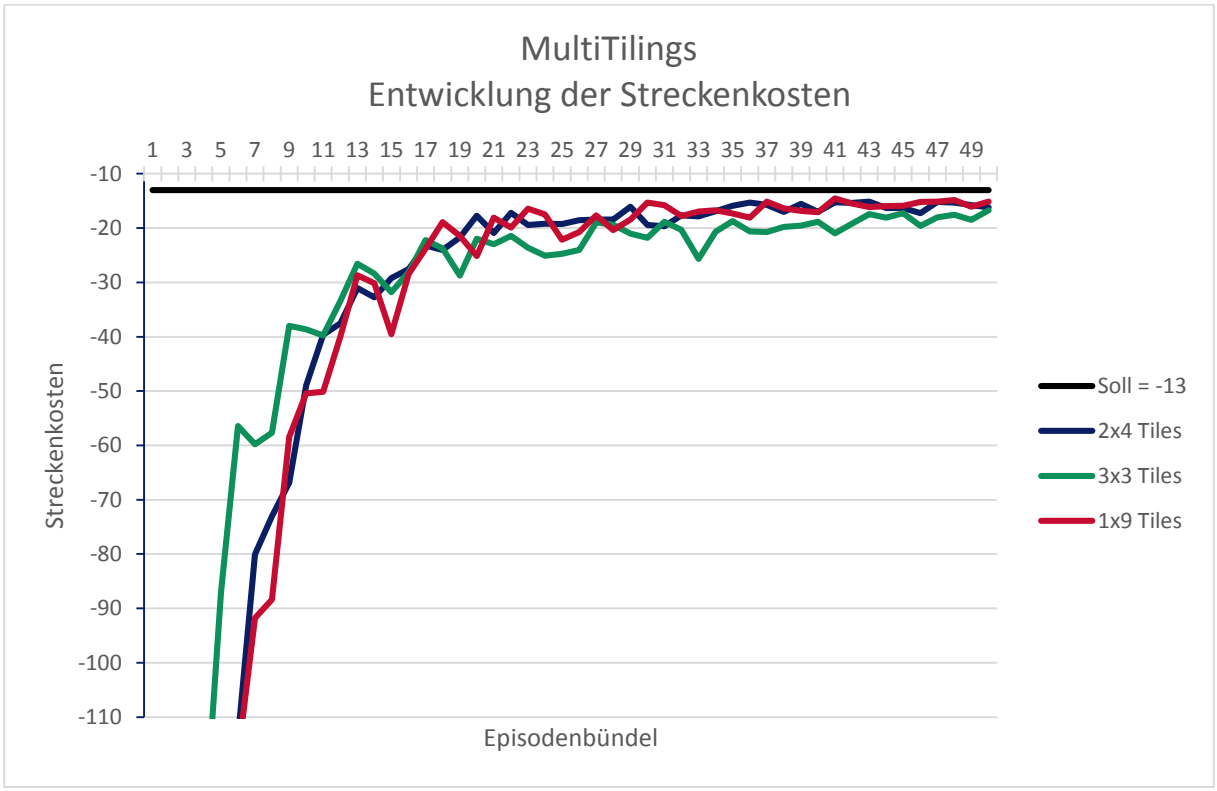


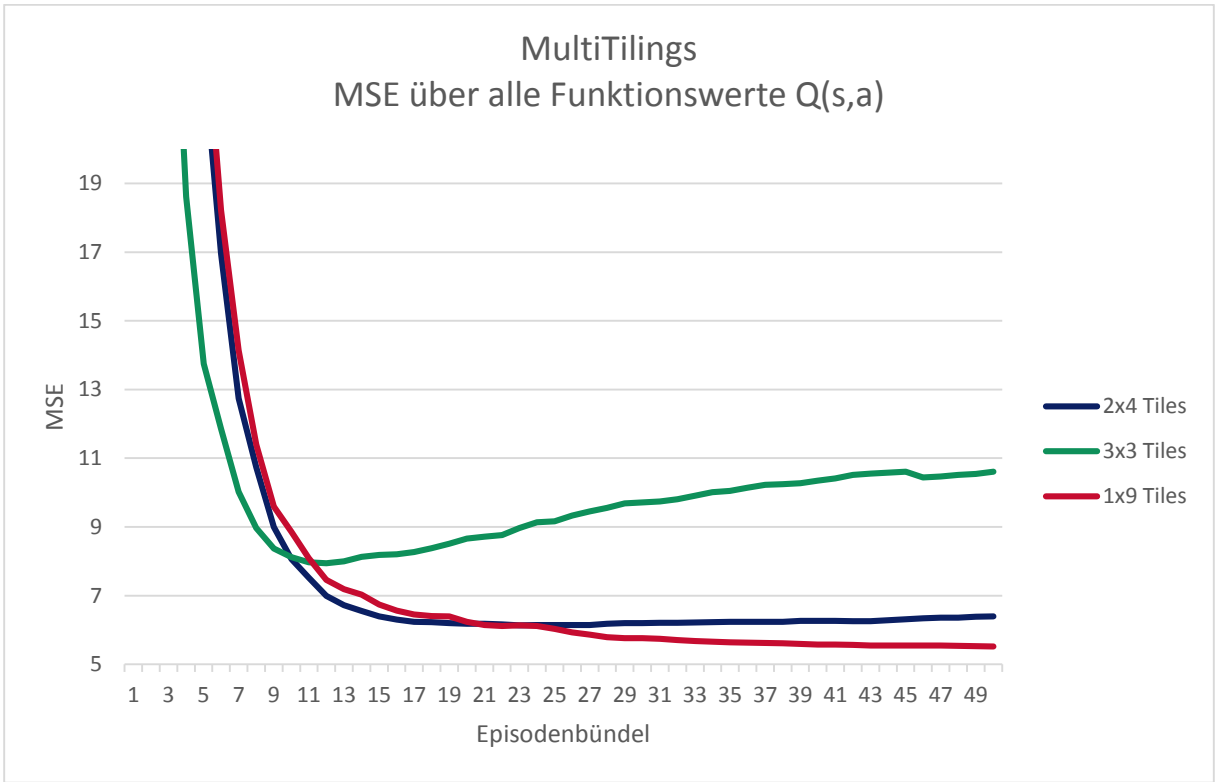
Variierende Auflösung





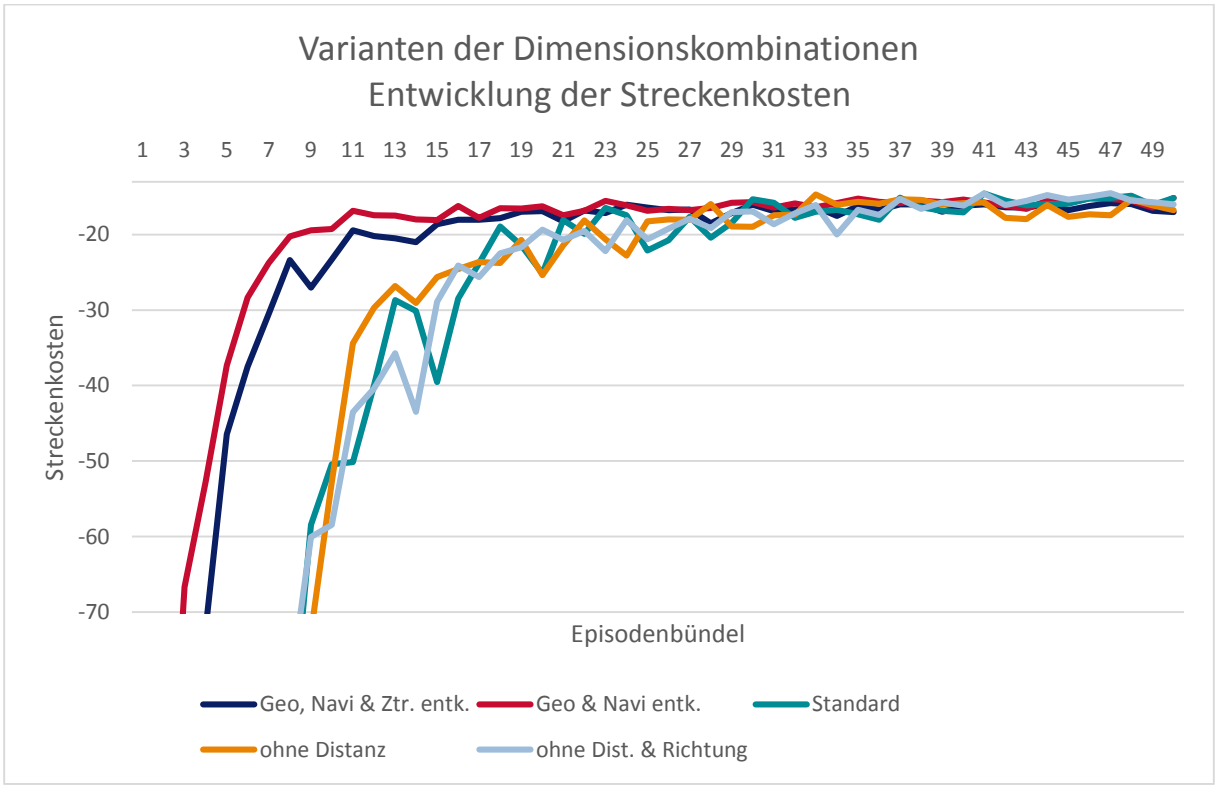
MultiTiling

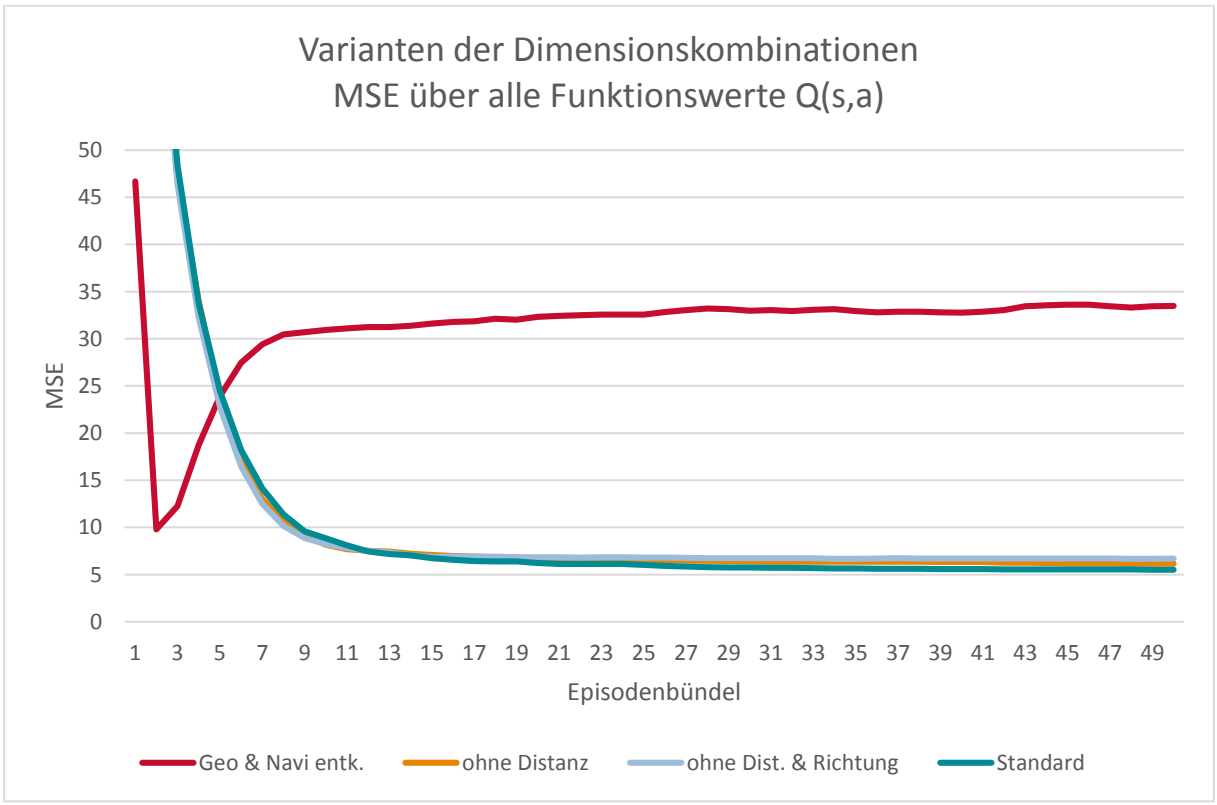


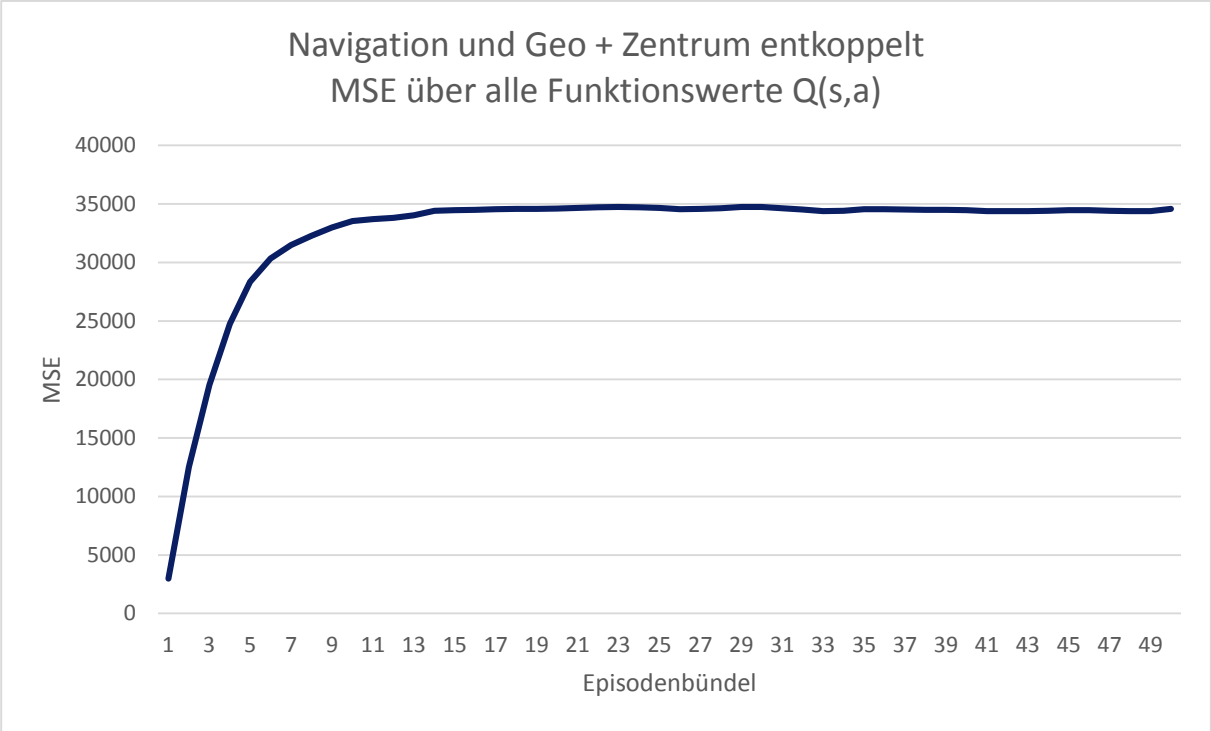


B.4 Versuchsergebnisse Merkmalskombinationen

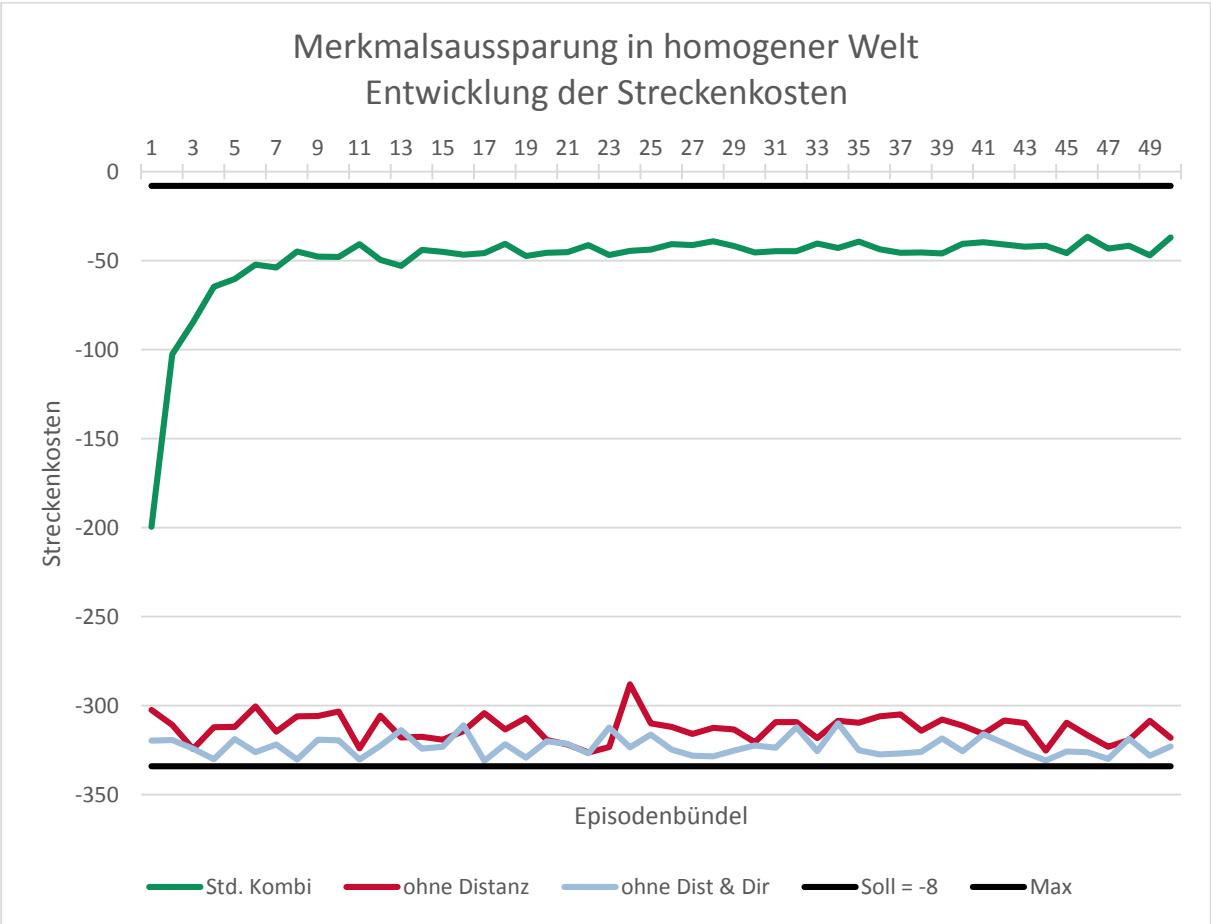
Kodierungsvarianten



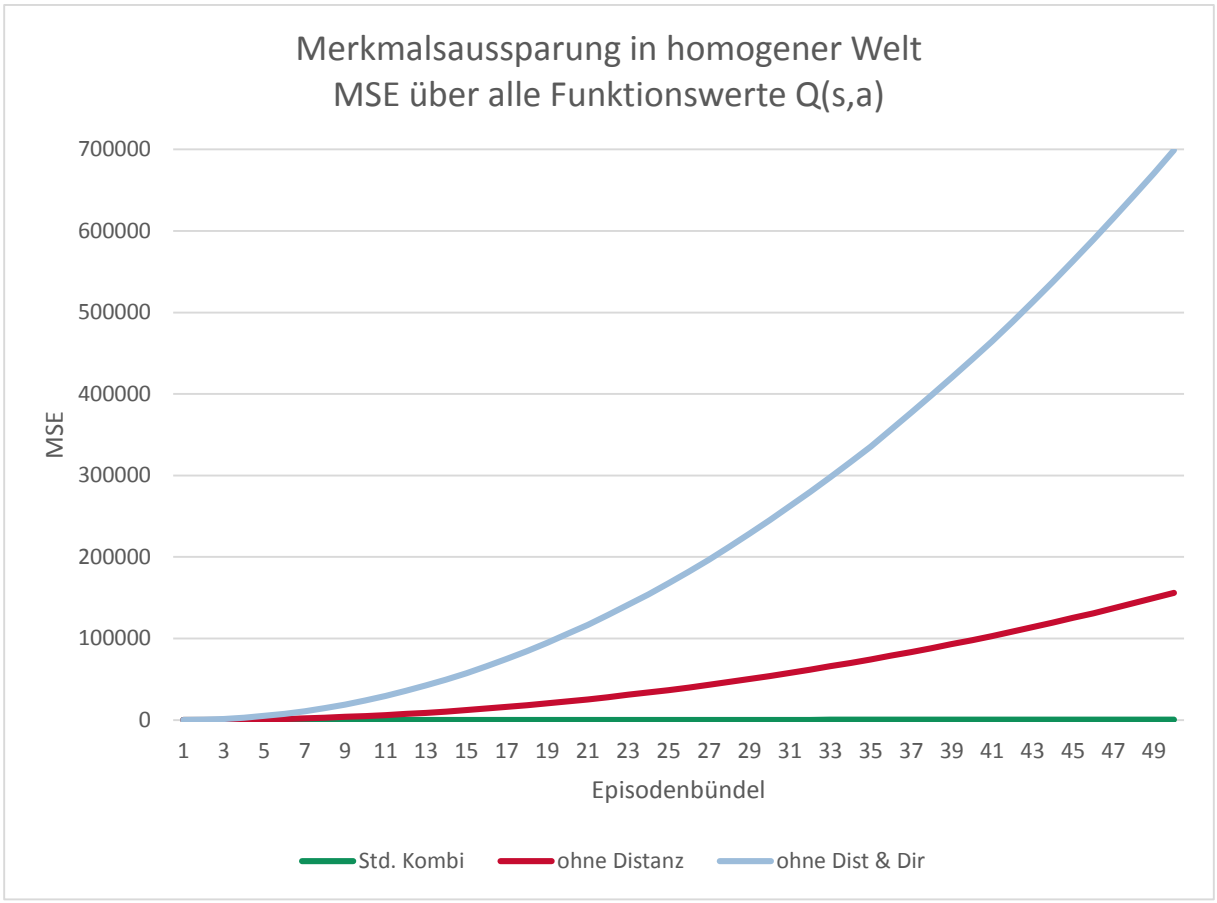


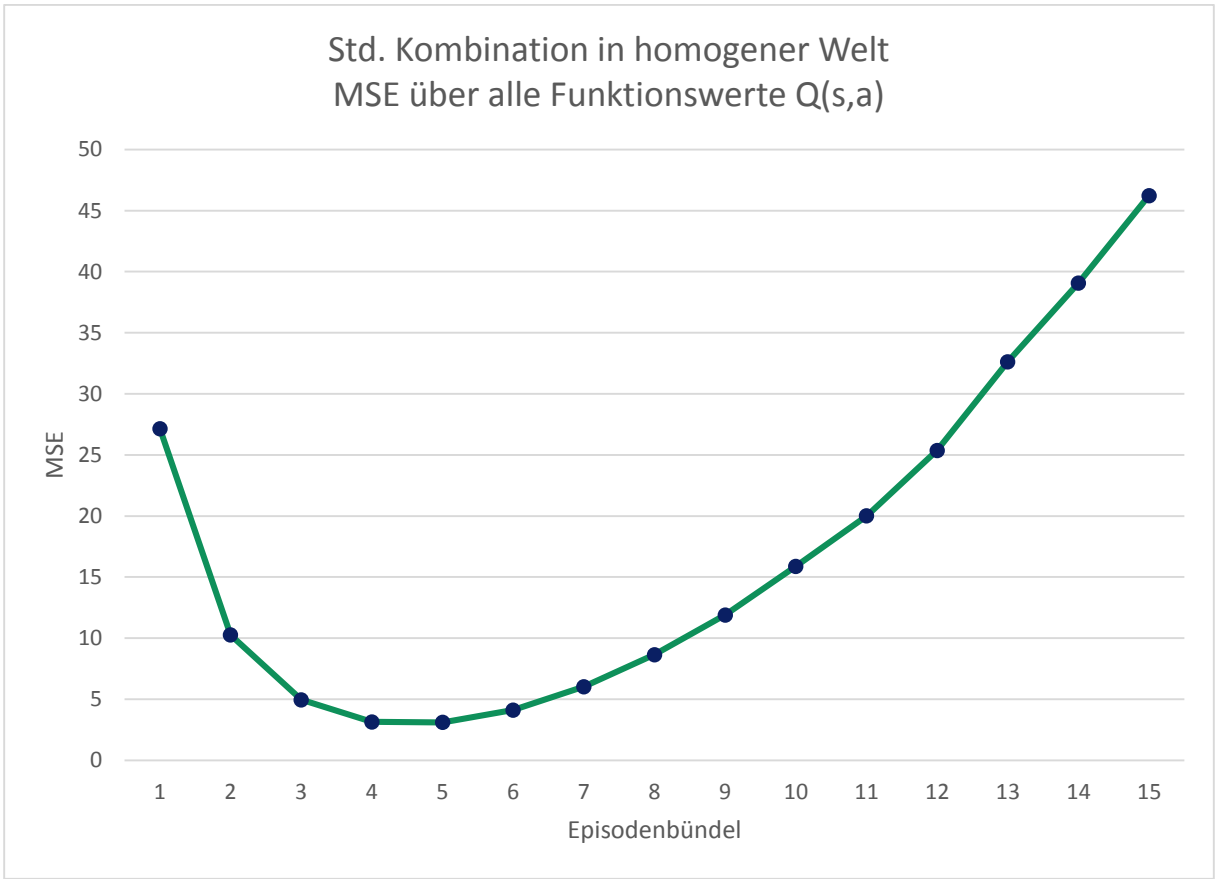


Homogene Welt

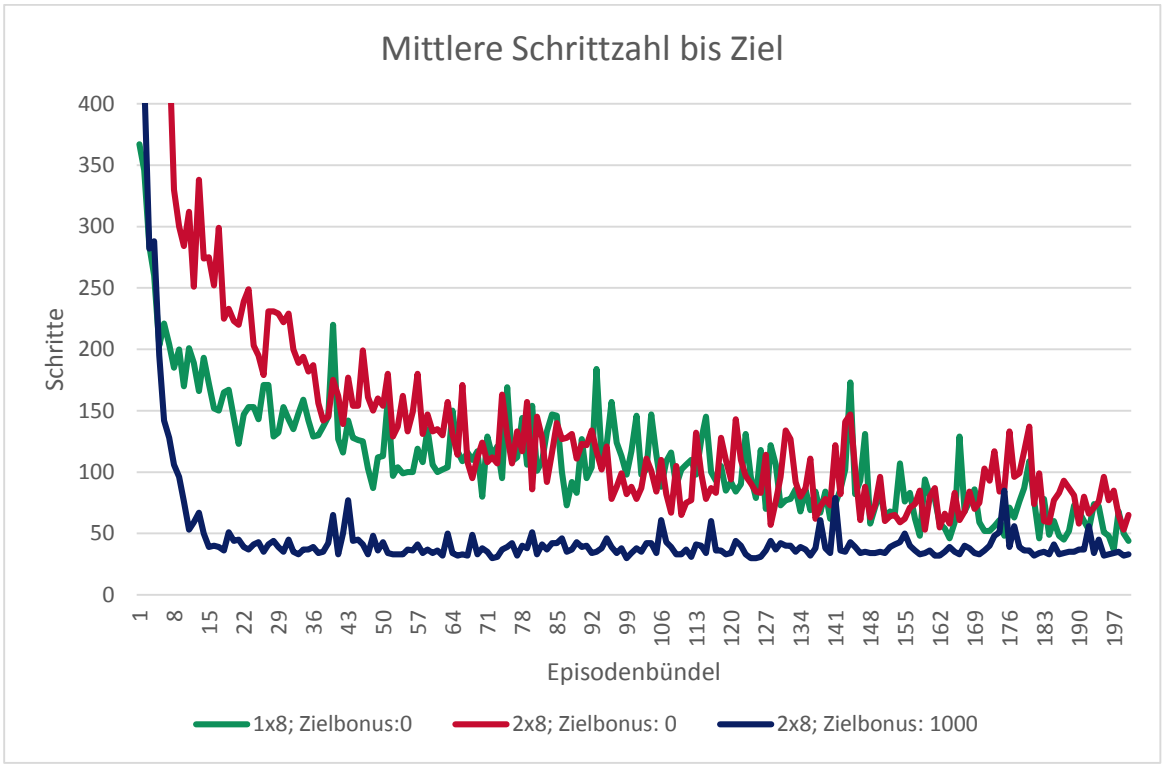


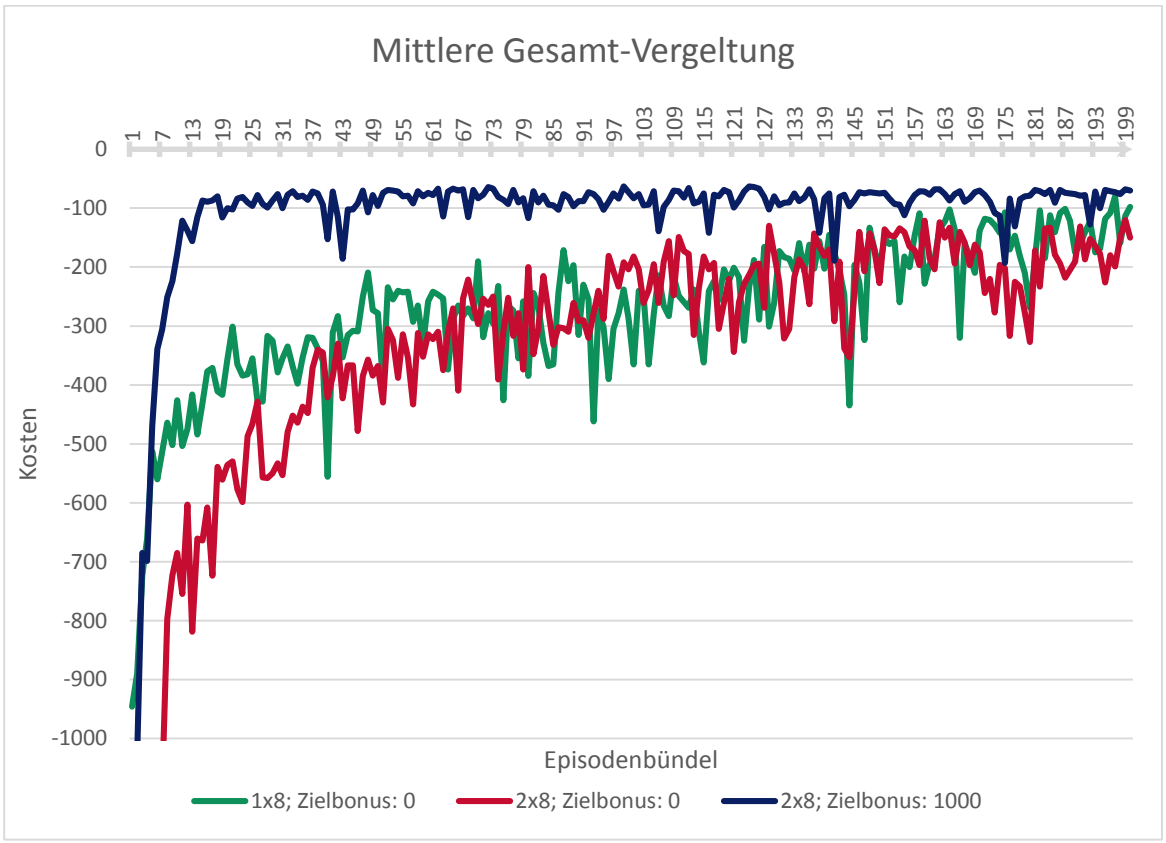
xxx


Merkmalsaussparung in homogener Welt
MSE über alle Funktionswerte Q(s,a)
MSE
700000
600000
500000
400000
300000
200000
100000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Episodenbündel
Std. Kombi
ohne Distanz
ohne Dist & Dir


Std. Kombination in homogener Welt
MSE über alle Funktionswerte Q(s,a)
MSE
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Episodenbündel

B.5 Versuchsergebnis AMEE-Szeanrio





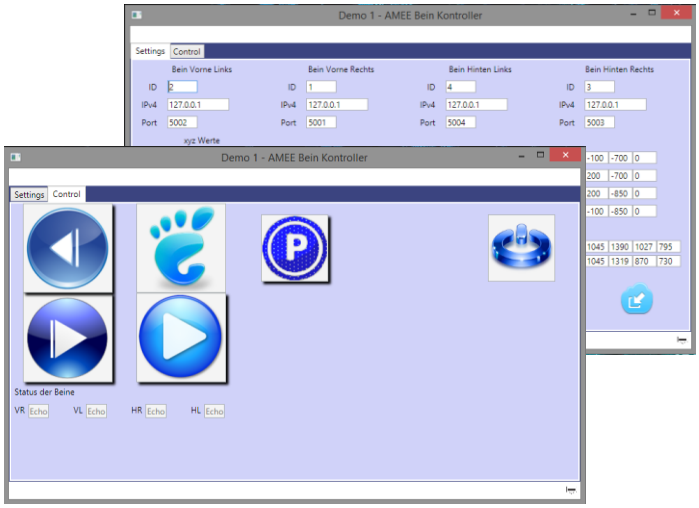
Anhang C Impressionen

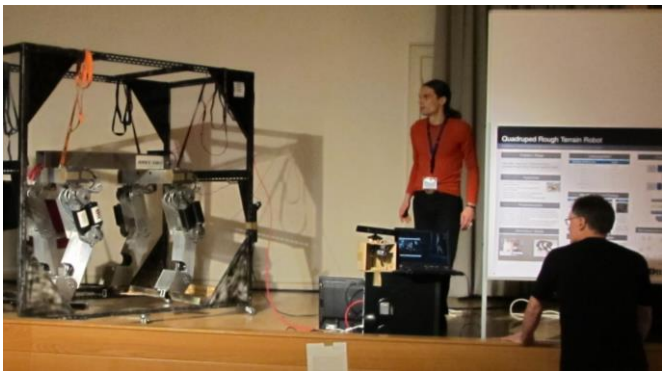
Erstes Design-Konzept von Ruhnke 2010

Fertigung Jan – Apr 2012

Zu 2012, über ein Jahr nach Beginn des Masterstudiums wurden uns von der Hochschule 8.800 Euro Projektgelder bewilligt. Aus einer theoretischen Betrachtung wurde eine praktische Arbeit – die viel Zeit in Anspruch nahm aber für einen Informatiker so ungewöhnlich wie bereichernd war.

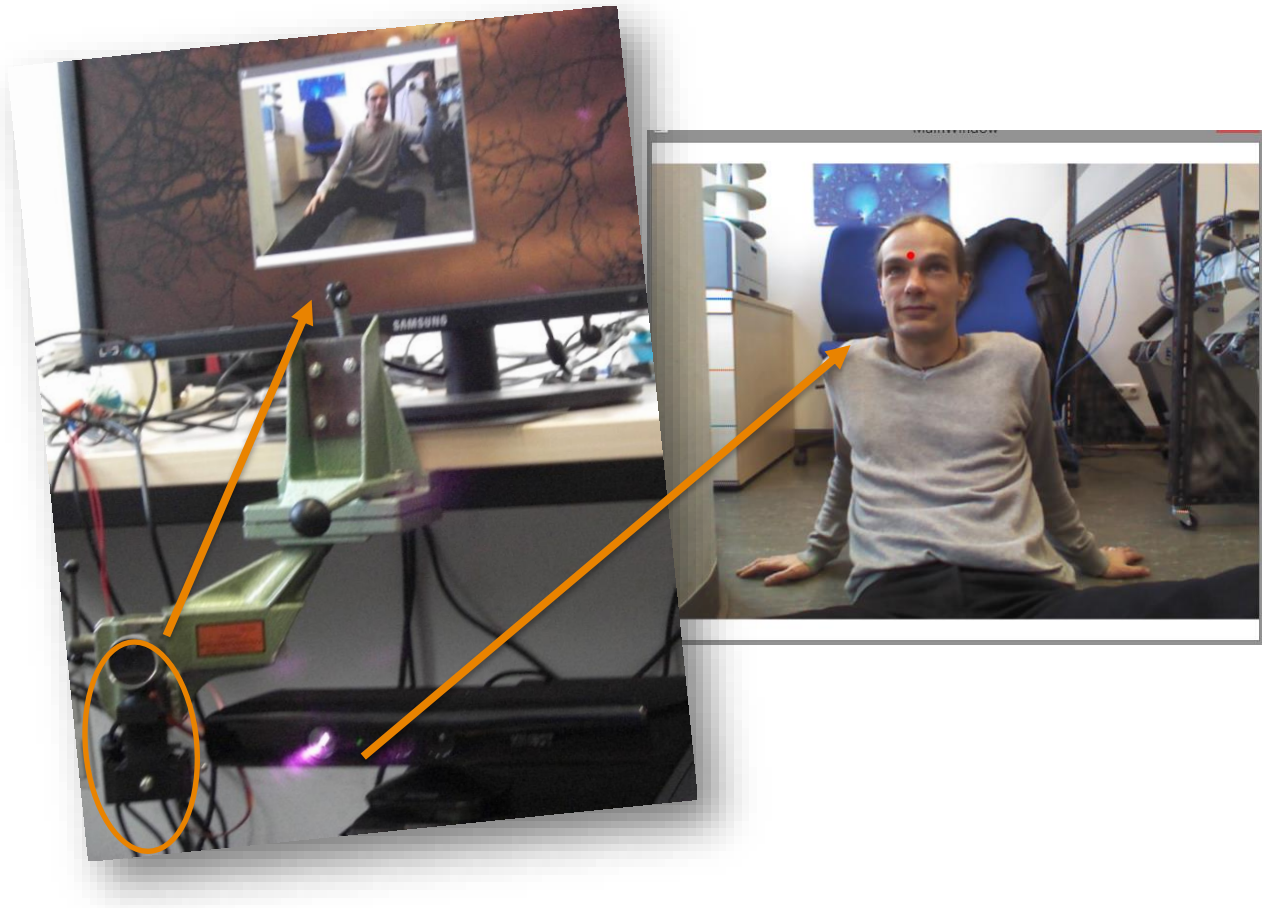
Nacht des Wissens - Nov. 2013, HAW Hamburg





Der Hauptkontroller war noch nicht fertig und die Beinkontroller waren gerade eben einem neuen MCU-Typ angepasst. Damit die Besucher „was zu sehen“ hatten, wurde eine rudimentäre Befehlssteuerung und Oberfläche entwickelt, die den Roboter einen Schrittzyklus ausführen oder die Parkposition (hinlegen) ausführen lässt. AMEE blieb dabei an einer eigens gebauten Traverse hängen.

Ebenfalls für die Nacht des Wissens wurde eine „Ich schau Dir in die Augen“ Demo entwickelt. Mittels der Kinect® wurde ein beliebiges Gesicht erfasst. Der Pan-Tilt-Servo richtete dann die daran montierte Kamera auf den Besucher aus. Für Schmunzeln sorgte die Kennzeichnung des anvisierten Punktes durch einen roten Punkt – zwischen den Augen…

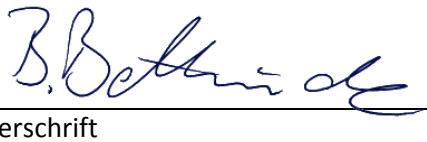


Versicherung über Selbstständigkeit

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, 10.10.2014

Ort, Datum



Unterschrift