Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Sensomotorische Kartografierung und Planung unter Verwendung neuronaler Netze und homöokinetischer Exploration

Diplomarbeit
zur Erlangung des akademischen Grades Diplom-Informatiker

Berlin, den 31. März 2010

eingereicht von: Dipl.-Ing. (BA) Matthias Markl
Gutachter: Prof. Dr. Hans-Dieter Burkhard
Prof. Dr. Bernd-Holger Schlingloff
Betreuer: Dr. Manfred Hild

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 31. März 2010

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 31. März 2010

Kurzfassung

In der vorliegenden Arbeit wird ein Lernalgorithmus zur Exploration des sensomotorischen Raumes eines humanoiden Roboters mit der Möglichkeit einer anschließenden Planung beschrieben. Wichtige Eigenschaften dieses Algorithmus' sind das lebenslange Lernen, der niedrige Energieverbrauch und der Selbstschutz des Roboters.

Da der Lernalgorithmus auf künstlichen neuronalen Netzen basiert, wird zunächst dieses Themengebiet erläutert. Basierend auf diesen Kenntnissen werden vier Algorithmen aus der Literatur und der universitären Arbeitsgruppe vorgestellt. Nach einer Analyse der Unterschiede und Gemeinsamkeiten wird auf Basis der vorhandenen Algorithmen ein neues Lernverfahren vorgestellt. Die Exploration des Lernverfahrens wird durch eine homöokinetische Regelung umgesetzt. Diese Regelung führt zu einem ständigen Antrieb zur Exploration und wird um einen Mechanismus zum Selbstschutz des Roboters erweitert. Die Erkenntnisse der Exploration werden repräsentativ in künstlichen neuronalen Netzen gespeichert. Auf diesen Netzen ist über einen dritten Teilalgorithmus anschließend eine Planung mit unterschiedlichen Kriterien, wie der Energie- oder Zeit-Effizienz möglich. Nach einer kompakten Beschreibung der objektorientierten Implementierung des Algorithmus', wird dieser umfangreich in der Simulation und an einem humanoiden Roboter getestet. Die Experimente umfassen die Analyse der explorierten künstlichen neuronalen Netze auf Basis der homöokinetischen Regelung, die Wirksamkeit des Selbstschutzes und die Auswirkungen unterschiedlicher Schwerpunkte in der Planung unter Verwendung der künstlichen neuronalen Netze.

Die durchgeführten Experimente zeigen, dass der entwickelte Algorithmus für die Exploration des sensomotorischen Raumes geeignet ist. Weiterhin ist lebenslanges Lernen möglich, der Selbstschutz wurde erfolgreich integriert und es kann eine Planung nach unterschiedlichen Kriterien wirksam durchgeführt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

> *Denn indem ein Mensch mit den ihm von Natur gegebenen Gaben sich zu verwirklichen sucht, tut er das Höchste und einzig Sinnvolle, was er kann.*
>
> Hermann Karl Hesse (*1877, †1962), deutscher Dichter

Während der Entwicklung in seinen ersten Monaten und Jahren handelt der Mensch wie in obigem Zitat beschrieben. Aus einem Zellverband entstehen alle Organe und Körperteile, es bilden sich Sinnesorgane, um die Umwelt zu erfahren, und Muskeln, um sich zu bewegen. Alle Sinneswahrnehmungen werden über Nervenzellen weitergereicht und verarbeitet. Nach der Geburt beginnt der Mensch seine Umwelt zunächst ungerichtet zu erforschen. Aus diesen Erfahrungen lernt er beispielsweise seine Laute zu Wörtern zu formen oder seine Muskeln zum Krabbeln oder Aufstehen zu verwenden.

Um diese Analogie auf einen humanoiden Roboter zu übertragen, wird das Heranwachsen ausgeblendet. Basis ist ein fertiger Roboter, der seine Umwelt durch verschiedene Sensoren als Äquivalent zu den Sinnesorganen zu erfassen vermag. Weiterhin ist eine primäre Vorverarbeitung der Sensordaten, wie sie auch beim Menschen stattfindet, möglich. Ein Beispiel wäre das Scharfstellen der Linse einer Kamera. Die Muskeln werden durch Motoren repräsentiert, die eine aktive Bewegung aller Gelenke ermöglichen.

Sowohl die Vorverarbeitung als auch das eigentliche Denken finden beim Menschen über elektrische Impulse und Botenstoffe in Verbänden aus Nervenzellen statt. In der Informatik wurden hierfür die künstlichen neuronalen Netzwerke entwickelt. Zwar können noch nicht alle Aspekte des biologischen Vorbildes realisiert werden, doch können auch mit einfacheren Modellen bereits interessante Parallelen gefunden werden. Ein Beispiel sind die sogenannten „Place Cells“[Haf05]. Experimente an Ratten zeigen, dass es Nervenzellen gibt, die eine hohe Aktivität haben, wenn sich das Tier an einem bestimmten geografischen Ort befindet. Über diese Zellen kann dieses Tier navigieren und wieder nach Hause finden. Das Beispiel zeigt, dass die Fähigkeit das erlernte Wissen zur Planung von Handlungen zu nutzen mindestens genauso wichtig ist wie das Lernen selbst.

Diese Arbeit befasst sich sowohl mit dem Aneignen von Wissen als auch mit der Verwendung von Wissen. Dieses Wissen bezieht sich in erster Linie auf das Kennenlernen des

eigenen Körpers. Während einer Explorationsphase werden sensorische Aufnahmen der Umwelt mit den eigenen Handlungen verknüpft und verarbeitet. Die dadurch erhaltenen Informationen werden in einem wachsenden künstlichen neuronalen Netz abgespeichert. Während einer Planungsphase können bereits explorierte Bereiche angesteuert werden, wobei das erlernte Wissen aus dem Netz verwendet wird.

Ziel dieser Arbeit ist es einen Algorithmus zur Exploration und Planung im sensomotorischen Raum auf Basis künstlicher neuronaler Netze zu entwickeln. Da der Algorithmus in einer Simulationsumgebung und an einem humanoiden Roboter getestet wird, soll während der Exploration der Selbstschutz des Roboters beachtet und in den Algorithmus integriert werden. Weiterhin verfügt der Roboter über Sensoren zur Messung des Energieverbrauches. Deswegen soll der Algorithmus die Möglichkeit bieten eine Planung nach unterschiedlichen Kriterien durchzuführen. Diese Kriterien sind beispielsweise die Energie- oder Zeit-Effizienz. Ein weiteres Ziel ist das lebenslange Lernen.

1.1 Aufbau der Arbeit

Dem vorliegenden Abschnitt folgt eine Einordnung der Arbeit in den aktuellen Stand der Forschung und eine Einführung in die humanoiden Roboter der M-Serie, die das einleitende Kapitel abschließt. Dort wird der Aufbau mit den vorhandenen Sensoren und Motoren beschrieben.

In Kapitel 2 werden die Grundlagen zu künstlichen neuronalen Netzen erläutert. Nach einer Einführung in dynamische Systeme und deren Eigenschaften werden die verwendeten Neuronenmodelle vorgestellt. Anschließend werden neuronale Netze und Felder beschrieben. Im letzten Abschnitt werden verschiedene Lernverfahren für künstliche neuronale Netze dargestellt. In diesem Kapitel wird nicht auf spätere Inhalte vorgegriffen, so dass es bei Kenntnis der vorgestellten Themen übersprungen werden kann.

Das anschließende Kapitel 3 betrachtet Algorithmen zur Exploration des sensomotorischen Raumes. Es werden vier unterschiedliche Algorithmen vorgestellt und auf bestimmte Eigenschaften hin analysiert. Diese Eigenschaften sind beispielsweise die Möglichkeit der neuronalen Implementierung oder des lebenslangen Lernens. Nach dieser Analyse werden Gemeinsamkeiten und mögliche Kombinationen aufgezeigt. Abschließend wird ein neuer Algorithmus entwickelt. Dieser kombiniert die Vorteile zweier bereits vorgestellter Algorithmen und wird an passender Stelle erweitert, um die Ziele dieser Arbeit zu erreichen.

Die Software-Architektur und Implementierung des entwickelten Algorithmus' ist Thema in Kapitel 4. Neben der objektorientierten Architektur wird ebenfalls der Programmfluss der Software beschrieben. Im Abschnitt der Implementierung wird auf Details der Programmierung eingegangen.

In Kapitel 5 werden die Experimente in der Simulation und an einem Roboter der M-Serie beschrieben. Nach einführenden Tests zur Repräsentativität der neuronalen Netze und der Parameterwahl werden die Ergebnisse der Aktionsauswahl des neuen Al-

gorithmus' untersucht. Anschließend werden die Maßnahmen zum Selbstschutz getestet. In einem letzten Experiment werden die entwickelten Planungsmethoden analysiert.

Zuletzt werden in Kapitel 6 ein Fazit zur gesamten Arbeit gezogen und die Errungenschaften und Probleme kritisch reflektiert. Abschließend werden Möglichkeiten zur Erweiterung und Anpassung aufgeführt.

1.2 Einordnung der Arbeit

Die Arbeit entstand im Labor für Neurorobotik des Instituts für Informatik an der Humboldt-Universität zu Berlin. Vorbetrachtungen zur sensomotorischen Exploration wurden in [Hea08] durchgeführt. In der Forschungsgruppe wird parallel ein weiterer Explorationsalgorithmus unter Verwendung von intrinsischer Motivation [OK07] entwickelt. Hier wird die Exploration durch eine interne Belohnung für neu erlerntes Wissen vorangetrieben. Auf die grundlegenden Eigenschaften dieser Methode wird in Abschnitt 3.3 eingegangen.

Eine andere Möglichkeit eine autonome Exploration durchzuführen ist die Homöokinese. Das Prinzip einer gleichmäßigen Exploration, die einen konstanten kinetischen Zustand erhält, wird in [Der99] vorgestellt und in dieser Arbeit in Abschnitt 3.4 erklärt. Neuere Veröffentlichungen zeigen, dass die Ergebnisse der Exploration auch in Experten abgespeichert werden können [MFH08]. Diese Experten stellen dann ein bestimmtes Verhalten bereit, jedoch ist mit diesem keine konkrete Planung möglich. Der Algorithmus wurde bisher auf unterschiedlichen Systemen in der Simulation und der Realität getestet. Diese Systeme besitzen, bis auf einen menschlichen Arm und eine menschliche Hand, meist eine einfache Form und verfügen über wenige Freiheitsgrade. Eine Anpassung an die Roboter der M-Serie wird in Abschnitt 3.6.1 beschrieben.

Basierend auf einer zufälligen Exploration werden in [Tou04] die Explorationsergebnisse in einem künstlichen neuronalen Netz gespeichert. Der Aufbau dieses Netzes basiert auf einem unüberwachten Lernverfahren aus [Fri97]. Anhand dieses Netzes kann anschließend eine Planung und Vorhersage der nächsten Zustände erfolgen [Tou06]. Die dort durchgeführten Experimente dieses Algorithmus' wurden nur in der Simulation mit einem punktförmigen, masselosen Roboter durchgeführt. Die Funktionalität dieser Netze wird in Abschnitt 3.2 erläutert. Eine Erweiterung des Algorithmus' zur Verwendung mit der M-Serie wird in Abschnitt 3.6 beschrieben. Ähnliche Verfahren unter der Verwendung künstlicher neuronaler Netze zur Navigation von mobilen Robotern werden in [GSK98], [DBJ98] oder auch [Hol08] vorgestellt.

Der in dieser Arbeit entwickelte Algorithmus in Abschnitt 3.6 basiert auf der homöokinetischen Exploration und der Wissensrepräsentation in wachsenden neuronalen Netzen. Erweiterungen, wie beispielsweise eine Belohnung bestimmter Zustände aus [HDH09], werden an die Ziele dieser Arbeit angepasst. Konkret kann dadurch der Selbstschutz oder ein niedriger Energieverbrauch integriert werden. Weiterhin werden neue Ideen eingebracht, um den Algorithmus auch für humanoide Roboter verwenden zu können.

1.3 Die M-Serie - Eine humanoide Roboter-Serie

Die Entwicklung der humanoiden Roboter der M-Serie wurde im Labor für Neurorobotik vom ersten Entwurf an durchgeführt. Während der Entstehung dieser Arbeit wurde der erste Prototyp fertig gestellt und konnte für Experimente verwendet werden. Eine Gesamtdarstellung des Roboters ist in Abbildung 1.1 zu sehen.

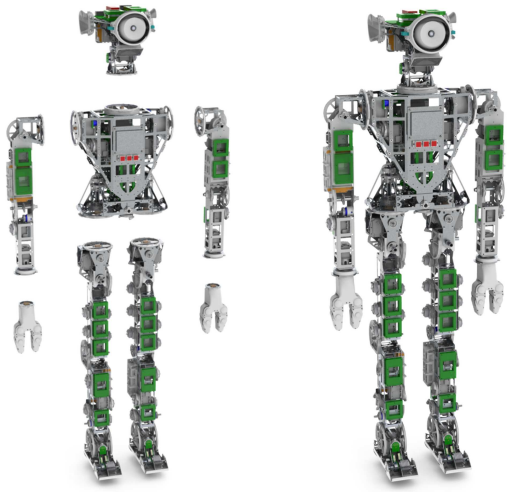

Abbildung 1.1: (links) Komponenten eines Roboters der M-Serie: Kopf, Rumpf, Arme, Greifer, Beine. Jede Komponente verfügt über einen Flansch zur Kopplung der Komponenten (rechts) Gesamtdarstellung der M-Serie.

Ein Roboter der M-Serie besteht aus einzelnen Komponenten, die über Flansche miteinander verbunden werden können. Die einzelnen Komponenten sind Kopf, Rumpf, Arme, Greifer und Beine. Jede Komponente, ausgenommen der Greifer, verfügt über eine eigene Stromversorgung, mehrere Prozessoren, einem Datenbus und zahlreiche Sensoren und Motoren. Als Prozessoren sind auf sogenannten AccelBoard3D (AB3D) verbaut. An den Flanschen sind serielle RS-232 Anschlüsse montiert, an denen die Sensordaten ablesbar und eine externe Stromversorgung möglich sind.

Für die Experimente der Exploration sind Komponenten mit mehreren Freiheitsgraden sinnvoll. Hier bieten sich die Arme in Kombination mit den Greifern und die Beine an. Der Aufbau der Gliedmaßen ist im Wesentlichen gleich, weshalb im Folgenden stellvertretend das rechte Bein, dargestellt in Abbildung 1.2 auf der nächsten Seite, genauer beschrieben wird.

Das Bein besteht aus vier Teilkomponenten: Hüftansatz, Ober- und Unterschenkel und Fuß. Diese sind über die drei Gelenke Hüfte, Knie und Fuß miteinander verbunden. Das Bein hat fünf Freiheitsgrade: Rollen und Neigen in der Hüfte, Neigen im Knie und Rollen und Neigen im Fuß. Die entsprechenden Bewegungen werden mit einer unterschiedlichen Anzahl an Servomotoren pro Gelenk durchgeführt. Die Kraft der Servomotoren wird über Federkupplungen, die mit dem Gelenk über einen Seilzug verbunden sind, an das Gelenk weitergegeben. Die Servomotoren werden von den AccelBoard3D wie in Tabelle 1.1 auf der nächsten Seite aufgeführt gesteuert.

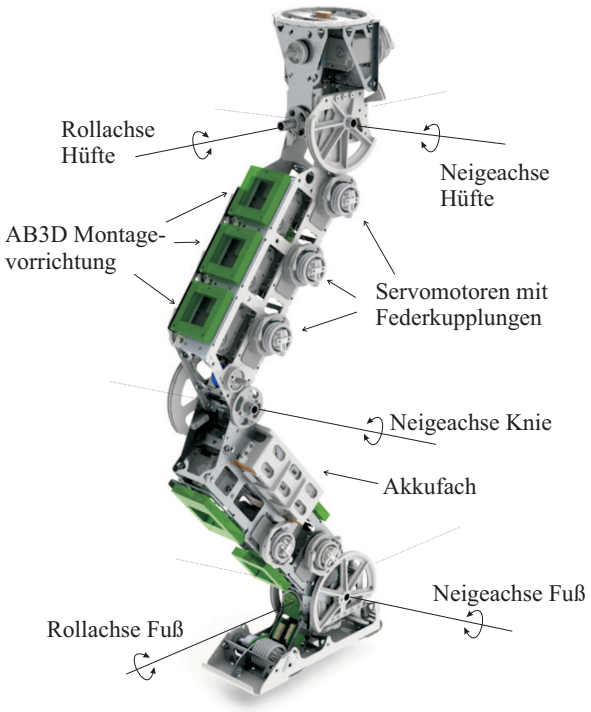

Abbildung 1.2: Rechtes Bein eines Roboters der M-Serie: Es sind die fünf Freiheitsgrade des Beines mit den entsprechenden Achsen markiert. Weiterhin sind die Montageflächen für die AccelBoard3D, die Servomotoren mit montierten Federkupplungen und das Akkufach gekennzeichnet.

Jedes AccelBoard3D ist eine Platine mit einem Prozessor und einem dreidimensionalen Beschleunigungssensor. An jedem AccelBoard3D ist ein Drehpotentiometer des gesteuerten Gelenkes angeschlossen. Weiterhin können zu jedem Servomotor der Stromverbrauch und der Drehwinkel ausgelesen werden. Mit dem AccelBoard3D des Fußes sind außerdem vier piezoelektrische Kraftsensoren an den Ecken der „Fußsohle“ und ein weiteres Drehpotentiometer für den Winkel der Zehen verbunden. Alle AccelBoard3D sind über einen Datenbus, den sogenannten „Spinal Cord“, miteinander und mit dem RS-232 Anschluss am Flansch verbunden. Die Kommunikation über den Datenbus wird im 100 Hz-Takt gesteuert.

Gelenk	Drehrichtung	Servomotor	AccelBoard3D
Hüfte	Rollen	24, 26	AB04
	Neigen	18, 20, 22	AB03
Knie	Neigen	12, 14, 16	AB02
Fuß	Rollen	2	AB01
	Neigen	4, 6, 8, 10	AB00

Tabelle 1.1: Konfiguration der Servomotoren des rechten Beins: Je mehr Kraft für die Bewegung benötigt wird, desto mehr Servomotoren wirken auf das gleiche Gelenk. Es sind ebenfalls die zugehörigen Namen der AccelBoard3D und die Nummern der Servomotoren angegeben.

Kapitel 2

Grundlagen neuronaler Netze

Solange wir nicht wissen, wie menschliche Intelligenz zustande kommt, können wir keine künstliche Intelligenz schaffen.

Reinhard Alfred Furrer (*1940, †1995), deutscher Physiker

Dieses Kapitel führt in das Themengebiet der künstlichen neuronalen Netze ein. Vorab werden grundlegende Eigenschaften dynamischer Systeme erläutert. Anschließend wird ein Neuronenmodell mit zwei unterschiedlichen Umsetzungen vorgestellt. Die Verknüpfung mehrerer Neuronen zu neuronalen Netzen ist Thema des darauf folgenden Abschnittes. Hier werden neben der mathematischen Berechnung auch eine Möglichkeit zur Approximation und neuronale Felder vorgestellt. Weiterhin wird hier die Brücke zwischen neuronalen Netzen und dynamischen Systemen geschlagen. Im letzten Abschnitt werden verschiedene Lernregeln für neuronale Netze beschrieben.

Das Kapitel lehnt sich an die Betrachtungen in [Zel97], [Hil07], [Gal07] und [Hay08] an. Das Kapitel erhebt keinen Anspruch auf Vollständigkeit, sondern es werden nur die für diese Arbeit notwendigen Begriffe und Formeln eingeführt. Diese behalten die gesamte Arbeit über ihre Bedeutung. Um dies zu realisieren, wird bei der Verwendung von referenzierten Algorithmen, Gleichungen oder Beschreibungen von der Nomenklatur der Quelle, wenn nötig, abgewichen.

2.1 Dynamische Systeme

Dynamische Systeme zeichnen sich durch eine Veränderung des Zustandes über die Zeit hinweg aus. Der Zustand ist der Vektor

$$\boldsymbol{x}(t) := (x_1(t) \ \ldots \ x_n(t))^T \in \mathbb{R}^n, \tag{2.1}$$

wobei n die Dimension und t die Zeit des Systems angeben. Ein System wird nach einem bestimmten Startzeitpunkt t_0 betrachtet, so dass $t \geq t_0$ ist. Die Zeit eines dynamischen Systems kann kontinuierlich oder diskret sein. In dieser Arbeit werden ausschließlich

zeitdiskrete Systeme mit $t \in \mathbb{N}$ betrachtet. Alle möglichen Zustände $\boldsymbol{x}(t)$ beschreiben den Phasenraum $\mathbb{P}$ eines dynamischen Systems. Eine Folge von Zuständen

$$T_k := (\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_1 + k - 1)), \; k \in \mathbb{N}\backslash\{0\} \tag{2.2}$$

ist eine Trajektorie der Länge k.

Ein dynamisches System ist durch die iterative Funktion $f : \mathbb{P} \to \mathbb{P}$ beschrieben, die den Phasenraum auf sich selbst abbildet:

$$\boldsymbol{x}(t+1) = f(\boldsymbol{x}(t)). \tag{2.3}$$

Die l-fache Verknüpfung $f \circ f \circ \ldots \circ f$ der Funktion wird mit f^l gekennzeichnet.

Ein dynamisches System kann durch sein Verhalten an Fixpunkten charakterisiert werden. Für einen Fixpunkt $\boldsymbol{x}^*$ gilt:

$$\boldsymbol{x}^* = f(\boldsymbol{x}^*). \tag{2.4}$$

Existiert eine ϵ-Umgebung $\mathbb{U}$ mit $\epsilon > 0$ um den Fixpunkt $\boldsymbol{x}^*$

$$\mathbb{U} := \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| < \epsilon\}, \tag{2.5}$$

wobei die Vektornorm $\|\cdot\|$ den euklidischen Abstand beschreibt, so dass

$$\forall \boldsymbol{x}_0 \in \mathbb{U} : \lim_{l \to \infty} f^l(\boldsymbol{x}_0) = \boldsymbol{x}^* \tag{2.6}$$

gilt, so ist der Fixpunkt lokal stabil. Ein lokal stabiler Fixpunkt $\boldsymbol{x}^*$ ist von einem Basin

$$\mathbb{B} := \left\{\boldsymbol{x} \mid \lim_{l \to \infty} f^l(\boldsymbol{x}) = \boldsymbol{x}^*\right\} \tag{2.7}$$

umgeben. Für jeden Punkt dieser Menge geht das dynamische System für $t \to \infty$ in den Fixpunkt über. Existiert keine ϵ-Umgebung so ist der Fixpunkt instabil.

Ein Fixpunkt $\boldsymbol{x}_p^*$ der p-fachen Ausführung der Funktion

$$\boldsymbol{x}_p^* = f^p\left(\boldsymbol{x}_p^*\right) \tag{2.8}$$

beschreibt einen periodischen Orbit der Periode p der ursprünglichen Funktion

$$\mathbb{O}_p := \left\{\boldsymbol{x} \mid f^l\left(\boldsymbol{x}_p^*\right), \; l \in \{1, \ldots, p\}\right\}. \tag{2.9}$$

Die Invertierung der Funktion f des dynamischen Systems führt zu einem Vertauschen des Verhaltens der Fixpunkte. So wird ein ursprünglich instabiler Fixpunkt nach der Invertierung stabil. Die Invertierung entspricht einer Umkehrung der Zeitskala, so dass das dynamische System in Richtung der Vergangenheit betrachtet wird. Diese Eigenschaft wird in Abschnitt 3.4.1 ab Seite 47 verwendet.

Bei der Betrachtung von physikalischen dynamischen Systemen liegen meist keine eindeutigen Fixpunkte vor. Durch die Reibung des Systems verwischt ein mathematischer

Fixpunkt zu einer Menge. Ein Beispiel ist in Abbildung 3.1 auf Seite 25 dargestellt. Dort ist der Unterschenkel eines M-Serie Roboters als invertiertes Pendel montiert. Der nahezu senkrechte Zustand ist innerhalb eines bestimmten Winkelintervalls ohne Störung von außen stationär. Dieses Intervall markiert den verwaschenen instabilen Fixpunkt dieses dynamischen Systems.

Ein verwaschener Fixpunkt wird als Ruhemenge $\mathbb{F}$ bezeichnet. Eine Ruhemenge beschreibt eine abgeschlossene, zusammenhängende Menge von Fixpunkten. Eine lokal stabile Ruhemenge $\mathbb{F}^s$ liegt vor, wenn gilt:

1. Es existiert eine ϵ-Umgebung $\mathbb{U}$ um einen beliebigen Fixpunkt $\boldsymbol{x}_0^*$ der Ruhemenge mit $\epsilon > 0$ so dass $\mathbb{F}^s$ Teilmenge von $\mathbb{U}$ ist.
2. Von jedem Punkt der Umgebung geht das System für $t \to \infty$ in einen Fixpunkt der Ruhemenge über:

$$\forall \boldsymbol{x}_0 \in \mathbb{U} : \lim_{l \to \infty} f^l(\boldsymbol{x}_0) = \boldsymbol{x}^* \in \mathbb{F}^s \tag{2.10}$$

Ist eine solche Umgebung nicht vorhanden so handelt es sich um eine instabile Ruhemenge $\mathbb{F}^i$. Generell ist jeder einzelne Fixpunkt einer Ruhemenge nach Gleichung (2.6) auf der vorherigen Seite instabil.

Eine Betrachtung der Stabilität der Fixpunkte eines dynamischen Systems kann über dessen Eigenwerte erfolgen. Für diese Betrachtung werden zunächst eindimensionale Systeme herangezogen. Der Eigenwert λ_i ist somit gleich der ersten Ableitung an einem Fixpunkt x_i^*

$$\lambda_i := \frac{\delta}{\delta x} f(x_i^*) \tag{2.11}$$

Je nach Belegung von λ_i hat das dynamische System am Fixpunkt x_i^* ein anderes Verhalten:

Fall 1:	$\lambda_i < -1$:	periodische Orbits möglich
Fall 2:	$-1 < \lambda_i < 1$:	lokal stabiler Fixpunkt
Fall 3:	$1 < \lambda_i$:	instabiler Fixpunkt

Im ersten Fall ist der Betrag von λ_i größer als 1, so dass sich der Zustand des Systems nach einer Störung vom Fixpunkt entfernt. Begrenzt die Funktion f dieses Wachstum so kann durch das negative Vorzeichen ein periodischer Orbit vorliegen. Im zweiten Bereich liegt ein lokal stabiler Fixpunkt vor. Bei einem negativen Vorzeichen von λ_i wird dieser alternierend erreicht. Der Betrag von λ_i ist im dritten Fall wieder größer als 1, so dass sich das System von diesem Fixpunkt wegbewegt. Das positive Vorzeichen führt zu einer monotonen Entfernung, so dass es sich um einen instabilen Fixpunkt handelt. An den Grenzen -1 und 1 müssen höherwertige Ableitungen betrachtet werden, um Aussagen über die Stabilität treffen zu können.

Um eine Stabilitätsanalyse für eine Ruhemenge durchzuführen, muss eine differenzierte rechts- beziehungsweise linksseitige Betrachtung der Randpunkte der Ruhemenge durchgeführt werden. Die sich hier ergebenden Eigenwerte legen nach der oben durchgeführten Fallunterscheidung fest, ob die Ruhemenge insgesamt betrachtet lokal stabil

oder instabil ist. Zu beachten ist hier, dass in Grenzfällen ebenfalls eine lokal stabile Ruhemenge bei $\lambda_i < -1$ auftreten kann.

Das Verhalten eines eindimensionalen dynamischen Systems kann wie in Abbildung 2.1 dargestellt werden. Es sind die Funktion $f(x(t))$ über $x(t)$ und die Identitätsfunktion abgetragen. Der Startwert $x(t_0)$ des Systems ist als leere Raute auf der Identität dargestellt. In jedem Zeitschritt wird der nächste Zustand $x(t+1)$ berechnet und mit dem letzten Zustand $x(t)$ auf der Identität senkrecht verbunden. Anschließend wird $x(t+1)$ waagerecht auf die Identität projiziert. Dieser Ablauf wird zyklisch wiederholt, bis ein Fixpunkt erreicht ist. Ein Fixpunkt des Systems ist durch eine ausgefüllte Raute markiert.

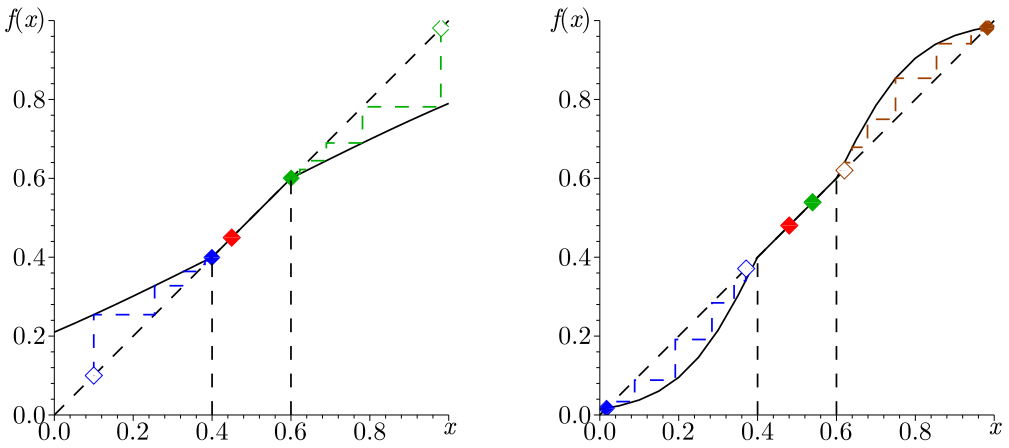

Abbildung 2.1: Verhalten eindimensionaler physikalischer dynamischer Systeme: Beide Systeme haben im geschlossenen Intervall $[0.4, 0.6]$ ein Ruhemenge; (links) lokal stabile Ruhemenge mit den beiden Randpunkten (blau, grün) und einem inliegenden Fixpunkt (rot); (rechts) instabile Ruhemenge mit zwei inliegenden Fixpunkten (rot, grün) umgeben von zwei lokal stabilen Fixpunkten (blau, braun)

Auf der linken Grafik ist eine lokal stabile Ruhemenge dargestellt, wie sie beispielsweise bei einem physikalischen Pendel entstehen würde. Die rechte Grafik zeigt zwei lokal stabilen Fixpunkte in blau und braun. Weiterhin ist eine instabile Ruhemenge dargestellt. Diese Grafik könnte ein invertiertes physikalisches Pendel darstellen. Ähnliche Eigenschaften weist der Aufbau in Abschnitt 3.1 ab Seite 25 auf. Im weiteren Verlauf der Arbeit wird auf die explizite Nennung der lokalen Stabilität zur besseren Lesbarkeit verzichtet und lediglich der Begriff stabil verwendet.

2.2 Mathematische Modelle eines Einzelneurons

Das Modell eines zeitdiskreten Einzelneurons mit $t \in \mathbb{N}$ ist in Abbildung 2.2 auf der nächsten Seite dargestellt.

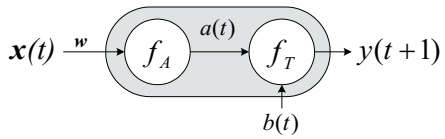

Abbildung 2.2: Modell eines Einzelneurons: Der Eingangsvektor $\boldsymbol{x}(t)$ und der Gewichtsvektor $\boldsymbol{w}$ führen über die Funktion $f_A(\boldsymbol{x}(t), \boldsymbol{w})$ zu der Aktivierung $a(t)$ des Neurons. Der Ausgangswert $y(t+1)$ wird anschließend über die Transferfunktion $f_T(a(t), b(t))$ mit der Grundaktivierung $b(t)$ bestimmt.

Der Eingangsvektor des Neurons ist in Anlehnung an Gleichung (2.1) auf Seite 6 der Vektor

$$\boldsymbol{x}(t) := (x_1(t) \ \ldots \ x_n(t))^T \in \mathbb{R}^n, \tag{2.12}$$

wobei n die Anzahl der Eingänge ist. Anhand des Gewichtsvektors

$$\boldsymbol{w} := (w_1 \ \ldots \ w_n)^T \in \mathbb{R}^n \tag{2.13}$$

wird über die Aktivierungsfunktion $f_A : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ die Aktivierung $a(t) \in \mathbb{R}$ des Neurons bestimmt:

$$a(t) := f_A(\boldsymbol{x}(t), \boldsymbol{w}). \tag{2.14}$$

Die Aktivierung führt mit der Grundaktivierung $b(t) \in \mathbb{R}$ über die Transferfunktion $f_T : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ zum Ausgangswert $y(t+1) \in \mathbb{R}$:

$$y(t+1) := f_T(a(t), b(t)) \tag{2.15}$$

In dieser Arbeit werden zwei unterschiedliche mathematische Realisierungen verwendet: Das STD-Neuron (Standard) und das RBF-Neuron (Radiale Basis Funktion). Beide Modelle werden in den nächsten Abschnitten erläutert.

2.2.1 Das STD-Neuron

Die Aktivierung eines STD-Neurons wird über das Skalarprodukt des Eingangsvektors und des Gewichtsvektors berechnet:

$$f_A^{STD}(\boldsymbol{x}(t), \boldsymbol{w}) := \langle \boldsymbol{x}(t), \boldsymbol{w} \rangle = \boldsymbol{x}(t)^T \boldsymbol{w}. \tag{2.16}$$

Ein STD-Neuron ist genau dann stark aktiviert, wenn die Vektoren in die gleiche Richtung zeigen. Der Ausgangswert eines STD-Neurons kann durch unterschiedliche Transferfunktionen berechnet werden: eine lineare, eine sprunghafte, die Sigmoid oder die Tangens Hyperbolicus Funktion. Die letzten beiden haben ähnliche Eigenschaften und sind über

$$\operatorname{sigm}(x) = \frac{\tanh(x/2) + 1}{2} \tag{2.17}$$

ineinander überführbar. Der Tangens Hyperbolicus verfügt über vier Arbeitsbereiche, die in Abbildung 2.3 auf der nächsten Seite dargestellt sind. Mit diesen Arbeitsberei-

chen sind auch die Sprungfunktion (4) und lineare Funktionen (1) mit zusätzlichen Parametern $c_i \in \mathbb{R}$ approximierbar (vgl. [Hil07, S. 14]):

$$y\left(t+1\right) = c_1 \tanh\left(c_2\, \boldsymbol{x}\left(t\right)^T \boldsymbol{w} + b\left(t\right)\right) + c_3 \tag{2.18}$$

In der vorliegenden Arbeit werden die lineare Funktion der Identität oder der Tangens Hyperbolicus als Transferfunktion verwendet. Wird nicht explizit auf die Identität hingewiesen, so ist die Tangens Hyperbolicus Funktion die Transferfunktion.

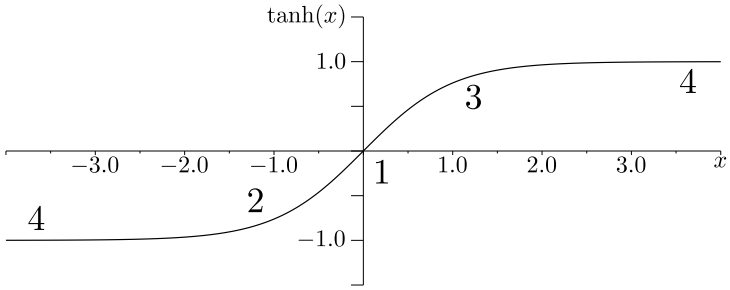

Abbildung 2.3: Der Tangens Hyperbolicus hat vier verschiedene Arbeitsbereiche mit dem folgenden Verhalten: (1) linear, (2) exponentiell, (3) logarithmisch, (4) gesättigt

Da der Tangens Hyperbolicus im offenen Intervall $]-1, 1[$ definiert ist, wird diese Transferfunktion für die Bewegungssteuerung verwendet. Dabei wird $\boldsymbol{y}\left(t\right)$ als Motoraktion interpretiert. So kann ein Motor in zwei Richtungen betrieben werden und es existieren feste Grenzen zur maximalen Ansteuerung. Neben der Bewegungssteuerung des Roboters werden STD-Neuronen ebenfalls zur Prädiktion des Folgezustandes der Umwelt verwendet.

2.2.2 Rekurrenz am STD-Neuron

Dieser Abschnitt betrachtet die Rekurrenz an einem STD-Neuronen ohne weitere Eingänge. In Abbildung 2.4 ist ein Einzelneuron mit Rekurrenz dargestellt.

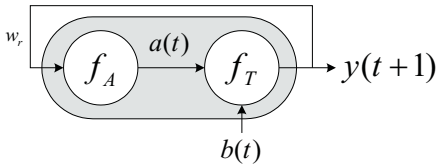

Abbildung 2.4: Einzelneuron mit Rekurrenz: Die Rückkopplung mit dem Gewicht w_r führt den Ausgangswert $y\left(t+1\right)$ im nächsten Zeitschritt als Eingangswert zurück.

Ein rekurrentes Einzelneuron besitzt eine Rückkopplung, die den Ausgangswert $y\left(t+1\right)$ mit dem Rückkopplungsgewicht w_r als Eingangswert im nächsten Zeitschritt zurückführt:

$$y\left(t+1\right) := \tanh\left(w_r y\left(t\right) + b\left(t\right)\right). \tag{2.19}$$

Diese Verschaltung entspricht einem dynamischen System nach Gleichung (2.3) auf Seite 7 mit der Verknüpfung der Funktionen $f_T \circ f_A$ zu f. Abhängig vom Rückkopplungsgewicht w_r können folgende Verhaltensweisen des Neurons unterschieden werden:

Fall 1:	$w_r < -1$:	Periode 2-Orbit
Fall 2:	$-1 < w_r < 0$:	Hochpassfilter beziehungsweise Differentiator
Fall 3:	$0 < w_r < 1$:	Tiefpassfilter beziehungsweise Integrator
Fall 4:	$1 < w_r$:	Hysterese

Bei dieser Fallunterscheidung sind wie in Abschnitt 2.1 auf Seite 8 die Grenzen -1 und 1 ausgenommen. Um die Stabilität hier festzulegen, müssen Ableitungen der Transferfunktion höherer Ordnung betrachtet werden. Weiterhin ist der Spezialfall von $w_r := 0$ ausgeschlossen, da hier ein Einzelneuron ohne Rückkopplung entsteht.

Fall 1: $w_r < -1$

Sei für diese Betrachtungen $b(t) := 0$. Da der Betrag von w_r größer eins ist, würde der Ausgangswert ohne die Transferfunktion unbegrenzt wachsen. Der Tangens Hyperbolicus limitiert das Wachstum und es stellt sich ein betragsmäßig konstanter Wert y^* ein. Da das Vorzeichen von w_r negativ ist, wechselt dieser Wert in jedem Zeitschritt das Vorzeichen. Diese Oszillation ist ein Periode 2-Orbit $\mathbb{O}_2 := \{-y^*, +y^*\}$.

Fall 2: $-1 < w_r < 0$

Im zweiten Wertebereich ist das Vorzeichen von w_r ebenfalls negativ. Für $b(t) := 0$ stellt sich ein stabiler Fixpunkt ein, der alternierend erreicht wird. Ist $b(t)$ ein Signal mit langsamen Amplitudenänderungen, wird ein Großteil dieses Signals subtrahiert, wohingegen schnelle Amplitudenänderungen nahezu unverändert bleiben. Diese Filterart kann demnach als Hochpassfilter bezeichnet werden. Gleichermaßen stellt diese Verschaltung einen nicht idealen Differentiator dar.

Fall 3: $0 < w_r < 1$

Für $b(t) := 0$ existiert hier ein stabiler Fixpunkt. Wird ein Signal für $b(t)$ angelegt, so ist das Neuron ein Tiefpassfilter, das Komplement zum Hochpassfilter. Das positive Vorzeichen führt dazu, dass hohe Frequenzen abgeschwächt werden. Außerdem kann diese Verschaltung als ein nicht idealer Integrator angesehen werden, da ein Gewicht nahe eins zu einer Summation der Einzelwerte führt.

Fall 4: $1 < w_r$

Sei für diese Betrachtungen $b(t) := 0$. Da der Betrag von w_r größer eins ist, stellt sich einer von zwei komplementären stabilen Fixpunkten $\pm y^*$ ein. Dieser ist vom Startwert $y(t_0)$ abhängig. Diese Verschaltung kann einen bestimmten Wert ohne äußere Einwirkung beibehalten.

Einzelne STD-Neuronen mit Rekurrenz werden in Kapitel 3 häufig als Tiefpassfilter verwendet. Weiterhin ist eine eindimensionale Welt aus Abschnitt 3.1 ab Seite 25 als Hysterese-Neuron implementiert.

2.2.3 Das RBF-Neuron

Die Aktivierung eines RBF-Neurons wird über die Aktivierungsfunktion

$$f_A^{RBF}\left(\boldsymbol{x}\left(t\right),\boldsymbol{w}\right) := \left\|\boldsymbol{x}\left(t\right)-\boldsymbol{w}\right\| := \sqrt{\sum_{i=1}^{n}(x_i\left(t\right)-w_i)^2} \tag{2.20}$$

bestimmt. Als Vektornorm $\|\cdot\|$ dient hier der euklidische Abstand der beiden Vektoren $\boldsymbol{x}\left(t\right)$ und $\boldsymbol{w}$. Die Transferfunktion eines RBF-Neurons ist eine Gauß-Funktion

$$f_T^{RBF}\left(a\left(t\right)\right) := \exp^{-0.5\,a(t)^2\,\sigma^{-2}} \tag{2.21}$$

mit der Breite σ ([KM97, S. 1]). Bei dieser Transferfunktion ist die Grundaktivierung $b\left(t\right) := 0$. Je kleiner die Aktivierung des Neurons ist, d.h. je kleiner der Abstand der beiden Vektoren $\boldsymbol{x}\left(t\right)$ und $\boldsymbol{w}$ ist, desto größer ist der Ausgangswert $y\left(t+1\right)$.

Diese Art der Neuronen wird zur Approximation von Funktionen und zur Kategorisierung verwendet. Der Vorteil von RBF-Neuronen gegenüber den STD-Neuronen ist das definierte Verhalten bei einer großen Abweichung von Eingangs- und Gewichtsvektor. Hier strebt die Exponentialfunktion schnell gegen 0, wohingegen bei einem STD-Neuron unterschiedliche Werte angenommen werden können.

2.3 Neuronale Netze

Ein künstliches neuronales Netz (KNN) ist ein gerichteter Graph

$$\mathcal{G} := (\mathbb{V},\mathbb{E}),\ \mathbb{E}\subseteq\mathbb{V}^2 \tag{2.22}$$

mit der Knotenmenge $\mathbb{V} := \{V_1,\ldots,V_n\}$ und der Kantenmenge $\mathbb{E}$. Jeder Knoten wird durch ein Neuron dargestellt. In jedem Netz sind ausgewiesene Mengen an Knoten als Eingangs- und Ausgangsneuronen mit n_E und n_A gekennzeichnet. Die zugehörigen Vektoren sind der Eingangsvektor $\boldsymbol{x}_E\left(t\right)\in\mathbb{R}^{n_E}$ und der Ausgangsvektor $\boldsymbol{y}_A\left(t\right)\in\mathbb{R}^{n_A}$. Alle anderen Knoten n_V sind versteckte Neuronen. Jede Kante ist eine Verbindung von einem V_j zu einem V_i mit einem Gewicht w_{ij}. Alle Gewichte des Graphen sind in einer Gewichtsmatrix $\boldsymbol{W}\in\mathbb{R}^{n\times n}$ darstellbar. Existiert keine Verbindung von V_j zu V_i, so ist das Gewicht $w_{ij} := 0$.

Eine statische Grundaktivierung der Neuronen kann über ein ON-Neuron erfolgen. Dieses Neuron hat keinen Eingang und der Ausgangswert ist immer 1. Dieses Neuron wird mit V_0 bezeichnet. Die Grundaktivierung kann dann über die erste Spalte $\boldsymbol{W}_{\cdot 0}$ der Gewichtsmatrix festgelegt werden.

Künstliche neuronale Netze können in zwei Klassen unterteilt werden: Feed-Forward- und Rekurrente-Netze. Je nach interner Struktur können diese Klassen weiter spezifiziert werden (vgl. [Zel97, S. 78]).

Feed-Forward-Netze (FFN) sind azyklische Graphen. Schichtenweise verbundene FFN sind Netze, die in k Schichten eingeteilt werden können, so dass nur Kanten von einer Schicht i in die nächste $i + 1$ existieren. Ist eine solche Einteilung nicht möglich liegt ein allgemeines FFN vor. Diese Netze können keinen internen Zustand speichern und werden beispielsweise als FIR-Filter (finite impulse response) verwendet. In dieser Arbeit werden mit dieser Netzstruktur Prädiktoren der Umwelt und Regler für den Roboter entwickelt. Bei diesen Reglern wird ausgenutzt, dass die Umwelt den Zustand des Systems speichert. So kann über diese eine vollständige sensomotorische Schleife etabliert werden.

Rekurrente Netze zeichnen sich durch Rückkopplungen aus. Dadurch kann das Netz selbst einen Zustand speichern. Je nach Art der Rückkopplung können vereinfacht folgende Klassen unterschieden werden:

- Netze mit direkte Rückkopplungen: Neuronen des Netzes besitzen direkte Rückkopplungen zu sich selbst. Diese Art der Rückkopplung wurde in Abschnitt 2.2.2 ab Seite 11 vorgestellt. Über Einstellung des Rückkopplungsgewichtes w_r kann ein Neuron beispielsweise als Hysterese-Neuron agieren.

- Netze mit Rückkopplungen innerhalb einer Schicht: Grundlage dieses Netzes ist ein mehrschichtiges FFN. Durch laterale Kanten innerhalb einer Schicht können sich die Neuronen gegenseitig hemmend oder aktivierend beeinflussen. Diese Struktur wird verwendet, wenn möglichst nur ein Neuron aus einer Schicht einen hohen Ausgangswert haben soll. Dieses Neuron hemmt dann alle anderen Neuronen dieser Schicht. Diese Netzart wird in dieser Arbeit zur Kategorisierung unter anderem in Abschnitt 3.6 ab Seite 53 verwendet.

- Zufällige oder vollständig verbundene Netze: Diese Netze kennzeichnen sich durch keine besondere Struktur aus. Es können direkte und indirekte Rückkopplungen vorhanden sein. Ein vollständig verbundenes Netz kennzeichnet sich durch eine Gewichtsmatrix aus, bei der jeder Eintrag ungleich 0 ist.

Bisher wurde bereits ein Einzelneuron mit Rückkopplung den dynamischen Systemen zugeordnet. Wird allgemein ein künstliches neuronales Netz verwendet, so werden in der vorliegenden Arbeit die Eingangswerte $x_i(t)$ als sensorische Messwerte und die Ausgangswerte $y_i(t)$ als Motoraktionen interpretiert. Wie bereits erwähnt, kann die sensomotorische Schleife über die Umwelt geschlossen werden. Das dynamische System besteht somit aus dem künstlichen neuronalen Netz und der Umwelt. Wird die Umwelt als Funktion f_W aufgefasst, die aus der Motoraktion einen neuen sensorischen Zustand berechnet, so kann die Gesamtfunktion f des dynamischen Systems durch $f_W \circ f_T \circ f_A$ beschrieben werden.

2.3.1 Berechnung der Ausgabe eines Netzes

Bei der Berechnung des Ausgangsvektors eines Netzes mit STD-Neuronen wird immer ein ON-Neuron verwendet. Weiterhin werden keine direkten oder indirekten Rückkopplungen auf die Eingangsneuronen und das ON-Neuron zugelassen. Dadurch sind die ersten $n_E + 1$ Zeilen der Gewichtsmatrix $\boldsymbol{W}$ mit Nullen besetzt. Daher werden diese Zeilen gelöscht und die Matrix hat anschließend eine Dimension von $\mathbb{R}^{(n_V+n_A)\times(n+1)}$. Die neuen Indizes der Gewichte sind $w_{(i-n_E-1),j}$.

Der Eingabevektor des Netzes

$$\boldsymbol{x}(t) := \begin{pmatrix} 1 & \boldsymbol{x}_E(t)^T & \boldsymbol{y}(t)^T \end{pmatrix}^T \in \mathbb{R}^{n+1} \tag{2.23}$$

besteht aus dem Wert 1 des ON-Neurons, dem Eingangsvektor $\boldsymbol{x}_E(t)$ der Eingangsneuronen und dem Ausgabevektor des aktuellen Zeitschrittes $\boldsymbol{y}(t)$. Der gesamte Ausgabevektor

$$\boldsymbol{y}(t) := \begin{pmatrix} \boldsymbol{y}_V(t)^T & \boldsymbol{y}_A(t)^T \end{pmatrix}^T \in \mathbb{R}^{n_V+n_A}, \tag{2.24}$$

umfasst die Ausgänge der versteckten Neuronen $\boldsymbol{y}_V(t)$ und der Ausgangsneuronen $\boldsymbol{y}_A(t)$. Zum Startzeitpunkt der Berechnung t_0 wird der gesamte Ausgabevektor mit $\boldsymbol{y}(t_0) := \boldsymbol{0}$ initialisiert. Die Berechnung zu nächsten Zeitschritten $t > t_0$ erfolgt über die Funktion

$$\boldsymbol{y}(t+1) := f_T(\boldsymbol{W}\boldsymbol{x}(t)). \tag{2.25}$$

Die Transferfunktion wird auf den resultierenden Vektor elementweise angewendet. Alle Ausgangsvektoren der in den folgenden Kapiteln verwendeten Netze mit STD-Neuronen werden nach diesem Schema berechnet.

Werden RBF-Neuronen als versteckte Neuronen verwendet, so entfällt die Verwendung des ON-Neurons. Weiterhin besteht ein solches Netz meist aus nur einer verdeckten Schicht. Anhand des Eingangsvektors wird für jedes RBF-Neuron der Ausgangswert bestimmt. Dieser Wert wird anschließend innerhalb der Schicht verteilt und führt zu einer Hemmung oder Anregung der verbundenen Neuronen. Diese Verteilung kann auf unterschiedliche mathematische Weise umgesetzt werden. In Abschnitt 2.3.3 ab Seite 16 werden die Grundlagen erläutert.

2.3.2 Approximation der Ausgabe eines Netzes

Die hier beschriebene Approximation bezieht sich auf Netze mit STD-Neuronen. Um die Ausgabe eines Netzes zu approximieren, wird das Netz als lineares System betrachtet. Wird eine lineare Transferfunktion verwendet, liegt bereits ein lineares System vor. Als Approximationsmatrix $\boldsymbol{L}(\boldsymbol{x}(t))$ kann direkt die Gewichtsmatrix $\boldsymbol{W}$ verwendet werden. Wird der Tangens Hyperbolicus verwendet, so wird die jeweilige partielle Ableitung der Transferfunktion nach der aktuellen Eingabe $\boldsymbol{x}(t)$ gebildet:

$$\boldsymbol{L}(\boldsymbol{x}(t)) := \frac{\delta}{\delta \boldsymbol{x}} f_T(\boldsymbol{W}\boldsymbol{x}(t)). \tag{2.26}$$

Jeder Wert l_{ij} der entstehenden Matrix, auch Jacobi-Matrix genannt, wird durch

$$l_{ij} := w_{ij}(1 - \tanh^2(\boldsymbol{W}_{i.}\boldsymbol{x}(t))) \tag{2.27}$$

bestimmt, wobei $\boldsymbol{W}_{i.}$ die Zeile der Gewichtsmatrix mit dem Index i kennzeichnet. Die Approximation des Netzes wird über ein Euler-Verfahren realisiert:

$$\boldsymbol{y}(t+1) \approx \boldsymbol{y}(t) + \boldsymbol{L}(\boldsymbol{x}(t)) \cdot (\boldsymbol{x}(t-1) - \boldsymbol{x}(t)) \tag{2.28}$$

Durch diese Approximation kann das System auch invertiert werden. Durch Umstellung der Gleichung und der Berechnung der pseudoinversen Matrix $\boldsymbol{L}(\boldsymbol{x}(t))^+$ folgt

$$\boldsymbol{x}(t-1) \approx \boldsymbol{x}(t) - \boldsymbol{L}(\boldsymbol{x}(t))^+ \cdot (\boldsymbol{y}(t+1) - \boldsymbol{y}(t)). \tag{2.29}$$

Diese Approximation wird in Abschnitt 3.4 ab Seite 46 verwendet, um eine Stabilisierung entgegen der Zeitskala durchzuführen. Diese wird mit einer Stabilisierung entlang der Zeitskala kombiniert und führt zu einer kontinuierlichen Veränderung des Netzes. Das Verfahren zur Stabilisierung eines Netzes wird in Abschnitt 2.4.1 ab Seite 18 erläutert.

2.3.3 Neuronale Felder

Die Betrachtung von neuronalen Feldern basiert meist auf auf einem einschichtigen neuronalen Netz aus RBF-Neuronen. Es kann eine Schicht zur Eingabe von Sensorwerten vorgeschaltet werden. Die Schicht der RBF-Neuronen hat häufig eine bestimmte Struktur, wie beispielsweise einen eindimensionalen Ring [Tou04] oder eine zweidimensionale Karte [Koh90].

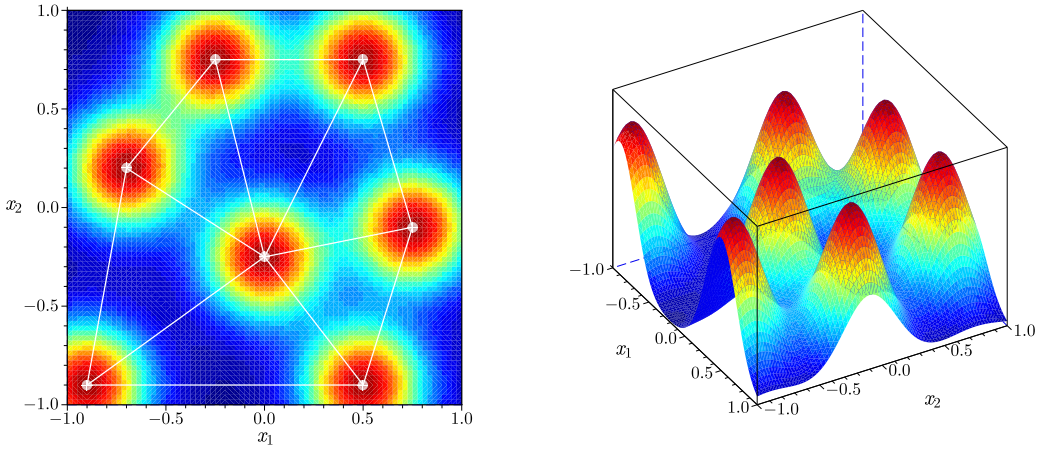

Abbildung 2.5: Neuronale Feldaktivität: (links) Darstellung eines Netzes aus RBF-Neuronen (weiß) mit Feldaktivität (farbig); (rechts) dreidimensionale Darstellung der Feldaktivität;

In einem neuronalen Feld, wie es in Abbildung 2.5 abgebildet ist, wird die überlagerte Feldaktivität jedes Neurons repräsentiert. Durch das zugrunde liegende neuronale Netz

kann eine Kategorisierung eines hochdimensionalen Eingangsvektors innerhalb eines einfachen Netzes erfolgen und anschließend aus dem neuronalen Feld eine Bewegung berechnet werden.

In Abschnitt 2.3.1 ab Seite 15 wurde bereits die Berechnung des neuronalen Feldes skizziert. Nach der Berechnung des Ausgangswertes für jedes einzelne RBF-Neuron wird über die lateralen Verbindungen die Feldaktivität $z\left(t\right)$ jedes Neurons bestimmt, die zusammen das neuronale Feld repräsentieren. Das neuronale Feld unterliegt somit ebenfalls einer zeitlichen Änderung, die als Felddynamik bezeichnet wird.

Eine allgemeine Gleichung für die Feldaktivität eines Neurons V_i ist

$$z_{V_i}\left(t+1\right) := \tau_z z_{V_i}\left(t\right) + \left(1-\tau_z\right)\left(y_{V_i}\left(t\right) + b_{V_i}\left(t\right) + \phi\left(z_{V_j}\left(t\right)\right)\right). \tag{2.30}$$

Der Parameter τ_z bestimmt die Dynamik der Feldaktivität. Neben der alten Feldaktivität $z_i\left(t\right)$, dem Ausgangswert $y_{V_i}\left(t\right)$ und der Grundaktivierung $b_{V_i}\left(t\right)$ geht ebenfalls ein lateraler Term $\phi\left(z_{V_j}\left(t\right)\right)$ in die Gleichung ein. Hier wirkt die Feldaktivität eines oder mehrerer anderer Neuronen V_j auf das aktuelle Neuron ein. Unterschiedliche Realisierungen für die Interaktion sind in Abschnitt 3.4.1 ab Seite 47 und Abschnitt 3.6 ab Seite 53 beschrieben.

Grundlegende Eigenschaften neuronaler Felder wie Stabilität oder Interaktivität wurden in [Ama77] untersucht. Basierend auf vorgegebenen neuronalen Netzen wurden diese Felder unter anderem zur Navigation eines Roboters in [GSK98], [DBJ98] oder [Hol08] verwendet. Um die statische Vorgabe eines Netzes abzulösen, wurden selbstorganisierende Karten („Self-Organizing Maps“, SOM) entwickelt. Einen guten Überblick gibt [Koh90]. Diese Karten ermöglichen es, dass die Anordnung der Neuronen variabel ist und sie sich an die eingehenden Sensordaten anpassen können. Dadurch kann erreicht werden, dass das entstehende neuronale Feld genau an den notwendigen Stellen detailliert aufgelöst ist.

Die fest vorgegebene Struktur der Karten kann ebenfalls durch sich selbst aufbauende Netze, wie etwa einem „growing neural gas“ (GNG), ersetzt werden. Oft zitierte Quellen sind hier die Arbeiten [Fri93], [Fri95] oder [KM97]. Eine Erweiterung, um nicht nur den Aufbau, sondern auch den Abbau kontrolliert zu gestalten, wird in [Fri97] vorgestellt. Hier wird dem GNG Verfahren ein Nützlichkeitswert (engl. „Utility“) hinzugefügt. Der erweiterte Algorithmus wird als GNG-U bezeichnet. Der Vorteil des selbst organisierenden Aufbaus des Netzes liegt in der daraus resultierenden, bestmöglichen Abdeckung der Sensorsignale. Durch die Möglichkeit des kontrollierten Abbaus des Netzes können ebenfalls veraltete Bereiche des Netzes entfernt werden. Weiterhin entstehen so Netze, die wenige Kanten aufweisen und es werden nur die Neuronen verbunden, die eine topologische Nähe haben.

In dieser Arbeit werden aufgrund der eben genannten Vorteile wachsende neuronale Netze verwendet. Auf den Roboter übertragen, könnte ein neuronales Netz beispielsweise das Aufstehen aus einer liegenden Position erlernen. Im Laufe des Lernprozesses würde sich das Netz ebenfalls in Bereiche ausbilden, in denen der Roboter umfällt. Wird schließlich eine geeignete Bewegung zum Aufstehen gefunden, so können die Be-

wegungsmuster, bei denen der Roboter keinen Erfolg erzielt, vergessen werden. Dieses Vergessen wird durch den Abbau des Netzes an der entsprechenden Stelle realisiert. Der GNG-U Algorithmus wird in Abschnitt 2.4.2 ab Seite 20 erläutert.

2.4 Lernverfahren neuronaler Netze

Der in dieser Arbeit entwickelte Algorithmus verwendet zwei Lernverfahren:

1. Überwachtes Lernen
2. Unüberwachtes Lernen

Ziel des überwachten Lernens ist es, die Gewichtsmatrix eines vorgegebenen neuronalen Netzes für eine bestimmte Aufgabe anzupassen. Bei diesem Verfahren ist eine Menge von Zweitupeln aus jeweils einem Eingangs- und einem erwarteten Ausgangsvektor $(\boldsymbol{x}_E, \boldsymbol{v})$ vorhanden. An das Netz wird ein zufällig aus der Menge ausgewählter Eingangsvektor angelegt und der berechnete Ausgangsvektor mit dem erwarteten Ausgangsvektor verglichen. Nun wird über eine bestimmte Lernregel die entstehende Abweichung zur Veränderung der Gewichte des Netzes verwendet. Ziel ist es neben der Vorhersagbarkeit der Trainigsdaten ebenfalls eine Generalisierung zu erreichen, so dass ähnliche neue Vektorpaare ebenfalls richtig vorhergesagt werden können. Diese Art des Lernens ist üblicherweise die schnellste, da die genaue Ausgabe der Ausgangsvektoren bekannt ist.

Gibt es keine Antwort an das neuronale Netz, so handelt es sich um ein unüberwachtes Lernverfahren. Das Netz lernt ähnliche Eingabevektoren zusammenzufassen und den gesamten Eingaberaum zu kategorisieren. Die zugrunde liegende Verteilung der Eingabevektoren kann durch Nachbarschaftsbeziehungen dargestellt werden. Im Gegensatz zum überwachten Lernen ist somit keine Netzstruktur vorgegeben. Diese wird während des Lernens aufgebaut.

Neben diesen beiden Lernverfahren existieren weitere, wie beispielsweise bestärkendes oder evolutionäres Lernen. Die genauen Ausgangsvektoren sind beim bestärkenden Lernen nicht bekannt. Es wird über eine Belohnung ausgesagt, ob etwas gut oder schlecht war. In Abschnitt 3.2.2 ab Seite 37 und Abschnitt 3.3 ab Seite 42 wird diese Art des Lernens aufgegriffen, jedoch nicht vertieft behandelt. Hintergrundinformationen zu bestärkenden Lernen können beispielsweise [SB98], [Wat89] oder [DW01] entnommen werden. Evolutionäres Lernen lehnt sich an die natürliche Evolution an indem eine Population bewertet und anschließend eine neue Generation aus den am besten bewerteten der vorhergehenden Population gebildet wird.

2.4.1 Überwachtes Lernen

Ein überwachtes Lernverfahren führt im Allgemeinen für alle Paare von Eingangs- und Ausgangsvektoren $(\boldsymbol{x}_E, \boldsymbol{v})$ folgende Schritte durch (vgl. [Zel97, S.94]:

1. Auswahl eines Eingangsvektors $\boldsymbol{x}_E$
2. Berechnung des Ausgangsvektors $\boldsymbol{y}_A$
3. Bestimmung eines Vorhersagefehlers mit $f_e(\boldsymbol{y}_A, \boldsymbol{\upsilon})$ anhand des erwarteten Ausgangsvektors $\boldsymbol{\upsilon}$
4. Berechnung der Gewichtsänderungen anhand des Fehlerwertes
5. Änderung aller Gewichte im neuronalen Netz

Dieser Ablauf passt ausschließlich die Gewichte des Netzes an. Da die Grundaktivierung nicht als Gewicht vorliegt wird wiederum ein ON-Neuron eingeführt. Die ursprüngliche Grundaktivierung ist dann das jeweils abgehende Verbindungsgewicht vom ON-Neuron.

Es gibt zahlreiche Verfeinerungen dieses allgemeinen Vorgehens, die sich vor allem bei der Berechnung des Fehlers und der Gewichtsänderung unterscheiden. Im weiteren Verlauf dieser Arbeit werden einschichtige Netzwerke verwendet. Aufgrund dessen wird hier ein Verfahren für einschichtige Netze vorgestellt, die sogenannte Delta-Regel.

Nach der Auswahl und der Berechnung aus den ersten beiden Schritten wird der entstandene Fehler ermittelt. Eine mögliche Fehlerfunktion für den dritten Schritt ist der quadratische euklidische Abstand von $\boldsymbol{y}_A$ und $\boldsymbol{\upsilon}$:

$$f_e(\boldsymbol{y}_A, \boldsymbol{\upsilon}) := \frac{1}{2}\|\boldsymbol{y}_A - \boldsymbol{\upsilon}\|^2 \tag{2.31}$$

Der Vorfaktor dient einer vereinfachten Rechnung für die erste Ableitung der Funktion.

Im vierten Schritt werden nun die Gewichtsänderungen des Netzes berechnet. Die Gewichtsänderungen werden anhand des Gradienten der Fehlerfunktion bestimmt. Um diesen Gradientenabstieg durchzuführen, wird die Fehlerfunktion nach jedem Element w_{ij} der Gewichtsmatrix $\boldsymbol{W}$ abgeleitet. So entsteht ein Aktualisierungsterm für jedes Element:

$$\Delta w_{ij} = \epsilon_w(y_i - \upsilon_i)\left(\frac{\delta}{\delta w_{ij}} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_{i.}^T\right)\right)\right). \tag{2.32}$$

Der Term $\boldsymbol{W}_{i.}^T$ beschreibt die Zeile mit dem Index i als Spaltenvektor. Die Lernrate des Verfahrens ist $\epsilon_w \in]0, 1[$.

Je nach dem verwendetem Neuronenmodell und der Transferfunktion setzt sich der letzte Term wie folgt zusammen:

- Für ein STD-Neuron mit der Identität als Transferfunktion ist der Ableitungsterm

$$\frac{\delta}{\delta w_{ij}} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_{i.}^T\right)\right) := x_j \tag{2.33}$$

  und für den Tangens Hyperbolicus gilt

$$\frac{\delta}{\delta w_{ij}} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_{i.}^T\right)\right) := w_{ij}\left(1 - \tanh^2\left(\boldsymbol{W}_{i.}\boldsymbol{x}_E\right)\right). \tag{2.34}$$

- Bei der Verwendung von RBF-Neuronen in einem Netz wird neben dem Gewichtsvektor ebenfalls die Breite $\boldsymbol{\sigma}$ aktualisiert. Der Ableitungsterm aus Gleichung (2.32) auf der vorherigen Seite ist für ein RBF-Neuron

$$\frac{\delta}{\delta w_i} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_{i.}^T\right)\right) := 2\boldsymbol{y}_{A,i}\sigma_i^{-2}\left(\boldsymbol{x}_E - \boldsymbol{W}_{i.}^T\right). \tag{2.35}$$

  Für die Aktualisierung der Breite wird die Fehlerfunktion aus Gleichung (2.31) auf der vorherigen Seite nach der Breite $\boldsymbol{\sigma}$ abgeleitet. Für jedes einzelne Neuron folgt als Aktualisierung

$$\Delta\sigma_i = \epsilon_\sigma(y_i - \upsilon_i)\left(\frac{\delta}{\delta\sigma_i} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_i\right)\right)\right), \tag{2.36}$$

  wobei $\epsilon_\sigma \in]0,1[$ die Lernrate ist. Der Ableitungsterm ist

$$\frac{\delta}{\delta\sigma_i} f_T\left(f_A\left(\boldsymbol{x}_E, \boldsymbol{W}_i\right)\right) := 2\boldsymbol{y}_{A,i}\sigma_i^{-1} f_A^{RBF}\left(\boldsymbol{x}_E, \boldsymbol{W}_{i.}^T\right) \tag{2.37}$$

  mit der Aktivierungsfunktion f_A^{RBF} aus Gleichung (2.20) auf Seite 13.

Abschließend werden die erhaltenen Abweichungen Δw_{ij} und gegebenenfalls $\Delta\sigma_i$ im fünften Schritt mit den ursprünglichen Werten aufaddiert. Nach einer Iteration wird erneut beim ersten Schritt begonnen.

Für jeden zu lernenden Gewichtswert oder die zu lernende Breite wird nur ein Parameter benötigt: die Lernrate. Diese ist im Allgemeinen für jedes dynamische System neu zu bestimmen. Ein weiteres Problem dieses Lernverfahrens sind lokale Minima der Fehlerfunktion. Ist die Lernrate zu klein gewählt, können diese nicht verlassen werden. Ist im Gegenzug die Lernrate zu groß, so können Oszillationen auftreten. Um diese Problematik zu vermeiden, wird die Lernrate häufig einem „simulated annealing“ unterzogen. Das bedeutet, dass die Lernrate über die Zeit hinweg immer kleiner wird:

$$\epsilon(t+1) = \tau_\epsilon\epsilon(t) + b_\epsilon,\ \tau_\epsilon \approx 1,\ b_\epsilon \ll 1 \tag{2.38}$$

Dieses Verhalten kann mit einem einzelnen rekurrenten Neuron mit der Identität als Transferfunktion realisiert werden.

Dieses Lernverfahren wird vor allem in Abschnitt 3.4 ab Seite 46 und Abschnitt 3.6.1 ab Seite 54 verwendet, um eine Stabilisierung des dynamischen Systems entlang und entgegen der Zeitskala zu erreichen. Durch diesen Mechanismus wird sich das dynamische System ständig verändern und es ist nicht notwendig, dass ein globales Minimum der Fehlerfunktion gefunden wird. Da „simulated annealing“ ebenfalls dem Ziel des lebenslangen Lernens widerspricht, wird diese Technik in dieser Arbeit nicht verwendet.

2.4.2 Unüberwachtes Lernen

Unüberwachte Lernverfahren wurden bereits in Abschnitt 2.3.3 ab Seite 16 aufgegriffen. Das „growing neural gas“-Lernverfahren (GNG) aus [Fri95] ist eine Möglichkeit

eine Repräsentation eines Eingaberaumes in einem neuronalen Netz aufzubauen. Bei der Erweiterung zum GNG-U Algorithmus nach [Fri97] unterliegen die entstehenden Neuronen und Verbindungen einer Nützlichkeitsanalyse und einem Alterungsprozess. Sind Teile des Feldes zu alt oder werden sie nicht mehr verwendet, so wird das Feld an dieser Stelle wieder abgebaut.

Sei das neuronale Netz der Graph $\mathcal{G} := (\mathbb{V}, \mathbb{E})$ mit der Knotenmenge $\mathbb{V} := \{V_1, \ldots, V_m\}$ und der Kantenmenge $\mathbb{E}$. Sei weiterhin ein Knoten

$$V_i := (\boldsymbol{w}, e, u) \in \mathbb{V} \tag{2.39}$$

mit einem Gewichtsvektor $\boldsymbol{w} \in \mathbb{R}^n$, einem Fehlerwert $e \in \mathbb{R}_0^+$ und einem Nützlichkeitswert $u \in \mathbb{R}_0^+$. Zwei Knoten $V_p, V_q \in \mathbb{V}$ sind durch eine Kante

$$E_{pq} := (V_p, V_q, \alpha) \in \mathbb{E} \tag{2.40}$$

verknüpft. Eine Kante hat ein bestimmtes Alter $\alpha \in \mathbb{R}_0^+$. Die Menge

$$\mathbb{V}_{V_i} := \{V_i \mid \exists\, E_{ij} \in \mathbb{E} \vee \exists\, E_{ki} \in \mathbb{E}\} \tag{2.41}$$

ist die Menge aller Nachbarn von V_i.

Der Algorithmus verwendet als Vektornorm $\|\cdot\|$ die euklidischen Länge des Vektors. Der Ablauf des Algorithmus kann in elf Schritte unterteilt werden:

1. Initialisiere Graph $\mathcal{G}$ mit zwei Knoten $\mathbb{V} := \{V_1, V_2\}$ und $\mathbb{E} := \emptyset$, wobei die beiden Gewichtsvektoren $\boldsymbol{w}_{V_1}, \boldsymbol{w}_{V_2}$ zufällig gewählt werden. Die Fehlerwerte e werden mit 0 und die Nützlichkeitswerte u mit 1 initialisiert.

2. Wähle ein zufälliges Eingangssignal $\boldsymbol{x}(t) \in \mathbb{P}$.

3. Bestimme den Knoten V_w mit dem geringsten Abstand zu $\boldsymbol{x}(t)$ mit

   $$V_w = \min_{V_i} \|\boldsymbol{x}(t) - \boldsymbol{w}_{V_i}\|,\ V_i \in \mathbb{V} \tag{2.42}$$

   und den Knoten V_v mit dem zweitkleinsten Abstand zu $\boldsymbol{x}(t)$ mit

   $$V_v = \min_{V_i} \|\boldsymbol{x}(t) - \boldsymbol{w}_{V_i}\|,\ V_i \in \mathbb{V}\backslash\{V_w\}. \tag{2.43}$$

4. Falls noch keine Kante E_{wv} zwischen V_w und V_v existiert, so füge die Kante $E_{wv} := (V_w, V_v, 0)$ in $\mathbb{E}$ ein. Existiert die Kante, so setze deren Alter $\alpha_{E_{wv}}$ auf 0.

5. Summiere die Fehlervariable von V_w mit

   $$\Delta e_{V_w} = \|\boldsymbol{x}(t) - \boldsymbol{w}_{V_w}\|^2 \tag{2.44}$$

   und summiere den Nützlichkeitswert von V_w mit

   $$\Delta u_{V_w} = \|\boldsymbol{x}(t) - \boldsymbol{w}_{V_v}\|^2 - \|\boldsymbol{x}(t) - \boldsymbol{w}_{V_w}\|^2. \tag{2.45}$$

6. Summiere den Gewichtsvektor von V_w mit

$$\Delta \boldsymbol{w}_{V_w} = \epsilon_w \left(\boldsymbol{x}\left(t\right) - \boldsymbol{w}_{V_w} \right) \tag{2.46}$$

und alle Nachbarn von V_w mit

$$\Delta \boldsymbol{w}_{V_i} = \epsilon_v \left(\boldsymbol{x}\left(t\right) - \boldsymbol{w}_{V_i} \right), \; V_i \in \mathbb{V}_{V_w}, \tag{2.47}$$

wobei die Lernraten ϵ_w und ϵ_v aus dem offenen Intervall $]0,1[$ sind und $\epsilon_v \ll \epsilon_w$ ist.

7. Inkrementiere das Alter alle Kanten, die von Knoten V_w ausgehen, um 1.

8. Entferne alle Kanten, deren Alter größer als das maximale Alter $\alpha_{max} \in \mathbb{N}$ ist. Sollten Knoten ohne zugehörige Kante vorhanden sein, werden diese ebenfalls entfernt.

9. Ist die Anzahl der bearbeiteten Eingangssignale ein Vielfaches einer vorgegebenen Konstanten $k \in \mathbb{N}$, so wird ein neuer Knoten eingefügt. Bestimme den Knoten V_e mit maximalem Fehler e_{V_e} und den Knoten V_f mit dem maximalen Fehler aller Nachbarn von V_e. Reduziere die Fehler der beiden Knoten um βe_{V_e} beziehungsweise βe_{V_f} mit $\beta \in]0,1[$. Füge einen neuen Knoten V_g mit

$$V_g := \left(\frac{1}{2}(\boldsymbol{w}_{V_e} + \boldsymbol{w}_{V_f}), \frac{1}{2}(e_{V_e} + e_{V_f}), 1 \right) \tag{2.48}$$

in $\mathbb{V}$ ein. Lösche die alte Kante zwischen V_e und V_f und füge zwei neue Kanten $E_{eg} := (V_e, V_g, 0)$ und $E_{gf} := (V_g, V_f, 0)$ ein.

10. Reduziere den Fehler und den Nützlichkeitswert aller Knoten $V_i \in \mathbb{V}$ um γe_{V_i} beziehungsweise γu_{V_i} mit $\gamma \ll \beta$. Finde den Knoten mit dem kleinsten Nützlichkeitsmaß V_l und den Knoten mit dem größten Fehlerwert V_m. Wenn gilt

$$u_{V_l} < \frac{e_{V_m}}{|\mathbb{V}|}, \tag{2.49}$$

dann wird der Knoten V_l und alle dazugehörigen Kanten entfernt.

11. Solange kein Abbruchkriterium erfüllt ist, gehe zurück zu 2. Mögliche Abbruchkriterien sind die Gesamtzahl an Knoten oder der mittlere Fehler über alle Knoten.

Dieses Lernverfahren weist im Gegensatz zur Delta-Regel mehr Parameter auf:

- Anpassung des Gewinner-Neurons durch ϵ_w und dessen Nachbarn durch ϵ_v
- Maximales Kantenalter α_{max}
- Abklingraten für den Fehler und die Nützlichkeit mit β und γ
- Einfügen eines neuen Knoten nach k Schritten

Abhängig von diesen Parametern wird der Eingaberaum unterschiedlich kategorisiert. So kann beispielsweise mit den Anpassungsparametern die Variabilität des Netzes festgelegt werden. Je größer diese Werte sind, desto flexibler ist das Netz. Auch hier könnte wieder ein „simulated annealing“ verwendet werden um das Netz über die Zeit hinweg zu stabilisieren. Das maximale Kantenalter und die Abklingraten wirken sich ebenfalls auf die Dynamik des Netzes aus. Sie bestimmen wie schnell sich das Netz auf eine veränderte Verteilung im Eingaberaum anpassen kann.

Dieser Algorithmus dient als Basis für die sensomotorischen Karten aus [Tou04] und [Tou06]. Ein grundlegender Unterschied zwischen dem hier vorgestellten und dem angepassten Algorithmus besteht in der Reihenfolge der Eingabevektoren. Hier wird im ersten Schritt vorausgesetzt, dass ein Eingangsvektor zufällig aus dem Phasenraum entnommen wird. Bei der Exploration des Phasenraumes werden die Eingabevektoren jedoch kontinuierlich und aufeinander folgend sein. Diese und weitere Anpassungen an die speziellen Anforderungen einer Exploration des Phasenraumes werden in Abschnitt 3.2 ab Seite 34 im nächsten Kapitel beschrieben.

Kapitel 3

Analyse von Explorationsalgorithmen

Der Blick des Forschers fand nicht selten mehr,
als er zu finden wünschte.

Gotthold Ephraim Lessing (*1729, †1781), deutscher Dichter

Aufbauend auf den Grundlagen über neuronale Netze werden in diesem Kapitel vier Algorithmen zur Exploration des sensorischen beziehungsweise sensomotorischen Raumes vorgestellt. Die Funktionsweise der Algorithmen wird formal und anhand eines Beispiels erläutert. Weiterhin wird eine Untersuchung anhand folgender Fragestellungen durchgeführt:

- Wird ein Weltmodell benötigt?
- Wie werden die Explorationsergebnisse repräsentiert?
- Wie wird die Exploration gesteuert und kann die Exploration gezielt beeinflusst werden?
- Kann die Exploration unbegrenzt fortgesetzt werden?
- Wie kann auf die Explorationsergebnisse zugegriffen werden?
- Kann sich das System an eine wandelnde Umwelt anpassen?
- Wie hoch ist die Komplexität des Algorithmus'?
- Ist der Algorithmus neuronal implementierbar?

Nach einer Zusammenfassung der Algorithmen, in der Gemeinsamkeiten und Unterschiede herausgestellt werden, wird ein neuer Explorationsalgorithmus vorgestellt. Dieser vereint vorhandene Elemente aus den vorgestellten Algorithmen und erweitert diese im Hinblick auf die Ziele dieser Arbeit.

3.1 Exploration über stabile Systemzustände

In [Hea08] wird ein Algorithmus zur Analyse eines dynamischen Systems vorgestellt. Im Rahmen dieser Arbeit wurde der Algorithmus implementiert und an einem Roboter der M-Serie getestet. Dieser Algorithmus wird am ausführlichsten erläutert, da bisher keine Veröffentlichungen existieren.

Das exemplarische Beispiel für diesen Algorithmus ist ein eindimensionales dynamisches System. In der Simulation entspricht es einer Hysterese und in der Realität einem invertierten Pendel. Der Aufbau und die im weiteren Verlauf verwendeten Bezeichnungen sind in Abbildung 3.1 dargestellt.

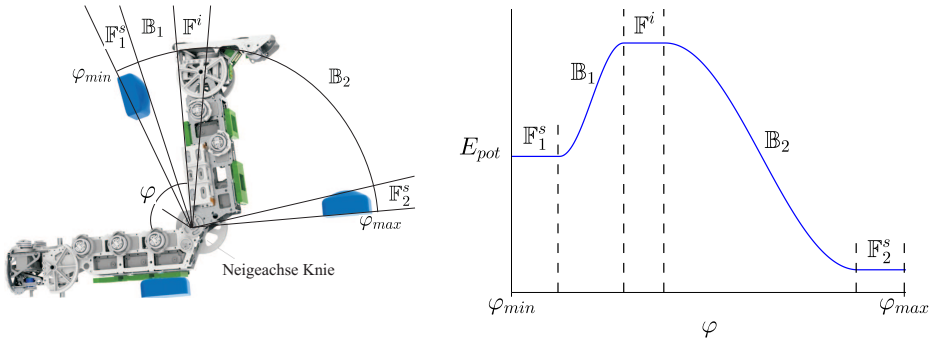

Abbildung 3.1: Aufbau des Beispiels: (links) In der Grafik sind folgende Elemente abgebildet: waagerechte Fixierung des Roboterbeines, Haltepunkte für den Ober- und Unterschenkel, stabile Ruhemengen $\mathbb{F}_1^s$ und $\mathbb{F}_2^s$ mit dem jeweils zugehörigen Basin $\mathbb{B}_1$ und $\mathbb{B}_2$, instabile Ruhemenge $\mathbb{F}^i$ und der Winkel zwischen Ober- und Unterschenkel φ. (rechts) In der Grafik ist die potentielle Energie über den Winkel φ abgetragen. Hier sind ebenfalls die Ruhemengen und Basins gekennzeichnet.

Der sensorische Eingangsvektor reduziert sich auf den Winkel $\varphi(t)$ zwischen Unter- und Oberschenkel und der motorische Ausgangsvektor ist das Drehmoment $m(t)$ auf das Gelenk. Die sensomotorische Schleife wird über die Umwelt zu einem dynamischen System zusammengeführt.

Ziel des Algorithmus' ist es die Ruhemengen im Phasenraum zu finden. In diesem Szenario sind dies die beiden stabilen Ruhemengen $\mathbb{F}_1^s$ und $\mathbb{F}_2^s$ und die instabile Ruhemenge $\mathbb{F}^i$. Wie in Abschnitt 2.1 ab Seite 8 bereits erläutert, werden hier keine Fixpunkte, sondern Ruhemengen angenommen. Diese Mengen resultieren aus der physikalischen Reibung in den Gelenken des Roboters. Dieses Verhalten wurde ebenfalls in die Simulation übernommen.

Um die markanten Bereiche zu finden, besteht der Algorithmus aus einem aktiven und einem passiven Teil. Befindet sich das System in einem stabilen Zustand, wird die aktive Steuerung gestartet. Dieser Zeitpunkt wird in t^* festgehalten. Für einen bestimmten Zeitraum $\delta_t \in \mathbb{N}$ wird eine motorische Aktion durchgeführt. Die auszuführenden Mo-

toraktionen $m(t)$ können durch ein einzelnes Drehmoment $M(t^*)$ beschrieben werden. Dieses kann sich konstant auf die Einzelwerte verteilen

$$m(t) := \frac{M(t^*)}{\delta_t + 1} \tag{3.1}$$

oder eine Impulsform, wie beispielsweise

$$m(t) := \left(\sum_{t=t^*}^{t^*+\delta_t} \sin\left(\frac{t - t^* + 1}{\delta_t + 2} \pi \right) \right)^{-1} M(t^*) \sin\left(\frac{t - t^* + 1}{\delta_t + 2} \pi \right), \tag{3.2}$$

annehmen, wobei jeweils $t \in \{t^*, \ldots, t^* + \delta_t\}$ ist. Beide Anregungen stellen sicher, dass das verteilte Drehmoment gleich dem Gesamtdrehmoment ist:

$$\sum_{t=t^*}^{t^*+\delta_t} m(t) = M. \tag{3.3}$$

Anschließend wird das System sich selbst überlassen, bis es erneut einen stabilen Zustand erreicht hat. Der Übergang von einem stabilen Zustand einer Ruhemenge in einen anderen Zustand in der gleichen oder einer anderen Ruhemenge wird durch

$$\mathbb{F}_i \overset{(d,M)}{\rightarrow} \mathbb{F}_j \tag{3.4}$$

dargestellt, wobei $d \in \{-1, 1\}$ die Explorationsrichtung und M das verwendete Drehmoment sind.

Die markanten Punkte werden in einem sogenannten Experten,

$$V_k := (\mathbb{F}_k, \mathbb{B}_k, \mathbb{M}_{k,max}, \mathbb{M}_{k,min})$$

abgespeichert. Jeder Experte umfasst eine Ruhemenge $\mathbb{F}_k$ und das zugehörige Basin $\mathbb{B}_k$. Die Ruhemengen werden als geschlossenes Intervall $[F_{k,min}, F_{k,max}]$ abgespeichert. Das Basin $\mathbb{B}_k$ ist durch das offene Intervall $]B_{k,min}, B_{k,max}[$ definiert. Handelt es sich um eine instabile Ruhemenge, so werden die Grenzen von Basin und Ruhemenge gleich sein. Dies führt zu einem Verhalten als wäre kein Basin vorhanden. Weiterhin hat ein Experte zwei Mengen an Drehmomenten:

$$\mathbb{M}_{k,max} := \{(d, M) \mid \mathbb{F}_k \overset{(d,M)}{\rightarrow} \mathbb{F}_k\}$$

und

$$\mathbb{M}_{k,min} := \{(d, M) \mid \mathbb{F}_k \overset{(d,M)}{\rightarrow} \mathbb{F}_j,\ k \neq j\}.$$

Die beiden Mengen beschreiben in Abhängigkeit der Explorationsrichtung das maximale Drehmoment, mit dem die Ruhemenge nicht, und das minimale Drehmoment, mit dem die Ruhemenge verlassen werden kann. Das Neuron erhält hier die Bezeichnung Experte, da die Information, wie der aktuelle Zustand verlassen werden kann, vorhanden ist.

Die Experten werden in einem Expertengraph $\mathcal{G} := (\mathbb{V}, \mathbb{E})$ abgespeichert. Eine Kante

$$E_{ij} := (V_i, V_j, d) \in \mathbb{E} \tag{3.5}$$

enthält neben der Information des Start- und Zielexperten ebenfalls die Information der Explorationsrichtung d.

3.1.1 Beschreibung des Algorithmus'

Der grundlegende Ablauf des Algorithmus, der zu jedem Zeitschritt abgearbeitet wird, ist in Abbildung 3.2 in einem Flussdiagramm dargestellt.

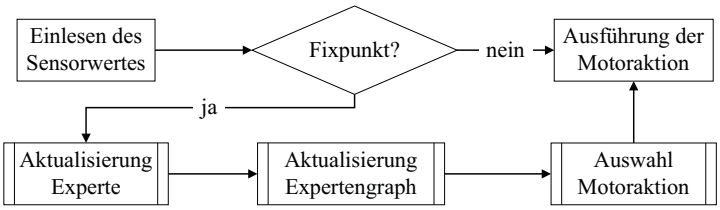

Abbildung 3.2: Flussdiagramm des Algorithmus': Wird nach dem Einlesen der Sensoren ein Fixpunkt detektiert, wird der untere Pfad durchlaufen. Nach der Aktualisierung des Experten und des Expertengraphen wird eine neue Motoraktion ausgewählt. Abschließend wird diese Motoraktion über mehrere Zeitschritte verteilt ausgeführt.

Zunächst wird nach dem Einlesen des Sensorwertes $\varphi(t)$ detektiert, ob ein Fixpunkt vorliegt. Ein solcher Zustand wird durch die Zustandsänderung $\delta(t)$ des Systems mit

$$\delta(t+1) = \tau_\varphi \delta(t) + (1 - \tau_\varphi) \left| \varphi(t) - \varphi(t+1) \right| \tag{3.6}$$

mit der Zeitkonstante $\tau_\varphi \in]0,1[$ erkannt. Gilt $\delta(t+1) < \epsilon_\varphi$, mit $\epsilon_\varphi \in]0,1[$ und $\epsilon_\varphi \ll 1$, so ist ein Fixpunkt erreicht. Befindet sich das System noch in Bewegung, wird nur die Motoraktion $m(t)$ oder keine Aktion ausgeführt. Liegt ein Fixpunkt vor wird der untere Pfad des Flussdiagramms abgearbeitet.

Aktualisierung des Experten

Zunächst wird der zugehörige Experte V_k mit $\varphi(t) \in [F_{k,min} - \epsilon_f, F_{k,max} + \epsilon_f]$ aus $\mathcal{G}$ ermittelt. Die Ruhemenge wird mit $\epsilon_f \in]0,1[$ erweitert. Der Parameter ϵ_f sollte deutlich kleiner als eins sein, so dass die Ruhemenge sehr langsam wachsen kann.

- Existiert der Experte nicht, so wurde dieser Zustand noch nicht besucht und der Experte wird mit $B_{k,min} := B_{k,max} := F_{k,min} := F_{k,max} := \varphi(t)$ neu angelegt und zu $\mathcal{G}$ hinzugefügt.
- Existiert der Experte, so werden die Ruhemenge

$$\mathbb{F}_k := [\min\{\varphi(t), F_{k,min}\}, \max\{\varphi(t), F_{k,max}\}] \tag{3.7}$$

und das Basin

$$\mathbb{B}_k :=]\min\{\rightarrow \varphi(t), B_{k,min}\}, \max\{\rightarrow \varphi(t), B_{k,max}\}[\tag{3.8}$$

aktualisiert, wobei $\rightarrow \varphi(t)$ alle Sensorwerte ab dem letzten Zeitpunkt der Aktionsauswahl t^* darstellt.

Aktualisierung des Expertengraphen

Anschließend wird, wenn es notwendig ist, eine neue Kante $(V_{k-1}, V_k, d(t^*))$ zwischen dem letzten und dem aktuellen Experten mit der letzten Explorationsrichtung $d(t^*)$ in $\mathcal{G}$ angelegt. Die entsprechende Menge $\mathbb{M}$ und die Basingrenzen der Experten werden wie folgt aktualisiert:

- Ist der Experte V_{k-1} nicht verlassen worden, dann wird das Tupel $(d(t^*), M(t^*))$ der Menge $\mathbb{M}_{k-1,max}$ hinzugefügt:

$$\mathbb{M}_{k-1,max} = \mathbb{M}_{k-1,max} \cup \{(d(t^*), M(t^*))\} \tag{3.9}$$

- Ist der Experte V_{k-1} verlassen worden, so wird dessen Menge $\mathbb{M}_{k-1,min}$ angepasst:

$$\mathbb{M}_{k-1,min} = \mathbb{M}_{k-1,min} \cup \{(d(t^*), M(t^*))\} \tag{3.10}$$

Auswahl der Motoraktion

Nach diesem Schritt sind die vorliegenden Informationen der Bewegung seit dem letzten Zeitpunkt t^* verarbeitet. Nun wird t^* aktualisiert mit $t^* := t$. Danach wird eine Motoraktion generiert. Hierfür wird zunächst die Explorationsrichtung $d(t^*)$ zufällig ausgewählt. Anschließend wird das Drehmoment $M(t^*)$ bestimmt. Für die Bestimmung werden zunächst zwei Teilmengen definiert:

- Die Menge aller Drehmomente, mit denen der aktuelle Experte in die Explorationsrichtung $d(t^*)$ verlassen worden ist:

$$\mathbb{M}^*_{min} := \{M \mid (d, M) \in \mathbb{M}_{k,min},\ d = d(t^*)\} \tag{3.11}$$

Ist die Menge nicht leer, so wird das minimale Drehmoment, mit dem der Experte verlassen wurde, ermittelt:

$$M_{min} := \min_{\mathbb{M}^*_{min}} \{M\} \tag{3.12}$$

- Die Menge aller Drehmomente, mit denen der aktuelle Experte in die Explorationsrichtung $d(t^*)$ nicht verlassen worden ist:

$$\mathbb{M}^*_{max} := \{M \mid (d, M) \in \mathbb{M}_{k,max},\ d = d(t^*)\} \tag{3.13}$$

Ist die Menge nicht leer, so wird das maximale Drehmoment, mit dem der Experte nicht verlassen wurde, ermittelt:

$$M_{max} := \max_{\mathbb{M}^*_{max}} \{M\} \tag{3.14}$$

Mit diesen Zwischenwerten und -mengen kann das neue Drehmoment wie folgt berechnet werden:

$$M\left(t^*\right) = d\left(t^*\right) \cdot \begin{cases} 0.5 & \text{wenn } \mathbb{M}^*_{min} = \emptyset \text{ und } \mathbb{M}^*_{max} = \emptyset \\ 0.9 M_{min} & \text{wenn nur } \mathbb{M}^*_{max} = \emptyset \\ 1.1 M_{max} & \text{wenn nur } \mathbb{M}^*_{min} = \emptyset \\ 0.5(M_{max} + M_{min}) & \text{sonst} \end{cases} \tag{3.15}$$

Das resultierende Drehmoment wird abschließend über δ_t Zeitschritte verteilt an den Roboter geschickt. Ist die Zeitspanne abgelaufen, wird so lange keine Aktion ausgeführt, bis erneut ein Fixpunkt erreicht wird.

3.1.2 Experimente

Es wird zunächst auf die Ergebnisse einer Simulation mit einem Hysterese-Neuron eingegangen. Anschließend wird der Algorithmus mit einem Roboter der M-Serie mit dem Aufbau, der in Abbildung 3.1 auf Seite 25 dargestellt ist, getestet. Für beide Experimente wurde eine motorische Ansteuerung nach Gleichung (3.2) auf Seite 26 verwendet.

Abbildung 3.3 auf der nächsten Seite zeigt die Testergebnisse des Algorithmus' mit einem Hysterese-Neuron. Im oberen Teil ist der zeitliche Verlauf des Ausgangswertes des Neurons aufgetragen. Weiterhin sind die beiden stabilen Ruhemengen blau und die instabile Ruhemenge rot markiert. Nach etwa 1000 Zeitschritten reicht der Motorimpuls aus, um die untere stabile Ruhemenge zu verlassen und bereits bei etwa 4000 Zeitschritten ist der Motorimpuls so gut eingestellt, dass die instabile Ruhemenge genau erreicht und ein weiterer Experte etabliert wird. Nach etwa 6700 Zeitschritten ist der Motorimpuls auch von der unteren Ruhemenge so gut eingestellt, dass die instabile Ruhemenge genau erreicht wird.

Im unteren Bereich sind die Motorimpulse über die Zeit aufgetragen. Gerade innerhalb der ersten 4000 Zeitschritte ist der Anstieg der Impulse zu sehen. Dieses Anwachsen wird unterbrochen, wenn die Ruhemengen verlassen werden. Ab diesem Zeitpunkt beginnt eine Einstellung des optimalen Impulses zum Erreichen der nächsten Ruhemenge.

In weiteren Experimenten wurde der Algorithmus an einem Roboter der M-Serie getestet. Ein beispielhafter Verlauf der Exploration ist in Abbildung 3.4 auf Seite 31 dargestellt. Es sind die direkt empfangenen Sensordaten dargestellt, die auch fünf Spikes mit einem sehr hohen Wert beinhalten. Als Testaufbau wurde das Bein wie in Abbildung 3.1 auf Seite 25 montiert. Die Haltepunkte wurden mit einer Polsterung versehen, so dass der Roboter keinen Schaden nimmt. Der Versuchsaufbau ist hier so angeordnet, dass der Roboter von der annähernd senkrechten Position mit angewinkeltem Bein wenig

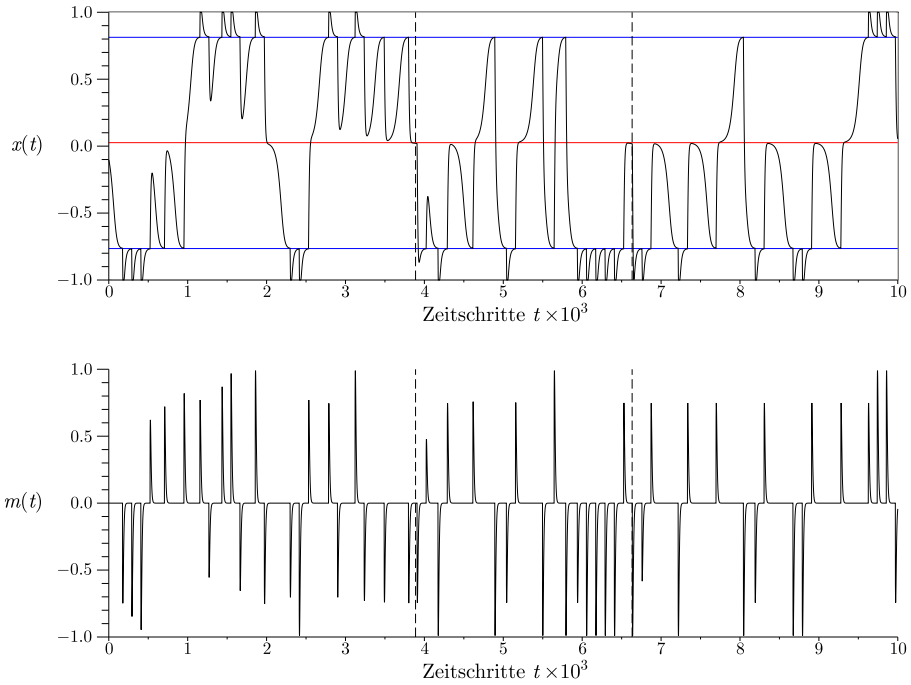

Abbildung 3.3: Testergebnisse der Simulation mit einem Hysterese-Neuron: (oben) Zeitlicher Verlauf des Ausgangswertes des Neurons mit den beiden stabilen Ruhemengen (blau) und der instabilen Ruhemenge (rot). Es ist deutlich zu erkennen, dass die Exploration von Motoraktion zu Motoraktion sich weiter von den stabilen Ruhemengen entfernt. (unten) Zeitlicher Verlauf der Motorimpulse. Jeder Impuls geht über mehrere Zeitschritte. Die steigende Amplitude des Impulses ist der Grund für die Exploration von Bereichen, die weiter von der Ruhemenge entfernt sind.

Energie zum Übergang in den annähernd waagerechten Zustand benötigt. Umgekehrt muss zusätzlich die Kraft der Erdanziehung aufgebracht werden.

Die Abbildung zeigt den zeitlichen Verlauf des Winkels zwischen Ober- und Unterschenkel beginnend in der waagerechten Position. Bei ca. 0.1 ist der senkrechte und bei ca. -0.6 der waagerechte Haltepunkt. Der Motorimpuls wächst bis zum Zeitschritt 13.000 kontinuierlich an, bis erstmalig der waagerechte Zustand verlassen werden konnte. Die stabilen Ruhemengen sind durch die gestrichelten Linien markiert. Es zeigt sich, dass das Bein unterschiedliche Zustände innerhalb dieser Regionen annimmt. Dies ist auf die Polsterung der Haltepunkte zurückzuführen, da hier eine breite Ruhemenge existiert. Hier ist es möglich, dass sich mehr als ein Experte an einer Ruhemenge etabliert. Auf diese Problematik und mögliche Lösungen der mehrfachen Besetzung einer Ruhemenge wird in der Zusammenfassung in Abschnitt 3.1.4 ab Seite 32 eingegangen.

3.1.3 Kategorisierung

Wird ein Weltmodell benötigt?

Es wird kein Weltmodell benötigt, da der Algorithmus keine Prädiktion verwendet.

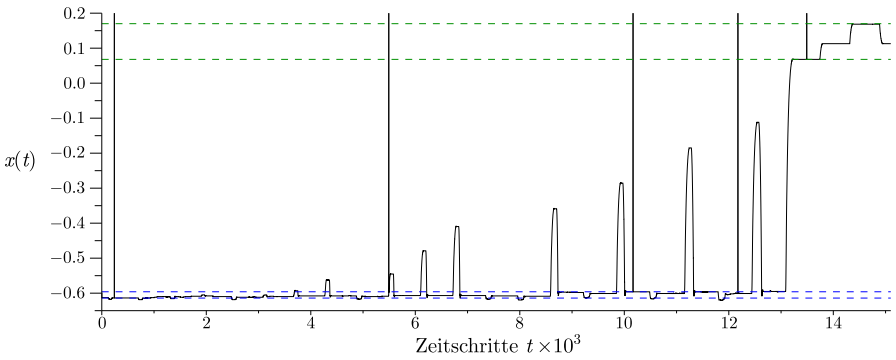

Abbildung 3.4: Testergebnisse mit einem Roboterbein: Zeitlicher Verlauf des Winkels zwischen Unter- und Oberschenkel des Roboterbeines mit zwei entstandenen Experten (grün, blau).

Wie werden die Explorationsergebnisse repräsentiert?

Der Algorithmus baut eine Repräsentation des sensorischen Raumes durch Experten auf. Die einzelnen Experten verfügen weiterhin über motorische Informationen. Alle Explorationsergebnisse sind in einem Expertengraphen gespeichert, wobei jeder einzelne Experte Kenntnis über seine Nachbarn und deren Erreichbarkeit hat.

Wie wird die Exploration gesteuert und kann die Exploration gezielt beeinflusst werden?

Die Steuerung der Exploration erfolgt über eine direkte Richtungs- und Drehmomentauswahl. Durch Veränderungen an diesem Auswahlprozess kann direkt in die Steuerung des Systems eingegriffen werden.

Kann die Exploration unbegrenzt fortgesetzt werden?

Die Exploration kann unbegrenzt fortgesetzt werden. Ein reales System besitzt nicht genügend markante Bereiche, so dass der Expertengraph zu groß werden würde. Einzig das kontinuierliche Speichern aller Wertepaare (d, M) müsste angepasst werden. Es könnte eine fixe Größe der Mengen vorgegeben werden. Ist diese Grenze überschritten, so könnte das älteste Element gelöscht werden. Andere Möglichkeiten sind das Kombinieren von nahe beieinander liegenden Drehmomenten oder gar das Abspeichern eines einzigen Mittelwertes pro Richtung.

Wie kann auf die Explorationsergebnisse zugegriffen werden?

Auf die Explorationsergebnisse kann über die Experten zugegriffen werden. Eine freie Planung eines Bewegungsablaufes im Phasenraum ist jedoch nicht einfach möglich, da es nur wenige Experten und somit große Basins gibt. Innerhalb eines Basins gibt es keine Information wie bestimmte Orte erreicht werden können. Jedoch könnte über

bestimmte Steuereinheiten eine Trajektorie generiert werden, bei der die gespeicherten Aktionen kombiniert werden.

Kann sich das System an eine wandelnde Umwelt anpassen?

Das System kann sich schnell an neue Gegebenheiten anpassen. Zum einen sei hier die Auswahl der Drehmomente angeführt, die sich in wenigen Iterationen auf einen nahezu optimalen Wert einstellen. Zum anderen ist der Expertengraph sehr variabel gestaltbar, so dass Kanten, die durch eine äußere Einwirkung nicht mehr realisierbar sind, nach wenigen Fehlversuchen direkt gelöscht werden können.

Wie hoch ist die Komplexität des Algorithmus'?

Die Algorithmuskomplexität ist maximal aus der Klasse $O(n)$, wobei n die Anzahl an Experten beziehungsweise die Größe der Mengen mit den Drehmomenten ist. Neben einfachen Operatoren aus der Komplexitätsklasse $O(1)$ sind die Suche im Expertengraphen nach dem passenden Experten und die Operatoren min und max auf die Drehmomentmengen die aufwändigsten Teilalgorithmen mit einer Komplexität von $O(n)$.

Ist der Algorithmus neuronal implementierbar?

Der Ansatz ist grundsätzlich neuronal implementierbar. Wird das Neuronenmodell erweitert, so dass die zusätzlichen Informationen eines Experten und einer Kante gespeichert werden können, kann der Algorithmus neuronal implementiert werden, da alle mathematischen Operatoren ebenfalls umsetzbar sind.

3.1.4 Zusammenfassung

Wie bereits erwähnt, ist der Algorithmus im Rahmen dieser Diplomarbeit implementiert und an einem Roboter der M-Serie getestet worden. Hier konnte der erläuterte Verlauf des Algorithmus bestätigt werden. Es werden erst die stabilen Ruhemengen gefunden, da das notwendige Drehmoment zum Erreichen der instabilen Ruhemengen erst durch die Anpassung der maximalen und minimalen Drehmomente durch den vierten Fall der Gleichung (3.15) auf Seite 29 eingestellt wird. Weiterhin wurde in den Experimenten eine Einteilung des Phasenraumes, wie in Abbildung 3.1 auf Seite 25 dargestellt, erreicht.

Der Algorithmus zeichnet sich durch seinen klaren Aufbau und die wenigen Parameter aus. Diese sind τ_φ in Kombination mit ϵ_φ und ϵ_f. Über τ_φ kann die Detektion eines Fixpunktes reguliert werden. Je näher dieser Parameter sich dem Wert 1 nähert, desto mehr Zeitschritte werden benötigt, bis ein Fixpunkt erkannt wird. Eine ähnliche Auswirkung hat eine Verkleinerung des Wertes von ϵ_φ. Der Parameter ϵ_f regelt das Wachstum einer Ruhemenge.

Eine Erweiterungsmöglichkeit wäre neben dem Anwachsen auch das Schrumpfen und Entfernen von Ruhemengen zu ermöglichen. In jedem Übergang hat eine Ruhemenge die Möglichkeit über ϵ_f zu wachsen. Bei jedem Eintreffen in die Ruhemenge könnten die Grenzen auch um den Faktor $(1 - \epsilon_f)$ schrumpfen. Ruhemengen, die einen Grenzwert unterschreiten, könnten anschließend gelöscht werden. Mit dieser Erweiterung könnte sich das entstehende Netz ebenfalls auf rasche Veränderungen der Umwelt einstellen, wenn sich Ruhemengen verschieben oder sie verschwinden.

Allerdings wurden auch einige Fragen aufgeworfen, die nicht beantwortet werden konnten, beispielsweise wie groß ϵ_f sein sollte. Ist es zu klein gewählt, können mehrere Experten in einem stabilen Bereich entstehen (siehe Abbildung 3.4 auf Seite 31). Ist es zu groß gewählt, können nahe beieinander liegende Ruhemengen zusammengefasst und nicht mehr unterschieden werden.

Wie kann eine Ruhemenge in $\mathbb{R}^n$ interpretiert werden? Bleibt die Intervallschachtelung erhalten, so könnte eine kreisförmige, elliptische oder quadratische Fläche als Ruhemenge dienen. Durch die Verwendung eines RBF-Neurons könnten die Grenzen zwischen den Experten auch fließend sein und der Ausgangswert könnte als Wahrscheinlichkeit für die Zugehörigkeit zu einem Experten stehen. Eine weitere Idee wäre den stabilen Bereich mit einem selbstwachsenden Netz aus einem GNG-Algorithmus abzudecken. Der große Vorteil wäre, dass Regionen beliebiger Gestalt abgebildet werden könnten. Eine ähnliche Variante wird in Abschnitt 3.2 ab Seite 34 vorgestellt.

Ein weiteres Problem sind die Intervallgrenzen der Basins. Es ist möglich, dass sich das Basin eines stabilen mit der Ruhemenge einer instabilen Ruhemenge schneiden. Dies ist vor allem auf die Roboter der M-Serie zurückzuführen. So kann in ein und derselben Position einmal der instabile Zustand gehalten werden und ein anderes mal fällt das Bein in einen stabilen Zustand zurück. Eine Lösungsmöglichkeit wäre das Erweitern des sensorischen Eingangsvektors. Neben den aktuellen Winkeln müssten dann ebenfalls die Winkeländerungen betrachtet werden.

Eine ähnliche Problematik werfen die minimalen und maximalen Drehmomente auf. Je mehr sich die Drehmomente annähern, desto höher wird die Wahrscheinlichkeit, dass das Ziel nicht mehr erreicht wird, da jeder Roboter bei der gleichen Ansteuerung leicht anders reagiert. Im Allgemeinen kann somit das minimale beziehungsweise maximale Drehmoment eines Experten nicht auf einen Wert reduziert werden.

Weitere Aufgaben ergeben sich bei einem mehrdimensionalen Sensorraum. Wie wird hier die Richtung der Exploration ausgewählt? Es könnten die Jacobi-Matrix und anschließend die Eigenwerte bestimmt werden. Eine weitere Möglichkeit ist die Umkehrung der Richtung, mit der die Ruhemenge erreicht wurde. Neben der Richtung an sich stellt sich dann auch die Frage nach dem Speicherplatz und der Geschwindigkeit. Wird die Explorationsrichtung in wenige Richtungen diskretisiert, so kann das Modell unverändert bleiben. Sobald $d\,(t^*)$ jedoch kontinuierlich wird, ist es bereits bei zwei Dimensionen fraglich, ob der Algorithmus noch funktioniert.

Andere Erweiterungen wären der Einsatz zusätzlicher Kriterien neben dem Drehmoment. Durch die Annäherung an das minimale Drehmoment strebt der Algorithmus

automatisch zu energetisch geringen Zuständen. Wird in höheren Dimensionen eine Analyse der Explorationsrichtung durchgeführt, so kann auch hier die Exploration entlang der energetisch niedrigen Richtungen der Eigenwerte erfolgen.

3.2 Sensomotorische Kartografierung durch zufallsgesteuerte Exploration

In [Tou04] wird ein Algorithmus zur Exploration des sensomotorischen Raumes und der Repräsentation der Ergebnisse in neuronalen Netzen mit einer anschließenden Planung vorgestellt. Erweiterungen um eine Vorhersage beziehungsweise Antizipation der Folgezustände werden in [Tou06] erläutert. Der Schwerpunkt dieser Arbeit liegt gleichermaßen auf der Exploration und der Planung, die in den nächsten beiden Abschnitten beschrieben werden. Die Exploration basiert auf einem angepassten GNG-U Algorithmus nach [Fri97]. Die folgenden Erläuterungen basieren somit vorwiegend auf diesen drei Arbeiten und es wird an gegebener Stelle auf ausführlichere Erläuterungen verwiesen.

Für die Beschreibung des Algorithmus' wird das zweidimensionale System aus [Tou06] verwendet. Der sensorische Wertebereich liegt im Intervall von $[-1,1]^2$ und beschreibt eine Position in einem ebenen Quadrat. Innerhalb dieser Welt existiert ein masseloser, punktförmiger Roboter, dem eine diskrete Anzahl k an Bewegungsrichtungen zur Verfügung steht:

$$\varphi_i := 2\pi(1 - \frac{i}{k}),\ i \in \{1, \ldots, k\} \tag{3.16}$$

Der Motorvektor $\boldsymbol{m}(t) \in [0,1]^k$ in jede Bewegungsrichtung k führt zu einer Ortsänderung

$$\boldsymbol{\delta}(t) := \sum_{i=1}^{k} m_i(t) \begin{pmatrix} \cos(\varphi_i) \\ \sin(\varphi_i) \end{pmatrix}, \tag{3.17}$$

die über ein einfaches Euler-Verfahren

$$\widetilde{\boldsymbol{x}}(t+1) := \boldsymbol{x}(t) + \tau_x \boldsymbol{\delta}(t),\ \tau_x \in]0,1[\tag{3.18}$$

zu einer neuen Position führt. Die neue Position wird anschließend auf den Wertebereich des Quadrates projiziert:

$$x_i(t+1) = \begin{cases} 1 & \text{für } \widetilde{x}_i(t+1) \geq 1 \\ \widetilde{x}_i(t+1) & \text{für } -1 < \widetilde{x}_i(t+1) < 1 \\ -1 & \text{für } \widetilde{x}_i(t+1) \leq -1 \end{cases} \tag{3.19}$$

3.2.1 Beschreibung des Explorationsalgorithmus'

Der Explorationsalgorithmus baut hier ein Netz

$$\mathcal{G} := (\mathbb{V}, \mathbb{E}, \sigma_{\boldsymbol{x}}) \tag{3.20}$$

aus RBF-Neuronen auf. Die Breite $\sigma_{\boldsymbol{x}} \in]0,1[$ jedes Neurons wird über das gesamte Netz konstant gehalten. Jedes Neuron

$$V_i := (z, \boldsymbol{w}, l, e) \in \mathbb{V} \tag{3.21}$$

besteht aus einer Feldaktivität z_{V_i}, einem Gewichtsvektor $\boldsymbol{w}_{V_i}$, einer Zählvariable l_{V_i} und einem Fehlerwert e_{V_i}. Jede Kante

$$E_{ij} := (V_j, V_i, w, \boldsymbol{m}, l, \alpha) \in \mathbb{E} \tag{3.22}$$

verfügt neben Start- und Endneuron über ein Verbindungsgewicht $w_{E_{ij}}$, eine Motoraktion $\boldsymbol{m}_{E_{ij}}$, eine Zählvariable $l_{E_{ij}}$ und ein Alter $\alpha_{E_{ij}}$. Diese Struktur unterscheidet sich von dem ursprünglichen GNG-U Algorithmus aus Abschnitt 2.4.2 ab Seite 20 im Nützlichkeitswert, der hier nicht vorhanden ist, der Zählvariable l und der Feldaktivität z. Die Zählvariable speichert die Anzahl an Aktualisierungen des Neurons der Kante. Anhand dieses Wertes wird die Lernrate für jedes einzelne Neuron beziehungsweise jede Kante separat berechnet.

Das entstehende neuronale Netz unterliegt einer Felddynamik, die sich aus den einzelnen Feldaktivitäten der Neuronen zusammensetzt. Die Änderung der Feldaktivität Δz_{V_i} für jedes einzelne Neuron V_i innerhalb eines Zeitschrittes ist:

$$\Delta z_{V_i}(t) = b_z + y_{V_i}(t) + \eta_z \sum_{V_j} \left(\gamma_{ij} w_{E_{ij}} - w_I\right) z_{V_j}(t), \; V_j \in \mathbb{V} \backslash \{V_i\} \tag{3.23}$$

wobei b_z eine Grundaktivierung der Feldaktivität und y_{V_i} der Ausgangswert des Neurons ist. Der letzte Term beschreibt eine laterale Beeinflussung der Feldaktivität über den Faktor $\eta_z \in]0,1[$ und das Skalarprodukt $\gamma_{ij} := \boldsymbol{m}_{E_{ij}}^T \boldsymbol{m}(t)$ des gespeicherten Motorvektors an der Kante von Neuron V_j nach V_i und des aktuellen Motorvektors. Weiterhin geht das Verbindungsgewicht $w_{E_{ij}} \in \{0,1\}$ ein. Existiert eine Verbindung, ist der Wert 1 ansonsten 0. Das Gewicht $w_I \in \mathbb{R}^+$ ist ein globales Gewicht, das die Feldaktivität hemmt. Der letzte Faktor ist die Feldaktivität des Neurons V_j, die in den Wertebereich von $[0,1]$, ähnlich zu Gleichung (3.19) auf der vorherigen Seite, projiziert wird. Diese Änderung führt nun in jedem Zeitschritt zu der Feldaktivität eines Neurons

$$z_{V_i}(t+1) = \tau_z z_{V_i}(t) + (1-\tau_z) \Delta z_{V_i}(t). \tag{3.24}$$

Diese beiden Gleichungen sind eine Konkretisierung der Gleichung (2.30) auf Seite 17. Wesentliche Eigenschaften dieser Variante sind die Faktoren γ_{ij} und z_{V_j}. Die laterale Interaktion ist demnach dort stark, wo die Feldaktivität einen großen Wert hat. Weiterhin ist die Richtung der Motoraktion entscheidend. Das Skalarprodukt der beiden Motorvektoren ist umso größer, je mehr die beiden Vektoren in die gleiche Richtung zeigen.

Die folgende Auflistung ist dem Algorithmus aus Abschnitt 2.4.2 ab Seite 20 nachempfunden. Die einzelnen Schritte wurden an den hier vorliegenden Algorithmus aus [Tou04] und [Tou06] angepasst. Einzig der zehnte Schritt wurde entfernt, da das neuronale Netz hier nicht alle Fehlerwerte global vergisst.

1. Initialisiere Graph $\mathcal{G}$ mit einem Knoten $V_0 := (\boldsymbol{x}(t_0), 1, 0)$ und $\mathbb{E} := \emptyset$.

2. Wähle eine Motoraktion $\boldsymbol{m}(t)$, führe diese aus und speichere $\boldsymbol{x}(t+1)$: Es ist immer genau ein m_i auf 1 gesetzt und alle anderen m_j, mit $i \neq j$ und $i, j \in \{1, \ldots, k\}$, haben den Wert 0. Mit einer geringen Wahrscheinlichkeit $p \in]0, 1[$ und $p \ll 1$ wird der aktive Index i ersetzt, ansonsten wird $\boldsymbol{m}(t)$ beibehalten.

3. Bestimme den Knoten $V_w(t+1)$ mit der größten Feldaktivität $z_{V_w}(t+1)$.

4. Existiert keine Kante zwischen $V_w(t)$ und $V_w(t+1)$, so füge die Kante

$$E := (V_w(t), V_w(t+1), 1, \boldsymbol{m}(t), 0) \tag{3.25}$$

   ein. Existiert die Kante E so setze deren Alter auf 0.
   Aktualisiere den Motorvektor jeder Kante E_{ij}, für die

$$z_{V_i}(t+1) > z_{V_i}(t) \text{ und } z_{V_j}(t+1) < z_{V_j}(t) \tag{3.26}$$

   gilt, mit

$$\boldsymbol{m}_{E_{ij}}(t+1) = \frac{l_{E_{ij}}}{l_{E_{ij}}+1}\boldsymbol{m}_{E_{ij}}(t) + \frac{1}{l_{E_{ij}}+1}\boldsymbol{m}(t+1) \tag{3.27}$$

   und inkrementiere jedes zugehörige $l_{E_{ij}}$ um 1.

5. Berechne die Fehlervariable von $V_w(t+1)$ mit

$$e_{V_w}(t+1) = \tau_e e_{V_w}(t) + (1-\tau_e)(1 - y_{V_w}(t+1)). \tag{3.28}$$

6. Aktualisiere den Gewichtsvektor von V_w mit

$$\boldsymbol{w}_{V_w}(t+1) = \frac{l_{V_w}}{l_{V_w}+1}\boldsymbol{w}_{V_w}(t) + \frac{1}{l_{V_w}+1}\boldsymbol{x}(t+1) \tag{3.29}$$

   und inkrementiere l_{V_w} um 1.

7. Inkrementiere das Alter aller Kanten um $\gamma_{ij} z_{V_j}$.

8. Entferne alle Kanten, deren Alter größer als das maximale Alter $\alpha_{max} \in \mathbb{R}^+$ ist. Sollten Knoten ohne zugehörige Kante vorhanden sein, werden diese ebenfalls entfernt.

9. Ist der Fehlerwert $e_{V_w}(t+1)$ größer als ein Grenzwert h, auch Vigilanz genannt, so wird ein neues Neuron $V_v := (\boldsymbol{x}(t+1), 1, 0)$ an die aktuelle Position eingefügt und der Fehlerwert von $V_w(t)$ auf 0 gesetzt.

10. Solange kein Abbruchkriterium erfüllt ist, gehe zurück zu 2.

Die Exploration beginnt an einem beliebigen Punkt $\boldsymbol{x}(t_0)$, der in einem RBF-Neuron V_0 mit dem Gewichtsvektor $\boldsymbol{w}_{V_0} := \boldsymbol{x}(t_0)$ abgespeichert wird. Nun werden kontinuierlich

die Motoren angesteuert und der Roboter bewegt sich von seinem ursprünglichen Zustand $\boldsymbol{x}(t_0)$ fort, was zu einer Verminderung der Aktivierung von V_0 führt. Es wird sich ein zweites Neuron V_1 bilden, sobald der Fehler e_{V_0} den Grenzwert h überschreitet. Der Abstand von V_0 und V_1 kann mit $\sigma_{\boldsymbol{x}}$ kontrolliert werden. Je größer dieser Wert, desto langsamer wächst der Fehlerwert und desto gröber wird der sensorische Raum eingeteilt. Dieser Ablauf führt schließlich zu einem Netz, wie es in Abbildung 3.5 dargestellt ist.

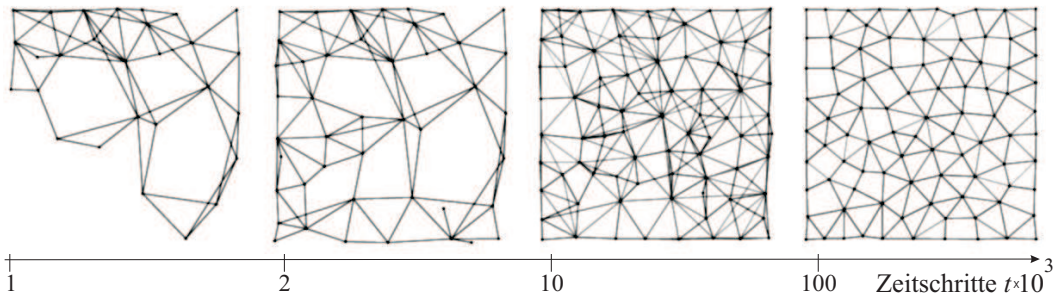

Abbildung 3.5: Entwicklung der Neuronenschicht auf einem ebenen Quadrat zu vier Zeitpunkten: Durch die zufällige Richtungsauswahl ergibt sich zunächst in den ersten drei Grafiken eine lückenhafte und nicht vollständige Struktur. Erst zuletzt stellt sich eine regelmäßige Struktur ein (vgl. [Tou06, S. 8]).

Es ist der Aufbau des neuronalen Netzes des sensorischen Raumes über die Zeit hinweg abgebildet. Die erste Grafik zeigt deutlich den Auswahlmechanismus der nächsten Motoraktion. Es gibt längere gerade Abschnitte, die auf keinen Wechsel in der Bewegungsrichtung hindeuten. In der dritten Grafik weist das Netz viele Kanten auf, die sich schneiden. Da die beiden Neuronen einer solchen Kante nicht mehr direkt hintereinander aktiviert werden, werden diese Kanten immer älter und verschwinden schließlich. Dadurch ergibt sich schließlich eine regelmäßige Struktur.

3.2.2 Beschreibung des Planungsalgorithmus'

Das neuronale Netz kategorisiert den sensorischen Raum und verknüpft diesen über die Kanten mit Motoraktionen. Eine Planung von Aktionen kann durch externe Vorgaben von Sensorwerten durchgeführt werden. Diese werden in [Tou04] als „goal stimulus" $\boldsymbol{g}$ bezeichnet. Die Veränderungen im Ablauf des Algorithmus sind in Abbildung 3.6 auf der nächsten Seite dargestellt.

Um das Ziel $\boldsymbol{g}$ zu erreichen, wird die ursprüngliche Motorsteuerung deaktiviert. Stattdessen werden die neuen Motoraktionen aus dem neuronalen Feld generiert. Dafür muss die Information des Weges über das Netz verteilt werden. Diese Verteilung wird über einen bestärkenden Lernalgorithmus umgesetzt. Als Grundlage dient die Bellmann-Gleichung

$$V_\pi^*(i) = \sum_a \pi(a \mid i) \sum_j P(j \mid i, a)[r(j) + \gamma V_\pi^*(j)] \tag{3.30}$$

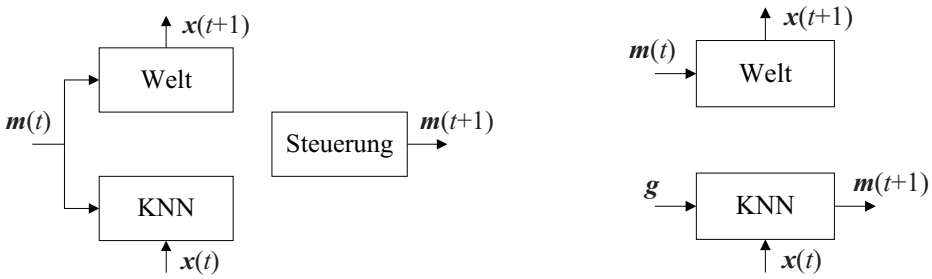

Abbildung 3.6: Flussdiagramme des Algorithmus: (links) Exploration: Aus dem aktuellen Sensor- und Motorvektor wird das künstliche neuronale Netz aufgebaut. Die Welt stellt einen neuen Sensorvektor und die Steuerung einen neuen Motorvektor bereit. (rechts) Planung: Durch die Vorgabe des Zieles $\boldsymbol{g}$ fällt die Steuerung weg und das Netz wird zu einem Regler, der die neuen Motoraktionen berechnet.

und deren algorithmische Approximation aus [SB98]. Eine detaillierte Betrachtung der Anwendbarkeit und Anpassung ist in [Tou04] und [Tou06] unter „The dynamics of planning“ zu finden.

Der aktuelle Einzelgewinn r_{V_i} eines Neurons V_i berechnet sich nach Gleichung (2.20) auf Seite 13 und Gleichung (2.21) auf Seite 13 mit dem Ziel $\boldsymbol{g}$ als sensorischen Eingangswert:

$$r_{V_i} := \exp^{-2\|\boldsymbol{g}-\boldsymbol{w}_{V_i}\|^2 \sigma_{\boldsymbol{x}}^{-}{}^2}. \tag{3.31}$$

Der Gesamtgewinn eines Neurons V_i ist

$$v_{V_i}(t+1) = -\tau_v v_{V_i}(t) + (1-\tau_v)(r_{V_i} + \eta_v \max_{V_j}\{w_{E_{ji}} v_{V_j}(t)\}),\ V_j \in \mathbb{V}_{V_i} \tag{3.32}$$

und beschreibt den aktuellen und den maximalen zukünftigen Gewinn eines Neurons V_i. Der Gewinn der Zukunft wird durch die gierige Strategie über den letzten Term umgesetzt, wobei $\eta_v \in]0,1[$ ist. Hier wird der maximale Gewinn eines Nachbarneurons v_{V_j} mit dem Gewicht $w_{E_{ji}}$ der Verbindung der beiden Neuronen V_i und V_j additiv zum Gesamtgewinn hinzugefügt. Die Gewichte können kontinuierlich definiert sein, jedoch zeigt [Tou06] dass der Wert 0 für keine und der Wert 1 für eine Verbindung genügen. Das Ziel $\boldsymbol{g}$ wird erreicht, wenn die Motoraktionen durch

$$\boldsymbol{m}(t) = c_1 \sum_{E_{ji}} z_{V_i}(t)\, w_{E_{ji}} (v_{V_j}(t) - v_{V_i}(t)) \boldsymbol{m}_{E_{ji}},\ E_{ji} \in \mathbb{E} \tag{3.33}$$

so gewählt werden, dass durch die passende Auswahl von $c_1 \in \mathbb{R}$ gilt: $\|\boldsymbol{m}(t)\| = 1$.

3.2.3 Kategorisierung

Wird ein Weltmodell benötigt?

Es wird kein vorher definiertes Weltmodell benötigt. Jedoch kann das erstellte Netz als Weltmodell verstanden werden, in der sogar eine gewisse Vorhersage möglich ist. In [Tou06] wird die Antizipation des Systems untersucht. Es zeigt sich, dass sich durch die Einkopplung der Motoraktionen in die Netzstruktur eine Vorhersage der nahen

Zukunft realisieren lässt. Basierend auf dieser Antizipation wurde zum Beispiel eine Hindernisvermeidung implementiert.

Wie werden die Explorationsergebnisse repräsentiert?

Die Explorationsergebnisse werden in einem künstlichen neuronalen Netz abgespeichert, deren Kanten die Motoraktion beinhalten. Es wird also, wie beim vorherigen Algorithmus, der sensorische Raum kartografiert und mit einzelnen Elementen aus der Menge der Motoraktionen punktweise verknüpft.

Wie wird die Exploration gesteuert und kann die Exploration gezielt beeinflusst werden?

Die Exploration wird direkt durch einen einfachen Auswahlmechanismus der Richtung der Motoraktion gesteuert. Durch die Auswahl keiner Richtung könnte ein Verhalten wie beim vorherigen Explorationsalgorithmus erzielt werden. Bei der Auswahl werden keine Informationen aus dem aktuell vorhandenen Netz verwendet. Wird das bereits erlangte Wissen verwendet, so muss das Problem der Ausbeutung gegenüber dem Erforschen betrachtet werden. Hier wird auf [ES02] verwiesen, wo unter anderem dieses Problem erläutert wird.

Außerdem vergisst die Neuronenschicht kontinuierlich durch das ansteigende Alter der Kanten. Aufgrund dessen muss bestehendes Wissen regelmäßig aktualisiert werden, um es nicht vollständig zu vergessen. Nur in den Zwischenphasen kann neues Wissen erlangt werden. Wird demnach vorhandenes Wissen verwendet, um die Exploration in bestimmte Bereiche des Phasenraumes zu führen, so muss dieser Aspekt bei der Koordination der Motoraktionen berücksichtigt werden.

Kann die Exploration unbegrenzt fortgesetzt werden?

Die Exploration kann unbegrenzt fortgeführt werden, jedoch wird das erlernte Wissen eine Obergrenze erreichen. Diese Obergrenze ist durch das Kantenalter vorgegeben, da der Roboter dann nicht mehr alle Kanten rechtzeitig vor ihrer Löschung aktualisieren kann. Der Algorithmus hat keine Speicher- oder Effizienzprobleme, solange das Alter nicht zu hoch und die Granularität der Kategorisierung nicht zu fein gewählt sind. Ist dies nicht der Fall, so können nicht mehr alle RBF-Neuronen innerhalb des 10 ms Zeitfensters der M-Serie berechnet werden.

Wie kann auf die Explorationsergebnisse zugegriffen werden?

Auf die Explorationsergebnisse kann direkt über eine Planung im sensorischen Raum zugegriffen werden, beispielsweise über die Keyframe-Methode. Bei dieser Methode werden Punkte im sensorischen Raum vorgegeben, die in einer bestimmten Reihenfolge zu erreichen sind.

Kann sich das System an eine wandelnde Umwelt anpassen?

das entstehende neuronale Netz kann sich langsam an eine wandelnde Welt anpassen. Abbildung 3.5 auf Seite 37 verleiht einen Eindruck der Reaktivität des Systems. Bis ein ebenes Quadrat vollständig und annähernd optimal kartografiert ist, werden 100000 Zeitschritte benötigt. Bei einer Aktualisierungsfrequenz von 100 Hz sind das mehr als 16 Minuten. Diese Dauer wird maßgeblich von der Auswahl der Motoraktion verursacht. Eine gezielte Steuerung in sensorisch unbekannte Bereiche könnte die Explorationszeit verkürzen. Im Allgemeinen ist das System zeitlich durch die Wahl des Kantenalters beschränkt. Je kürzer eine Kante erhalten bleibt, desto schneller kann es sich einer sich wandelnden Umwelt anpassen. Jedoch vergisst es dadurch auch sehr schnell und die Verbindungen müssen ständig aufgefrischt werden.

Wie hoch ist die Komplexität des Algorithmus'?

Die vorgestellten Teilalgorithmen für die Exploration und die Planung sind durch Gleichung (3.24) auf Seite 35 und Gleichung (3.32) auf Seite 38 der Komplexitätsklasse $O = (nm)$ mit $n = |\mathbb{V}|$ und $m = |\mathbb{E}|$ zuzuordnen. Die Planungsphase ist ebenfalls dieser Komplexitätsklasse zuzuordnen, da bei der Verteilung des Zieles $\boldsymbol{g}$ ebenfalls zu jedem Neuron alle Kanten betrachtet werden.

Ist der Algorithmus neuronal implementierbar?

Der Algorithmus ist neuronal implementierbar. Die Grundstruktur besteht aus einer Neuronenschicht und auch die Planung kann neuronal umgesetzt werden. Hierfür müssen das RBF-Neuron und die Kantenstruktur erweitert werden, so dass mehrere Werte speichern kann.

3.2.4 Zusammenfassung

Dieser Explorationsalgorithmus verwendet ein künstliches neuronales Feld aus RBF-Neuronen zur Kategorisierung. Die kontinuierliche Entwicklung von statischen Netzen über SOM zu GNG wird weitergeführt und verbindet die wachsende Struktur mit Motoraktionen. Dadurch wird die Information der Navigation im sensorischen Raum gewonnen.

Diese Erweiterungen führen zu einer Reihe von Parametern, mit denen dieser Algorithmus eingestellt werden kann. Zunächst seien hier τ_e, die Breite σ_x und die Vigilanz h genannt. Diese Parameter haben einen direkten Einfluss auf die Dichte des neuronalen Netzes. Je kleiner die Werte sind, desto feiner wird der sensorische Raum kategorisiert. Die Parameter $\boldsymbol{b}_z$, η_z, τ_z und w_I bestimmen durch Gleichung (3.23) auf Seite 35 und Gleichung (3.23) auf Seite 35 die Dynamik der Feldaktivität. Abschließend beeinflusst das maximale Kantenalter α_{max} das Erhalten selten oder gar nicht genutzter Kanten. In Abschnitt 5.2 ab Seite 72 werden diese und weitere Parameter einer genaueren Untersuchung unterzogen.

Es stellt sich nun die Frage: Wie kann dieses Wissen verwendet werden? In [Tou04] und [Tou06] werden einfache Anwendungen mit der Navigation in einem Labyrinth vorgestellt. Bei der Navigation werden dabei die Motoraktionen kontinuierlich ineinander überführt. Eine konsequente Weiterentwicklung wäre beispielsweise eine Rundreise durch ein Labyrinth, bei dem kontinuierlich die Ziele angepasst werden.

Wie könnte eine Anpassung auf die Roboter der M-Serie aussehen? Zunächst könnten hier nur absolute Sensoren verwendet werden. In [Tou06] werden Abstandssensoren in einem Labyrinth verwendet, die den globalen Zustand nicht unterscheiden können. Ähnlich verhält es sich mit den Strom- oder Beschleunigungssensoren der M-Serie Roboter. Diese Informationen können zur Kartografierung nicht verwendet werden. Jedoch wäre es möglich die einzelnen Kanten mit der verbrauchten Energie zu versehen. Während der Planungsphase könnte dann in Gleichung (3.32) auf Seite 38 diese Energie mit eingehen. Somit könnten jeweils die sparsamsten Kanten priorisiert werden. Da bei der Planungsphase immer sensorische Abbilder verwendet werden, ist die Keyframe-Technik verwendbar, um Bewegungsmuster zu erstellen. So kann eine bestimmte Abfolge an Posen aufgenommen und anschließend abgespielt werden.

Eine Anpassung dieses Algorithmus auf den Vorgänger der M-Serie, der A-Serie, ist in [Ste10] umgesetzt. Hier wurde der Rumpf der A-Serie fixiert, so dass sich nur der Arm bewegen konnte. Weiterhin wurden einige Hindernisse im Bewegungsraum des Armes eingefügt. Hier konnten die Ergebnisse aus [Tou04] und [Tou06] bestätigt werden.

Einige Probleme der Bewegungssteuerung zeigen sich auf Videos[1] innerhalb eines Labyrinths. Hier werden zufällig Positionen vorgegeben. Sobald das Ziel erreicht ist, wird ein neues vorgegeben. Auf den Videos ist sehr schön die Verteilung des Gewinnes von einem Ziel aus erkennbar. Das Problem ist jedoch die zeitliche Verschiebung, bis an dem Ort, an dem sich der Roboter gerade befindet, die neue Information der Richtung verfügbar ist. Unterdessen bewegt er sich noch nach den alten Gewinnwerten unter Umständen in eine komplett falsche Richtung. Weiterhin sieht es so aus, als ob der Roboter zu starke Motorsignale empfängt, da er sehr schnell an die Wände des Labyrinths anstößt. Dies ist auf die Normierung der Motoraktion auf 1 in Gleichung (3.33) auf Seite 38 zurückzuführen. Würde die Bewegung auf einen Roboter der M-Serie übertragen, so wären die Wände des Labyrinths beispielsweise die Anschläge der Gelenke. Durch dieses Verhalten würden diese außerordentlich belastet und schneller verschleißen. Speziell die zweite Aufnahme zeigt eine Variante mit Hindernisvermeidung. Hier werden die Wände gemieden, jedoch ist der Roboter dann auch entsprechend langsamer.

Allgemein haftet dieser Explorationsmethode an, dass das Wissen nur über gezielte Ansteuerung von Sensorwerten genutzt werden kann. Dadurch ist die Verwendung von Keyframe-Steuerungen möglich, jedoch werden keine komplexen Bewegungen von selbst erlernt und angeboten. Ein weiterer Mangel ist die Obergrenze des erlernbaren Wissens durch die Altersgrenze der Kanten, das ebenfalls Einfluss auf die Reaktivität des Systems hat. Ist der sensorische Raum so groß, dass nicht mehr alle Kanten rechtzeitig vor ihrem Löschen aktualisiert werden können, so geht zwangsläufig Wissen verloren.

[1] Siehe „path planning through a maze and inducing the corresponding motor sequences“ und „planning with obstacle avoidance enabled“ unter [Tou07]

3.3 Intrinsisch motivierte Exploration

In [OK07] wird ein umfangreicher Überblick über Explorationsmethoden durch intrinsische Motivation gegeben. Die dort vorgestellten Algorithmen werden durch das Vorhandensein eines Prädiktors, einer Fehlerfunktion des Systems und der Bestimmung der Motivation unterteilt. Weiterhin wird in [OKH07], basierend auf [OKHW05], ein intrinsisches Motivationssystem detailliert vorgestellt. Basierend auf dieser Grundlage wird parallel zu dieser Arbeit in der Arbeitsgruppe dieser Algorithmus für die M-Serie angepasst und erweitert.

3.3.1 Beschreibung des Algorithmus'

Als Grundlage für die Exploration dient das erste Experiment aus [OKH07]. Hier befindet sich ein Roboter mit zwei Motoren für ein rechtes und ein linkes Rad auf einer ebenen unbegrenzten Fläche. Neben der Motoraktion $\boldsymbol{m}(t) \in [-1,1]^2$ hat der Roboter weiterhin die Möglichkeit einen Ton mit einer Frequenz $f(t) \in [0,1]$ zu erzeugen. Diese drei Werte werden zum Aktionsvektor $\boldsymbol{y}(t)$ zusammengefasst. Die Werte des Aktionsvektors werden auf zwei Nachkommastellen gerundet. Auf der Ebene befindet sich weiterhin ein Spielzeug, das abhängig von $f(t)$ folgende Bewegungen ausführt:

- Zufällige Bewegung wenn $f(t) \in f_1 = [0, 0.33]$
- Keine Bewegung wenn $f(t) \in f_2 = [0.34, 0.66]$
- Sprung zur Roboterposition wenn $f(t) \in f_3 = [0.67, 1]$

Der eindimensionale Sensorvektor $\boldsymbol{x}(t) \in \mathbb{R}^1$ gibt den Abstand zwischen Roboter und Spielzeug an.

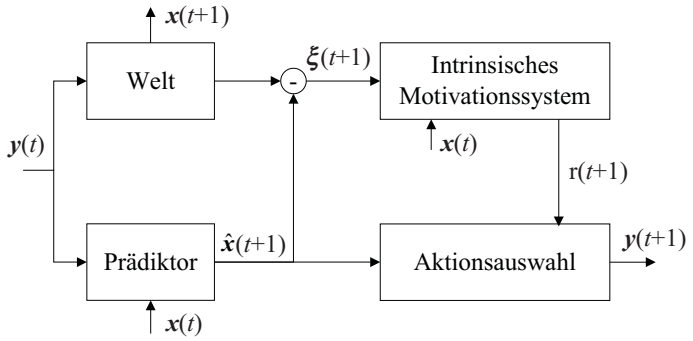

Abbildung 3.7: Basisarchitektur prädiktiver intrinsischer Motivationssysteme: Der Prädiktor erhält den Sensorvektor $\boldsymbol{x}(t)$ und den Aktionsvektor $\boldsymbol{y}(t)$. Die Prädiktion $\hat{\boldsymbol{x}}(t+1)$ führt im Vergleich zum realen Sensorvektor $\boldsymbol{x}(t+1)$ zum Fehler $\boldsymbol{\xi}(t+1)$, den das Motivationssystem mit Sensor- und Aktionsvektor zur Berechnung einer Belohnung $r(t+1)$ verwendet. Aus dieser Belohnung und der Prädiktion wird abschließend eine neue Aktion generiert (vgl. [OK07, S. 7]).

Die Vektoren $\boldsymbol{x}(t)$ und $\boldsymbol{y}(t)$ sind Eingangssignale in das System, das in Abbildung 3.7 dargestellt ist. Ein Prädiktor schätzt den nächsten Abstand zum Spielzeug $\hat{\boldsymbol{x}}(t+1)$.

Der Prädiktor besteht hier aus mehreren Experten, die sich auf unterschiedliche Bereiche des Phasenraumes spezialisieren. Ein Experte könnte zum Beispiel ein einfaches Feed-Forward Netzwerk sein. Zunächst existiert nur ein Experte, dem jedes Vektortripel $(\boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{x}(t+1))$ zugeordnet wird. Diese Tripel dienen als Grundlage für die Anpassung der Vorhersage. Wird ein bestimmtes Kriterium erreicht, so teilt sich der Experte in zwei disjunkte Experten auf. Dies geschieht durch Trennung der gespeicherten Vektortripel in einer Dimension anhand eines Grenzwertes. Da in jedem Zeitschritt nur ein Experte eine Vorhersage trifft, kann anhand der Grenzwerte einfach bestimmt werden, welcher Experte aktuell am besten passt.

Der entstehende Fehler $\boldsymbol{\xi}(t+1)$ wird verwendet um eine Belohnung $r(t+1)$ zu berechnen. Diese Belohnung richtet sich nach der Zu- oder Abnahme des Fehlers. Eine Möglichkeit wäre, dass die Belohnung proportional zum Fehler abnimmt. Die Belohnung wird schließlich verwendet um eine Aktion zu generieren. Hierfür wird ein bestärkendes Lernverfahren, das Q-Learning, verwendet. Einen guten Einblick in die allgemeine Funktionsweise gibt [Wat89] und in [DW01] ist die technische Realisierung kompakt dargestellt.

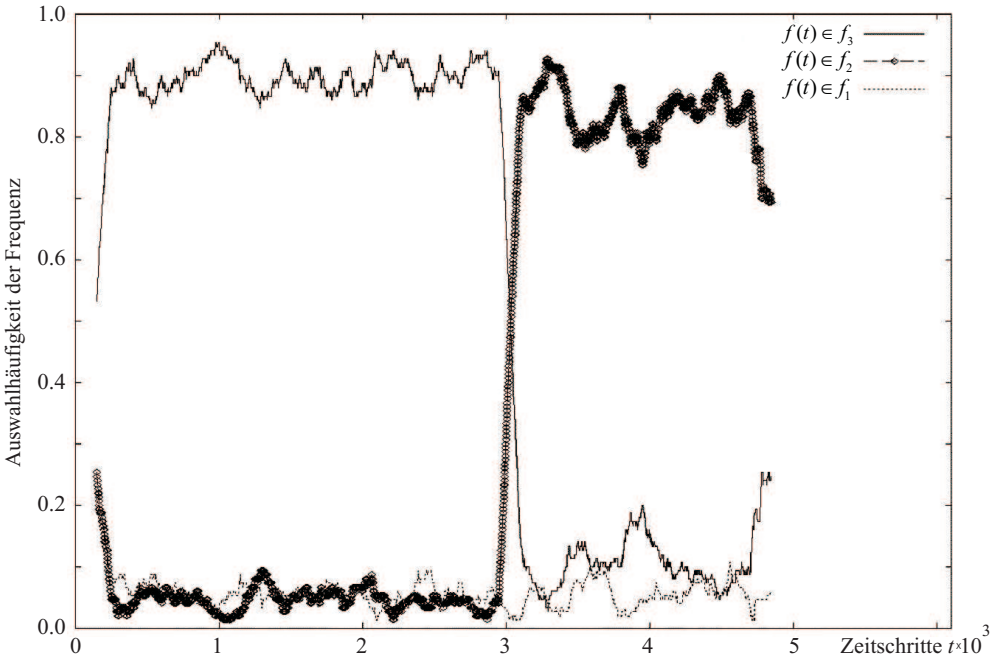

Abbildung 3.8: Auswahlhäufigkeit der Frequenz: Es sind drei Bereiche erkennbar. Während der ersten 250 Zeitschritte wird jede Frequenz annähernd gleich oft ausgewählt. Bis zum Zeitschritt 3000 werden hauptsächlich Frequenzen aus f_3 und danach fast ausschließlich Frequenzen aus f_2 ausgewählt (vgl. [OKH07, S. 10]).

Zurückgreifend auf den Beginn der Exploration werden der Roboter und das Spielzeug zufällig auf der Ebene postiert und die diskreten Aktionen mit einer initialen Auswahlwahrscheinlichkeit für das Q-Learning belegt. Da ohne eine Aktion nichts gelernt werden kann, führt der Q-Learning Algorithmus dazu, dass der Roboter aktiv wird. Eine Übersicht über die ausgewählten Aktionen im Hinblick auf die Frequenz $f(t)$ ist in Abbildung 3.8 dargestellt.

Durch die Initialisierung der Aktionen mit Zufallswerten werden bis Zeitschritt 250 alle drei Frequenzen recht gleichmäßig ausgewählt. Da sich die Auswirkungen des Spielzeugs innerhalb der Frequenz f_3 am einfachsten voraussagen lassen und am schnellsten gelernt werden können, werden diese Frequenzen zunächst bevorzugt. Nach ca. 3000 Zeitschritten wird jedoch kaum mehr etwas Neues gelernt, was zu sinkenden Belohnungen und anderen Aktionen führt. Somit wird anschließend meistens eine Frequenz aus f_2 gewählt, da der resultierende Abstand zum Spielzeug schwerer, im Vergleich zu Frequenzen aus f_1, jedoch noch vorhersagbar ist.

3.3.2 Kategorisierung

Wird ein Weltmodell benötigt?

Es wird ein Weltmodell benötigt, da eine Prädiktion erfolgt, jedoch wird in [OKH07] explizit und mehrmals darauf hingewiesen, dass der Roboter beziehungsweise das System keine Vorkenntnisse benötigen. So sind die Aktionen des Roboters in dem Experiment bereits um Töne erweitert. In einem weiteren Experiment ist zum Beispiel eine Stoßbewegung zu einer Motordimension zusammengefasst worden. All diese Aktionen sind dem System nicht vorgegeben, sondern die Experten passen sich selbst an die Umgebung an.

Wie werden die Explorationsergebnisse repräsentiert?

Die Explorationsergebnisse liegen in Form von Experten vor, die den Phasenraum kategorisieren. Jeder Experte kann innerhalb eines bestimmten Bereiches Vorhersagen über die Folgezustände treffen. Ein Experte kann beispielsweise ein Feed-Forward-Netz oder ein rekurrentes Netz sein.

Wie wird die Exploration gesteuert und kann die Exploration gezielt beeinflusst werden?

Die Exploration wird durch die Auswahl des Q-Algorithmus gesteuert. Indirekt können hier die Auswahlwahrscheinlichkeiten beeinflusst werden. Ein Kriterium könnte hier zum Beispiel der Energieverbrauch sein. Eine andere Stellgröße ist die Belohnungsfunktion, in die verschiedenste Kriterien integriert werden können. Jede Veränderung hat jedoch immer nur indirekten Einfluss auf die ausgeführten Motoraktionen.

Kann die Exploration unbegrenzt fortgesetzt werden?

Hier zeigt sich eine Schwäche des vorgestellten Algorithmus, da dieser jedes Vektortripel speichert. Somit wird zwangsläufig jedes Speicherlimit überschritten. In der Arbeitsgruppe wird der Algorithmus an dieser Stelle verändert. Hier wird für die Anpassung der Experten ein Gradientenabstieg verwendet. Somit ist die Vergangenheit der Vektoren nicht mehr notwendig. Allerdings ist dann auch die Auswahl des Experten über

die Grenzwertkriterien nicht mehr möglich. Dadurch werden alle Experten parallel berechnet und derjenige mit dem geringsten Fehler für den Lernprozess verwendet. Mit dieser Anpassung ist auch hier ein unbegrenztes Lernen unter Beachtung der Anzahl der Experten möglich. Je nach Rechenleistung ist es ab einer bestimmten Anzahl an Experten nicht mehr möglich die 100 Hz Taktfrequenz zu erreichen.

Wie kann auf die Explorationsergebnisse zugegriffen werden?

Der derzeitige Algorithmus bietet keine Möglichkeit auf den erworbenen Informationen eine Planung durchzuführen. Bereits einfache Aufgaben wie:

- Stelle eine bestimmte Distanz zum Spielzeug her!
- Bewege das Spielzeug von Position A zu Position B!

können nicht gelöst werden.

Kann sich das System an eine wandelnde Umwelt anpassen?

Die Variabilität des Systems ist hier schwer zu beurteilen, da es viele Subkomponenten gibt, die einen Einfluss haben. Wird Abbildung 3.8 auf Seite 43 erneut betrachtet, so kann davon ausgegangen werden, dass nach 5000 Zeitschritten ein Großteil des Szenarios erlernt wurde, da sich die Häufigkeiten von f_2 und f_3 wieder nähern. Bei einer Aktualisierungsfrequenz von 100 Hz ergibt das weniger als eine Minute. Zusammenfassend kann festgehalten werden, dass die Welt zügig erforscht wird, jedoch kann auf eine Verschiebung des Frequenzbereiches nicht reagiert werden.

Wie hoch ist die Komplexität des Algorithmus'?

Der Algorithmus ist insgesamt mindestens der Komplexitätsklasse $O(nm)$ zuzuordnen. Die Prädiktion durch zum Beispiel neuronale Netze kann in $O(n)$, wobei n die Anzahl an Experten ist, durchgeführt werden. Das verwendete Q-Learning und andere Subalgorithmen sind jedoch meistens der Komplexitätsklasse $O(nm)$ zuzuordnen, wobei n die Anzahl an Experten und m die Anzahl an möglichen Motoraktionen sind. In [OKH07] wurde versucht die weiteren Subalgorithmen, beispielsweise die Trennung der Experten, durch einfache Strategien in $O(n)$ zu überführen.

Ist der Algorithmus neuronal implementierbar?

Wie bereits erwähnt sind der Experte selbst und dessen Auswahl neuronal umsetzbar. Eine Überführung des diskreten Q-Learning zu einem kontinuierlichen neuronalen Q-Learning wird in [HK03] oder [CRGU02] vorgestellt.

3.3.3 Zusammenfassung

Die vorgestellte Explorationsmethode unter Verwendung der intrinsischen Motivation greift auf den bestärkenden Lernalgorithmus Q-Learning zurück. Dadurch können nach [OKH07] nicht nur zusammenhängende Aktionen, wie zum Beispiel die Navigation in einem Labyrinth, sondern auch unabhängige Aktionen erlernt werden. So ist es durch die unterschiedlichen Experten beispielsweise möglich sowohl mit einer Hand etwas zu greifen, als auch im nächsten Augenblick mit den Augen einen Gegenstand zu fokussieren.

Dem aktuell vorgestellten Algorithmus haftet das Problem der Diskretisierung der Aktionen an. Die Auswahl der Aktionen wird stark beschnitten und es gehen unter Umständen wichtige Zustände im Phasenraum verloren. Dies könnte durch eine Anpassung des Algorithmus nach [HK03] oder [CRGU02] für das Q-Learning gelöst werden. Einen Überblick über weitere Methoden das Q-Learning zu implementieren gibt [GWZZ99]. Dort wird auf [GSK98] verwiesen, wo das Q-Learning über ein neuronales Feld, ähnlich zu dem vorher vorgestellten Algorithmus, implementiert wurde.

Die größere Problematik ist somit das planerische Nutzen des Wissens. Wie kann aus den Experten eine bestimmte Aktion, beispielsweise der erwähnte Abstand zum Spielzeug, generiert werden? Zwar besitzen die Experten die Möglichkeit die Folgezustände gut vorherzusagen, jedoch ist mit diesen keine Möglichkeit der Planung gegeben. Auch andere Kontrollstrukturen über sensomotorische Schleifen sind nicht zu erkennen.

3.4 Homöokinetisch getriebene Exploration

Die Homöokinese beschreibt im Gegensatz zur Homöostase keinen statischen, sondern einen kinetischen Zustand. Ziel der Homöokinese ist es also den Körper in einer ständigen Bewegung zu halten. Basierend auf diesem Prinzip wird in [DSP99] ein System vorgestellt, welches vollständig ohne externe Vorgaben beginnt Verhaltensweisen zu erforschen. Diese Idee wird seitdem erweitert und kann unter anderem in [Der99], [DL02], [DHL04], [DHM05], [DHM06], [DM06], [DMH06], [HDH07], [MHD07], [MFH08], [HMDH09] und [HDH09] nachvollzogen werden.Anhand der letzten beiden Veröffentlichungen werden im Folgenden der Algorithmus und interessante Erweiterungen erläutert.

Für die Beschreibung des Algorithmus wird auf das Beispiel der „skidding snake“ aus [HMDH09] zurückgegriffen. Diese Schlange, siehe Abbildung 3.9 auf der nächsten Seite, besteht aus n Kugeln, die über gleichlaufende Kugeldrehgelenke verbunden sind. Diese erlauben den Kugeln sich in zwei Freiheitsgraden zu drehen. Ein Freiheitsgrad ist durch jeweils eine waagerechte Achse von einem Mittelpunkt eines Gelenkes zum nächsten festgelegt. Die senkrechten Achsen durch die Gelenke sind der zweite Freiheitsgrad. Einzig der Kopf (grau) kann aktiv bewegt werden, jedoch reicht die Kraft nicht aus, um die Schlange komplett zu ziehen. Nur durch gezielte Ansteuerung ist es möglich in eine Rotation um den Mittelpunkt der Schlange überzugehen.

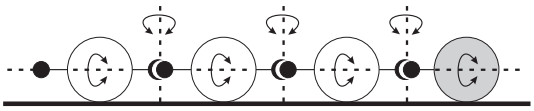

Abbildung 3.9: Aufbau der „skidding snake“: Die Schlange besteht aus n Kugeln, die über gleichlaufende Kugeldrehgelenke verbunden sind. Diese erlauben den Kugeln sich in zwei Freiheitsgraden zu drehen. Die Drehrichtungen sind durch die gestrichelten Achsen und die Pfeile angedeutet (vgl. [HMDH09, S. 6]).

Die Schlange bewegt sich auf einer ebenen Fläche und beschreibt ein zweidimensionales dynamisches System. Die aktuelle Geschwindigkeit des Kopfes $\boldsymbol{x}(t) \in [-1,1]^2$ ist der eingehende Sensorvektor und mit $\boldsymbol{y}(t) \in [-1,1]^2$ kann der Kopf der Schlange in jede beliebige Richtung gesteuert werden.

3.4.1 Beschreibung des Algorithmus'

Der Ablauf des Regelalgorithmus' ist in Abbildung 3.10 dargestellt.

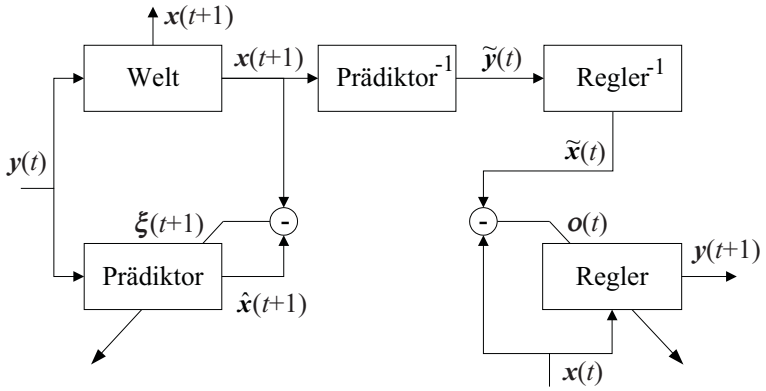

Abbildung 3.10: Homöokinetische Regelung: Die Parameter des Prädiktors beziehungsweise des Reglers werden angepasst, um die Differenz zwischen dem Sensorvektor $\boldsymbol{x}(t+1)$ und der Prädiktion $\tilde{\boldsymbol{x}}(t+1)$ beziehungsweise dem ursprünglichen Sensorvektor $\boldsymbol{x}(t)$ und dem erwarteten Sensorvektor $\hat{\boldsymbol{x}}(t)$ zu minimieren. Der invertierte Prädiktor und Regler berechnen ihre Parameter in jedem Zeitschritt aus den aktuellen Parametern der jeweiligen nicht invertierten Einheit (vgl. [MFH08, S. 2]).

Die Ansteuerung zum Zeitpunkt t berechnet sich durch das Feed-Forward Netzwerk

$$\boldsymbol{y}(t) := \tanh\left(\boldsymbol{W}_c \cdot \boldsymbol{x}(t) + \boldsymbol{b}_c\right) \tag{3.34}$$

mit der Gewichtsmatrix $\boldsymbol{W}_c \in \mathbb{R}^{2x2}$ und der Grundaktivierung $\boldsymbol{b}_c \in \mathbb{R}^2$. Ein weiteres Feed-Forward Netzwerk ohne Grundaktivierung ist der Prädiktor der Umwelt

$$\hat{\boldsymbol{x}}(t+1) = \boldsymbol{W}_a \boldsymbol{y}(t), \tag{3.35}$$

der den nächsten Zustand des Systems schätzt. Der entstehende Fehler

$$\boldsymbol{\xi}(t+1) = \boldsymbol{x}(t+1) - \hat{\boldsymbol{x}}(t+1) \tag{3.36}$$

wird verwendet, um den Prädiktor über einen Gradientenabstieg mit

$$\Delta \boldsymbol{W}_a (t+1) = \epsilon_a \boldsymbol{\xi} (t+1) \boldsymbol{y}^T (t) \tag{3.37}$$

anzupassen. Die Lernrate $\epsilon_a \in]0,1[$ ist groß genug, so dass der Prädiktor sich schnell an die aktuelle Situation der Umwelt anpassen kann. Dadurch wird die Einfachheit des Prädiktors kompensiert.

Würde dieser Algorithmus gestartet werden, so würden die Parameter $\boldsymbol{W}_c$, $\boldsymbol{b}_c$ und $\boldsymbol{W}_a$ gegen Null gehen, da hier der Fehler $\boldsymbol{\xi}(t)$ immer am kleinsten ist. Dies wäre ein homöostatisches Verhalten.

Um einen homöokinetischen Zustand zu erreichen, wird der stabile Zustand destabilisiert. Dies wird durch eine Umkehrung der Zeitskala erreicht. Diese Eigenschaft und das Vorgehen wurden bereits in Abschnitt 2.1 ab Seite 6 und Abschnitt 2.4.1 ab Seite 18 erläutert. Es wird also zu dem aktuellen Sensorvektor $\boldsymbol{x}(t+1)$ der Sensorvektor $\tilde{\boldsymbol{x}}(t)$ ermittelt, der notwendig gewesen wäre, um den wirklichen Folgezustand zu erreichen. Die Umkehrung dieses Systems wird über folgende Approximation mit der Pseudoinversen der Jacobi-Matrix $\boldsymbol{L}(\boldsymbol{x}(t+1))^+$ durchgeführt:

$$f_e(\boldsymbol{x}, \hat{\boldsymbol{x}}) := (\boldsymbol{L}(\boldsymbol{x})^+ \boldsymbol{xi})^T (\boldsymbol{L}(\boldsymbol{x})^+ \boldsymbol{xi}) \tag{3.38}$$

Zur besseren Lesbarkeit wurde hier die Bezeichnung $(t+1)$ weggelassen. Über den Gradientenabstieg

$$\Delta \boldsymbol{W}_c (t+1) = \epsilon_c \frac{\delta}{\delta \boldsymbol{W}_c} f_e (\boldsymbol{x}(t+1), \hat{\boldsymbol{x}}(t+1)) \tag{3.39}$$

mit der Lernrate $\epsilon_c \in]0,1[$ und $\epsilon_c < \epsilon_a$, können die Parameter des Reglers angepasst werden. Genauere Betrachtungen der Herleitungen und die explizite Bestimmung von $\boldsymbol{L}(\boldsymbol{x}(t+1))$ und $\Delta \boldsymbol{W}_c (t+1)$ sind unter anderem in [HDH09] zu finden.

Zu welchen Ergebnissen führt diese Ansteuerung nun bei der „skidding snake“? In Abbildung 3.11 auf der vorherigen Seite sind die Hüllkurven für $x_1(t)$ und $y_1(t)$ dargestellt. Deutlich sind die ansteigenden Motorwerte bis zum Zeitschritt 18000 zu erkennen. Dieser Anstieg resultiert aus der Vorhersagbarkeit der schwächeren Motoraktionen. Der dadurch resultierende Fehler ist sehr klein und führt umgekehrt zu einer großen Destabilisierung entgegen der Zeitskala. Schließlich wird der rotierende Zustand erreicht, der zunächst schwer vorhersagbar ist. Somit pendeln sich die Motorwerte etwas niedriger ein.

Dieses Verhalten kann sehr gut an den Parametern $\boldsymbol{W}_c$ und $\boldsymbol{W}_a$, dargestellt in Abbildung 3.12, erklärt werden. Hier steigt die Sensitivität des Reglers bis zum Zeitschritt 18000, was zu den erhöhten Motoraktionen $\boldsymbol{y}(t)$ führt. Da sich die Geschwindigkeit $\boldsymbol{x}(t)$ jedoch kaum ändert, muss der Prädiktor, repräsentiert durch $\boldsymbol{W}_a$, kaum angepasst werden. Erst bei Erreichen des rotierenden Zustandes wird eine wirkliche Modellierung notwendig. Außerdem sinkt die Sensitivität von $\boldsymbol{W}_c$, da eine bereits entstandene Rotation durch geringe Energiezufuhr aufrecht erhalten werden kann.

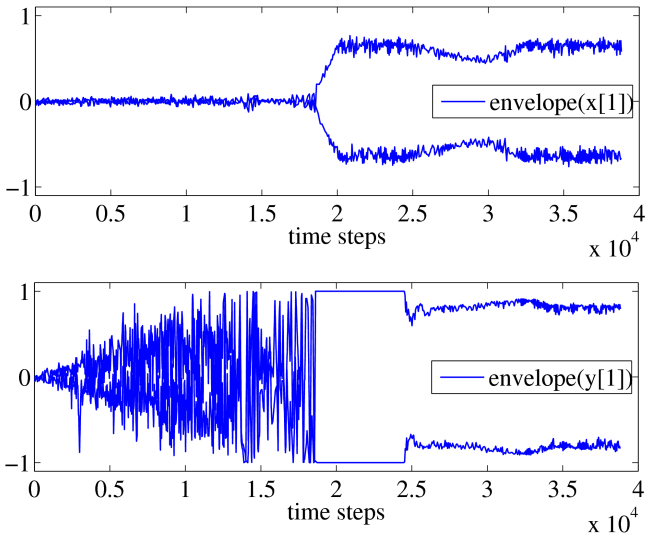

Abbildung 3.11: Sensor- und Motorwerte der „skidding snake“: Während der ersten 18000 Zeitschritte werden die Motoraktionen immer stärker, jedoch bewegt sich die Schlange nicht weit von ihrem ursprünglichen Ort weg, da der restliche Körper zu schwer ist. Danach erreicht die Schlange den rotierenden Zustand und die Motorwerte werden bis zum Zeitschritt 25000 wieder gedrosselt (aus [HMDH09, S. 7]).

3.4.2 Kategorisierung

Wird ein Weltmodell benötigt?

Es wird ein Weltmodell benötigt, da der Fehler $\boldsymbol{\xi}(t)$ der Prädiktion wesentlicher Bestandteil des Algorithmus ist. Jedoch ist der verwendete Prädiktor immer nur ein kleiner, linearisierter Ausschnitt der gesamten Welt. Über eine hohe Lernrate des Gradientenabstieges wird der Prädiktor immer an die aktuelle Situation angepasst. Es ist somit kein Vorwissen über die Welt notwendig.

Wie werden die Explorationsergebnisse repräsentiert?

Die Explorationsergebnisse sind in dem Parametersatz $\boldsymbol{W}_c$ und $\boldsymbol{b}_c$ für den Regler repräsentiert. Bei dem hier beschriebenen Algorithmus gehen diese Ergebnisse verloren, da die ständig aktualisiert und überschrieben werden. In [MFH08] wird eine Erweiterung beschrieben, welche den Parameterraum weiter einteilt. Hier werden ähnliche Bewegungsmuster zu Experten zusammengefasst. So kann später ein bestimmtes Verhalten durch Aktivieren eines Experten erneut aufgerufen werden.

Wie wird die Exploration gesteuert und kann die Exploration gezielt beeinflusst werden?

Die Exploration wird über die gegenläufigen Ziele der Fehlerminimierung gesteuert. Es entsteht eine Art Tauziehen zwischen Exploration und Ausbeutung, bei der immer eine bestimmte Vorhersagbarkeit und Kontinuität in den Bewegungen vorhanden

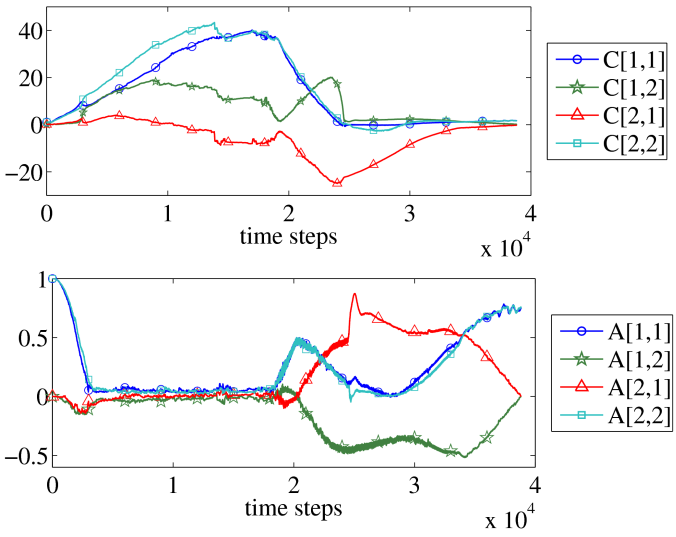

Abbildung 3.12: Parameterwerte des Reglers und des Prädiktors: Auch hier ist bis zum Zeitpunkt 18000 der Anstieg des Parameters $\boldsymbol{W}_c$ (hier C) zu erkennen, wobei der Prädiktor nahezu unverändert bleibt. Erst bei Eintritt in den rotierenden Zustand steigen die Werte von $\boldsymbol{W}_a$ (hier A). Weiterhin sinken die Werte von $\boldsymbol{W}_c$, da durch die hohe Rotationsgeschwindigkeit nur eine kleine Energiemenge hinzugefügt werden muss, um die Rotation aufrecht zu erhalten (aus [HMDH09, S. 8]).

bleibt. Doch auch wenn die bereits vorhandenen Ergebnisse in den Experten gespeichert werden, so gibt es noch keinen Regelmechanismus, der unbekannte Zustände im Parameterraum bevorzugt. In [HDH09] wird eine zweite Kontrollebene eingeführt, die über eine Hebb-Regel auf die Regelung Einfluss nimmt. In dem durchgeführten Experiment verbrachte ein fahrbarer Roboter auf einer begrenzten, ebenen Fläche dadurch weniger Zeit an den Begrenzungen, er explorierte viel mehr den Kern der Fläche. Abschließend kann festgehalten werden, dass ein direktes Eingreifen in die Regelstruktur nicht möglich ist, jedoch über die Fehlerfunktion aus Gleichung (3.38) auf Seite 48 das Verhalten beeinflusst werden kann.

Kann die Exploration unbegrenzt fortgesetzt werden?

Die Exploration unterliegt keinen Einschränkungen in Bezug auf die Dauer. Selbst eine Einteilung in die verschiedenen Experten oder eine weitere Kontrollebene führen zu keinen Speicherproblemen. Einzig die Realisierung innerhalb des 100 Hz Taktes der M-Serie könnte bei Verwendung aller Erweiterungen schwierig werden.

Wie kann auf die Explorationsergebnisse zugegriffen werden?

Der direkte Zugriff auf die Explorationsergebnisse würde über die entstehenden Experten des Parameterraumes erfolgen. Dadurch können bestimmte Verhaltensweisen abgespielt, jedoch keine aufgezwungenen Bewegungen von außen, beispielsweise über die Keyframe-Technik, durchgeführt werden.

Kann sich das System an eine wandelnde Umwelt anpassen?

Das System kann sich an wandelnde Bedingungen anpassen, da sich die Regelung von Grund auf ständig verändert. Durch die hohen Lernraten ϵ_a und ϵ_c ist es möglich sehr schnell das aktuelle Verhalten zu ändern. Die gezeigten Diagramme sind ebenfalls im Rahmen von wenigen Minuten auf einem echten Roboter realisierbar. Jedoch gibt es bisher noch keine Ansätze, wie gespeicherte Experten im Parameterraum aktualisiert oder gelöscht werden können.

Wie hoch ist die Komplexität des Algorithmus'?

Der Algorithmus und seine Erweiterungen beinhalten viele lineare Operationen. Die Matrix-Operationen, inklusive der Invertierung, sind in $O(l^3)$ durchführbar (siehe [Heu03, S. 301]). Diese Schranke kann jedoch erst bei sehr hochdimensionalen Räumen zu Problemen führen. Bei der Speicherung von Experten wird die Komplexität um das Suchen in dem Expertengraph um $O(n)$ erweitert.

Ist der Algorithmus neuronal implementierbar?

Der Ansatz eignet sich nur bedingt für eine neuronale Implementierung. Die zusätzliche Kontrollebene und die Vorwärtsrichtung des Algorithmus können neuronal mit einfachen Feed-Forward-Netzen umgesetzt werden. Jedoch kann die Invertierung eines neuronalen Netzes durch sich selbst nicht implementiert werden.

3.4.3 Zusammenfassung

Die Fülle an Veröffentlichungen und die Vielzahl der Forscher aus unterschiedlichen Forschungseinrichtungen, die an diesem Explorationsalgorithmus arbeiten, zeigt sowohl die Aktualität, als auch das erwartete Potential dieser Methode. Hier wird erstmals ohne gezielte Einwirkung eine Exploration durchgeführt.

Weiterhin zeigen die kurz aufgeführten Möglichkeiten der zusätzlichen Kontrollebene und der Abspeicherung gefundener Verhaltensweisen in Experten, dass dieser Ansatz an vielen Stellen erweiterbar ist. Andererseits ergeben sich hierbei ähnliche Probleme wie beispielsweise bei den Methoden der intrinsischen Motivation, da hier ebenfalls eine Einteilung in Experten geschieht. Es stellt sich die Frage nach der Anzahl an Experten. Ist diese fest oder können sich die Experten aufteilen?

Interessant ist ebenfalls die zusätzliche Kontrollebene, welche eine Exploration an den Stellen bevorzugt, an denen es auch etwas zu explorieren gibt. Indirekt wurde dadurch auch eine Hindernisvermeidung implementiert.

3.5 Gemeinsamkeiten der Explorationsalgorithmen

Die beiden ersten Algorithmen aus Abschnitt 3.1 ab Seite 25 und Abschnitt 3.2 ab Seite 34 sind sich von Grund auf sehr ähnlich. Es wird durch aktive Auswahl der Motoraktionen der sensorische Raum kategorisiert und mit den ausgeführten Motoraktionen verknüpft. Der Unterschied besteht in der Granularität der Einteilung des sensorischen Raumes. Während bei dem stabilitätsgetriebenen Algorithmus nur wenige Experten mit viel Zusatzwissen an den markanten Punkten des Phasenraumes existieren, wird das neuronale Netz gleichmäßig mit vielen Neuronen ohne zusätzliches Wissen etabliert. Eine Möglichkeit der Kombination wäre die feine Granularität beizubehalten, jedoch den einzelnen Neuronen weitere Informationen zuzuteilen. Diese Informationen könnten zum Beispiel die Zugehörigkeit zu einer Ruhemenge oder einem Basin sein.

Klar zu differenzieren sind die beiden Algorithmen jedoch an ihrer grundsätzlichen Ausrichtung, welche Art von dynamischen Systemen exploriert werden. Der stabilitätsgetriebene Algorithmus basiert darauf, dass das System eine eigene Dynamik besitzt und ohne äußere Einwirkung zu einem stabilen Zustand strebt. Der zweite Algorithmus legt genau das Gegenteil zugrunde. Es wird ein dynamisches System angenommen, in dem sich ein masseloser, punktförmiger Roboter auf einer ebenen Fläche bewegt. Erfolgt keine motorische Ansteuerung, so wird die Dynamik des Systems sofort zum Erliegen kommen. Diese Eigenschaft wird bei der Entwicklung des neuen Explorationsalgorithmus verändert werden.

Die ersten beiden Algorithmen generieren die Motoraktionen aktiv, wodurch Vorgaben durch eine Art Steuereinheit gemacht werden. Vorteilhafter ist hier die Methode der intrinsischen Motivation, da hier die Vorgaben an „natürliche“ Bedürfnisse angepasst werden können. Ein Bedürfnis für einen Roboter ist in erster Linie die Stromversorgung. Jedoch bleiben hier die möglichen Aktionen diskret, da sonst kein Q-Learning möglich ist. Wird das Q-Learning kontinuierlich und neuronal implementiert, kann ein neuronales Feld verwendet werden. Der dadurch entstehende Algorithmus ähnelt somit dem zweiten Algorithmus, der grundsätzlich auf neuronalen Netzen basiert. Der entscheidende Unterschied ist die Speicherung der Vorhersagen in Experten, die jedoch für eine Planung nicht herangezogen werden können. Der Algorithmus basierend auf der intrinsischen Motivation weist für die gestellten Aufgaben somit keine Vorteile auf.

Der zuletzt vorgestellte Algorithmus verwendet eine andere Art der Aktionsauswahl. Durch die Stabilisierung in und entgegen der Zeitskala wird hier eine kontinuierliche Bewegung des Roboters erzeugt. In weiteren Arbeiten basierend auf der Homöokinese werden die Parameter des Reglers mit einem GNG-Algorithmus eingeteilt. Dies ähnelt der Unterteilung des sensorischen Raumes des zweiten Algorithmus. Die beiden Netze können als zwei unterschiedliche Ebenen der Verhaltensrepräsentation angesehen werden. Das Netz des sensorischen Raumes bietet rudimentäre Möglichkeiten, um sensorische Werte anzusteuern, beispielsweise mit der Keyframe-Technik. Das Netz über dem Parameterraum des Reglers bietet umfangreichere Bewegungsmuster. Hier können Stabilisierungen und viele unterschiedliche Arten von Oszillationen in einem einzelnen Neuron repräsentiert werden. Die homöokinetische Exploration bietet somit als einziges

Verfahren die Möglichkeit, komplexere Verhaltensweisen, basierend auf einer sensomotorischen Schleife, in einem einzigen Neuron zu speichern und abzurufen.

3.6 Entwicklung eines neuen Explorationsalgorithmus

Die Kernideen des neuen Algorithmus' sind die Verwendung der homöokinetischen Regelung und der parallele Aufbau mehrerer neuronaler Netze. In Abbildung 3.13 ist der Ablauf dargestellt. Die Steuereinheit aus Abschnitt 3.2 ab Seite 34 wird für spätere Experimente zum Vergleich beibehalten und leicht modifiziert.

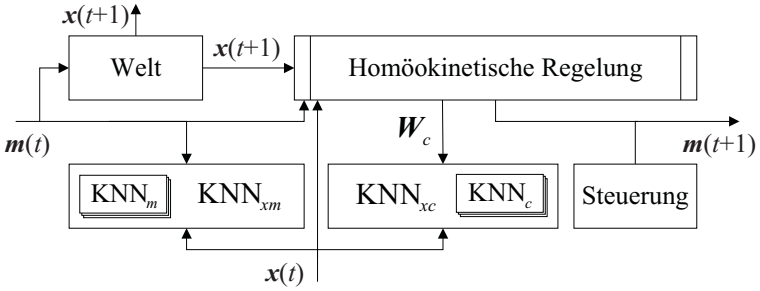

Abbildung 3.13: Kombinierte Architektur aus der homöokinetischen Exploration und der Speicherung in mehreren neuronalen Netzen: Die Architektur der homöokinetischen Regelung bleibt erhalten und wird hier zusammengefasst dargestellt. Weiterhin werden die zwei neuronalen Netze KNN_{xm} und KNN_{xc} parallel angelegt. Der untere Eingang kennzeichnet den zu kategorisierenden Raum und die Werte des oberen Einganges werden an die Kanten notiert.

Der Explorationsalgorithmus wird später sowohl in der Simulation als auch an einem Roboter der M-Serie getestet. Aufgrund dessen besteht der Eingangsvektor

$$\boldsymbol{x}\left(t\right) := \left(\boldsymbol{x}_{abs}\left(t\right)^T \quad \boldsymbol{x}_{rel}\left(t\right)^T \quad \boldsymbol{x}_{ene}\left(t\right)^T\right)^T \in \left[-1,1\right]^{3n} \tag{3.40}$$

aus folgenden Komponenten:

1. $\boldsymbol{x}_{abs}\left(t\right)$: Alle absoluten (Winkel-)Positionen des Roboters
2. $\boldsymbol{x}_{rel}\left(t\right)$: Alle Änderungen der (Winkel-)Positionen des Roboters zum letzten Zeitschritt. Der Wertebereich wird mit $\beta_{rel} > 1$, $\beta_{rel} \in \mathbb{R}$ und $\boldsymbol{b}_{rel} \in \mathbb{R}^n$ auf den Wertebereich $[-1,1]$ skaliert.

$$\boldsymbol{x}_{rel}\left(t\right) := \beta_{rel}\left(\boldsymbol{x}_{abs}\left(t\right) - \boldsymbol{x}_{abs}\left(t-1\right)\right) + \boldsymbol{b}_{rel} \tag{3.41}$$

3. $\boldsymbol{x}_{ene}\left(t\right)$: Der Energieverbrauch jedes einzelnen Motors, mit $\boldsymbol{x}_{ene}\left(t\right) \geq \boldsymbol{0}$

Jeder Freiheitsgrad, der sensorisch in $\boldsymbol{x}_{abs}$ erfasst wird, kann auch motorisch mit einem Signal gesteuert werden, weshalb der Motorvektor $\boldsymbol{m}\left(t\right)$ die Dimension n hat.

Der zu kategorisierende Raum der neuronalen Netze wird durch den Eingangsvektor von unten und die Information an den Kanten durch den Eingangsvektor von oben bestimmt. Das Netz KNN_{xm} kategorisiert den sensorischen Raum von $\boldsymbol{x}_{abs}\left(t\right)$ und

speichert die Motoraktionen $\boldsymbol{m}(t)$ an den Kanten, wie es in Abschnitt 3.2 ab Seite 34 beschrieben worden ist. Weiterhin verfügt jede Kante über ein neuronales Netz des motorischen Raumes KNN_m.

Das Netz KNN_{xc} kategorisiert ebenfalls $\boldsymbol{x}_{abs}(t)$, speichert jedoch an den Kanten den zugehörigen Parametersatz $\boldsymbol{W}_c$ in einem neuronalen Netz pro Kante.

In den folgenden Abschnitten werden die einzelnen Komponenten des Algorithmus' erläutert. Dabei wird gesondert auf die Funktionsweise während der Exploration und der Planung eingegangen.

3.6.1 Homöokinetische Regelung

Die homöokinetische Regelung wird aus Abschnitt 3.4 ab Seite 46 übernommen. Für eine differenziertere Exploration werden hier zwei Erweiterungen hinzugefügt. Während der Planungsphase erfolgt keine interne Aktualisierung des Reglers, sondern es können durch eines der neuronalen Netze Parametersätze $\boldsymbol{W}_c$ vorgegeben werden.

Exploration

Die reine homöokinetische Regelung verwendet $\boldsymbol{x}_{rel}(t)$. Die Erweiterungen werden die anderen Komponenten von $\boldsymbol{x}(t)$ verwenden. Die beiden Erweiterungen der Regelung sind:

1. Zusätzliche Neuronen-Schicht zum Selbstschutz
2. Belohnung energetisch niedriger Aktionen

Die Funktionsweise einer zusätzlichen Neuronen-Schicht wird in [HDH09] beschrieben. Hier ist ein Roboter mit zwei Motoren, die jeweils ein Rad steuern, in einem runden, ebenen Käfig gefangen. Dieser Raum wird nun sowohl mit, als auch ohne einer zusätzlichen STD-Neuronen-Schicht exploriert. Die Auswertung des Aufenthaltsortes zeigt, dass der Roboter nach einiger Zeit die Ränder der Welt meidet. Dies resultiert aus einem hohen Fehler an diesen Positionen. Die zusätzliche Neuronen-Schicht sagt diesen Fehler voraus und führt dazu, dass der Roboter bereits in der Nähe des Randes einen Weg einschlägt, der ihn in die Richtung des Zentrums führt.

Bei der zusätzlichen Schicht ist darauf zu achten, dass nicht die Geschwindigkeit des Roboters verwendet wird, sondern der Eingangsvektor aus den Messwerten von acht Abstandssensoren besteht. Dies ist für das Beispiel ausreichend, da lediglich eine lineare Separation der Entfernung zur Wand durchgeführt wird.

Bei der Verwendung der M-Serie Roboter soll dieser Mechanismus davor schützen, die Anschläge der Gelenke häufig anzufahren. Anstatt eines Abstandssensors wird hier der absolute Winkel des Gelenkes verwendet.Da es zwei Anschläge, in Beuge- und Streckrichtung gibt, genügt hier ein einzelnes STD-Neuronen pro Gelenk nicht. Es gibt im Wesentlichen zwei Möglichkeiten beide Anschläge zu erkennen:

1. Es wird die doppelte Anzahl an STD-Neuronen verwendet.

2. Es wird die gleiche Anzahl an RBF-Neuronen verwendet.

Hier wurde die zweite Möglichkeit gewählt. Die RBF-Neuronen werden mit einem Gradientenverfahren nach Abschnitt 2.4.1 ab Seite 18 trainiert. Ziel ein hoher Ausgangswert der Neuronen, wenn sich das Gelenk frei bewegen kann. In die Fehlerfunktion jedes Neurons V_i nach Gleichung (2.31) auf Seite 19 gehen deswegen $x_{abs,i}\,(t+1)$ und $(1-\xi_i\,(t+1))$ ein, mit

$$\boldsymbol{\xi}\,(t+1) := c_1\,(\boldsymbol{x}_{rel}\,(t+1) - \hat{\boldsymbol{x}}_{rel}\,(t+1)) + c_2. \tag{3.42}$$

Mit den Faktoren c_1 und c_2 wird eine Skalierung von $\boldsymbol{\xi}$ in einen bestimmten Wertebereich durchgeführt. Eine Analyse, welcher Wertebereich sinnvoll ist, wird in Abschnitt 5.4 ab Seite 81 durchgeführt. Die Ausgangswerte aller RBF-Neuronen werden zu dem Vektor $\boldsymbol{\zeta}\,(t+1) \in \mathbb{R}^n$ zusammengefasst. Dieser Vektor wird anschließend zusammen mit $\boldsymbol{\xi}\,(t+1)$ zur Erweiterung der Fehlerfunktion der homöokinetischen Regelung aus Gleichung (3.38) auf Seite 48 verwendet:

$$\boldsymbol{\chi}\,(t+1) = \boldsymbol{\xi}\,(t+1) + \boldsymbol{\zeta}\,(t+1)\,. \tag{3.43}$$

Zur Übersicht werden die Zeitindizes $(t+1)$ der Fehlerfunktionen in diesem Abschnitt weggelassen.

$$f_e\,(\boldsymbol{x}_{rel}, \hat{\boldsymbol{x}}_{rel}) := \left(\boldsymbol{L}\,(\boldsymbol{x}_{rel})^+\,\boldsymbol{\chi}\right)^T \left(\boldsymbol{L}\,(\boldsymbol{x}_{rel})^+\,\boldsymbol{\chi}\right) \tag{3.44}$$

Somit hat die RBF-Schicht eine direkte Auswirkung auf die Veränderung der Parameterwerte der Steuereinheit und das Explorationsverhalten. Mit dieser Funktionalität wird erreicht, dass bei den realen Tests mit den M-Serie Robotern die Anschläge der Gelenke gemieden werden und das Ziel des Selbstschutzes erreicht werden kann.

Die zweite Erweiterung zielt speziell auf das Erreichen eines niedrigen Energieniveaus der Exploration ab. So wird es in einer späteren Planungsphase möglich sein aus mehreren Varianten der Fortbewegung auszuwählen. Um sich jedoch energetisch auf einem niedrigen Niveau fortzubewegen wird hier bereits bei der Exploration dieser Aspekt berücksichtigt.

Die Möglichkeit der gezielten Exploration wird in [MHD07] vorgestellt. Hier wird eine hohle Kugel mit drei senkrecht zueinander stehenden Achsen im Inneren verwendet. An jeder Achse befindet sich eine Masse, die entlang dieser verschoben werden kann. Durch Verlagerung der Massen ist es möglich, die Kugel zu bewegen. In der Arbeit wird speziell das Rollen beziehungsweise Rotieren durch eine Erweiterung der Fehlerfunktion aus Gleichung (3.38) auf Seite 48 untersucht. Für die Betrachtungen dieses Algorithmus wird direkt Gleichung (3.44) verwendet.

Die erweiterte Fehlerfunktion

$$f_e^*\,(\boldsymbol{x}_{rel}, \hat{\boldsymbol{x}}_{rel}) := (1 - \tanh\,(r)) f_e\,(\boldsymbol{x}_{rel}, \hat{\boldsymbol{x}}_{rel}) \tag{3.45}$$

verwendet einen in jedem Zeitschritt berechneten Wert $r(t+1) \in \mathbb{R}$ als Belohnung. Negative Werte führen zu einer Bestrafung und zu einer raschen Änderung des Verhaltens, wohingegen positive Werte das aktuelle Verhalten belohnen und es länger bestehen bleibt.

Für die Experimente dieser Arbeit muss demnach eine Funktion gefunden werden, die den Energieverbrauch in einem Wert darstellt. Die Sensorwerte des Stromverbrauches eines M-Serie Roboters sind in $\boldsymbol{x}_{ene}$ vereint. Die Belohnung

$$r(t+1) := \exp^{-\boldsymbol{x}_{ene}(t+1)^T \boldsymbol{x}_{ene}(t+1)} \tag{3.46}$$

ist damit umso größer, je kleiner der Energieverbrauch ist.

Beide Erweiterungen zeigen, dass die Exploration über die homöokinetische Regelung beeinflussbar ist. Somit können, ähnlich zu den Ansätzen der intrinsischen Motivation über eine Funktion, in diesem Falle die Fehlerfunktion, bestimmte Verhaltensweisen bevorzugt werden. Beide Erweiterungen führen schließlich zu folgender Fehlerfunktion:

$$f_e(\boldsymbol{x}_{rel}, \hat{\boldsymbol{x}}_{rel}) := (1 - \tanh(r)) \left(\boldsymbol{L}(\boldsymbol{x}_{rel})^+ \boldsymbol{\chi}\right)^T \left(\boldsymbol{L}(\boldsymbol{x}_{rel})^+ \boldsymbol{\chi}\right) \tag{3.47}$$

Planung

Während der Planung wird die Steuereinheit nicht über die Gleichung (3.38) auf Seite 48 aktualisiert. Somit ist die Bestimmung des Fehlers nach Gleichung (3.47) nicht notwendig. Lediglich der Prädiktor wird weiterhin aktualisiert. Wird zu einem späteren Zeitpunkt zurück zur Exploration gewechselt, so ist dieser bereits aktuell. Die notwendigen Parameter der Regelung werden aus einem der parallel erstellten neuronalen Netze ermittelt.

3.6.2 Aktive Motorsteuerung

Exploration

Um einen Vergleich der explorierten neuronalen Felder zu ermöglichen, wird die in Abschnitt 3.2 ab Seite 34 erläuterte aktive Motoransteuerung leicht verändert. In der ursprünglichen Variante wurden nur ebene Welten exploriert. Sobald jedoch Welten mit einer Steigung exploriert werden, sollte die Stärke der Exploration variiert werden. Hierfür werden die Gleichungen (3.16) und (3.18) auf Seite 34 beibehalten und Gleichung (3.17) auf Seite 34 wird zu

$$\Delta \boldsymbol{y} := \eta_m \sum_{i=1}^{k} m_i \begin{pmatrix} \cos(\varphi_i) \\ \sin(\varphi_i) \end{pmatrix}, \tag{3.48}$$

mit $\eta_m \in]0,1[$ modifiziert. Ähnlich zu m_i wird η_m zufällig gewählt und ebenfalls mit der Wahrscheinlichkeit p aktualisiert.

Auch hier kann der Selbstschutz integriert werden. Wird die Steuerung um einen Prädiktor wie er in der homöokinetischen Regelung vorhanden ist erweitert, so kann ebenfalls eine RBF-Neuronenschicht trainiert werden, um Anschläge zu vermeiden. Eine einfache Möglichkeit um eine Richtungsänderung an den Rändern des gültigen Wertebereiches für die Gelenke zu erreichen ist die Veränderung der Auswahlwahrscheinlichkeit p. Statt p fest vorzugeben könnte der minimale Ausgangswert der RBF-Neuronen verwendet werden:

$$p\left(t\right) := \max_{V_i \in \mathbb{V}} \{1 - y_{V_i}\left(t\right)\} \tag{3.49}$$

Diese Auswahl führt dazu, dass an den Rändern die Wahrscheinlichkeit für einen Richtungswechsel zunimmt. Eine Erweiterung wäre, dass eine bestimmte Grundwahrscheinlichkeit p_0 nicht unterschritten wird. Weiterhin könnte die konkrete Auswahl der Richtung nicht rein zufällig, sondern ebenfalls abhängig von der letzten Bewegungsrichtung und dem Verlauf der Ausgangssignale der RBF-Neuronen gewählt werden.

Planung

Während der Planungsphase wird die Motorsteuerung deaktiviert.

3.6.3 Neuronale Netze des sensorischen Raumes: KNN_{xm} und KNN_{xc}

Exploration

Bei dem Aufbau der neuronalen Netze des sensorischen Raumes von $\boldsymbol{x}_{abs}\left(t\right)$ während der Exploration wird der bereits vorgestellte Algorithmus aus Abschnitt 3.2.1 ab Seite 34 erweitert.

Nur das Netz KNN_{xm} speichert einen durchschnittlichen Motorvektor an jeder Kante. Für die Berechnung des Durchschnitts werden zwei unterschiedliche Aktualisierungsmethoden untersucht. Es werden einerseits Gleichung (3.27) auf Seite 36 und Gleichung (3.29) auf Seite 36 verwendet, jedoch erscheint andererseits eine konstante Aktualisierung im Hinblick auf das lebenslange Lernen sinnvoller. Deswegen werden für die Aktualisierung der Gewichtsvektoren

$$\boldsymbol{w}_{V_i}\left(t+1\right) = \tau_w * w_{V_i}\left(t\right) + (1-\tau_w)\boldsymbol{x}_{abs}\left(t\right) \tag{3.50}$$

und für die Motoraktionen

$$\boldsymbol{m}_{E_{ij}}\left(t+1\right) = \tau_m * \boldsymbol{m}_{E_{ij}}\left(t\right) + (1-\tau_m)\boldsymbol{m}\left(t\right) \tag{3.51}$$

verwendet, mit τ_m und τ_w aus $]0,1[$. Neben dem durchschnittlichen Motorvektor wird an jeder Kante außerdem ein neuronales Netz des motorischen Raumes gespeichert. Dieses Netz wird im nächsten Abschnitt genauer beschrieben.

Die Neuronen des Netzes KNN_{xc} werden nach Gleichung (3.50) aktualisiert. Bei der Speicherung der Parameterwerte $\boldsymbol{W}_c$ ist es nicht sinnvoll, einen gesamten Mittelwert zu bilden. Wird beispielsweise eine einzelne Dimension betrachtet, so kann die Fallun-

terscheidung aus Abschnitt 2.1 ab Seite 6 zur Betrachtung der Stabilität herangezogen werden. Es könnten unterschiedliche Verhaltensweisen zu einer Eigenschaft des Systems verbunden werden, das in der Realität jedoch nie eintreten würde. Ein konkretes Beispiel wäre eine Hysterese mit $\lambda_i := 2$ und ein Periode-2 Orbit mit $\lambda_j := -3$. Diese beiden Eigenschaften würden bei einer Mittelwertbildung zu einem stabilen Fixpunkt mit $\overline{\lambda} := -0.5$ fusionieren. Aus diesem Grund wird im KNN_{xc} an jeder Kante ein neuronales Netz gespeichert, welches den Parameterraum von $\boldsymbol{W}_c$ kategorisiert. Die Verhaltensweise dieses Netzes wird ebenfalls im nächsten Abschnitt detailliert erläutert.

Planung

Grundsätzlich verläuft die Planung auf den neuronalen Netzen wie es in Abschnitt 3.2.2 ab Seite 37 beschrieben worden ist. Grundlage der Planung ist der Gewinnwert jedes Neurons durch die Gleichung (3.32) auf Seite 38. Erst bei der konkreten Auswahl des Vektors der Kante in Gleichung (3.33) auf Seite 38 wird hier unterschieden.

Beim KNN_{xm} werden zwei Varianten verwendet. Zum einen wird die Auswahl wie in Gleichung (3.33) auf Seite 38 und zum anderen unter Zuhilfenahme der neuronalen Netze an den Kanten durchgeführt. Da das Netz KNN_{xc} keinen Mittelwert an der Kante besitzt, kann hier nur die Auswahl aus dem Netz der Kante erfolgen. Wie die Auswahl im neuronalen Netz der Kante abläuft wird im nächsten Abschnitt näher erläutert. Beide Varianten werden in den späteren Experimenten genauer auf ihre Güte hin untersucht.

3.6.4 Kantenlose Neuronale Netze: KNN_m und KNN_c

Die Netze KNN_m und KNN_c bestehen aus einer unverbundenen Menge von Neuronen $\mathbb{V}$. Jedes einzelne Neuron $V_i := \{\boldsymbol{p}, \boldsymbol{\psi}, e, u\}$ besteht aus einem Parametervektor $\boldsymbol{p}_{V_i}$, einem Vektor mit Zusatzinformationen $\boldsymbol{\psi} \in \mathbb{R}^2$, einem Fehlerwert und einem Nützlichkeitswert. Der Parametervektor ist bei KNN_m ein Motorvektor $\boldsymbol{m}_{V_i} \in \mathbb{R}^n$ und bei KNN_c wird die Matrix $\boldsymbol{W}_c$ als Spaltenvektor $\boldsymbol{w}_{V_i} \in \mathbb{R}^{n \cdot n}$ interpretiert. In $\boldsymbol{\psi}_{V_i}$ sind die Informationen über die benötigte Zeit und Energie für den Übergang zwischen zwei Neuronen des KNN_{xm} beziehungsweise KNN_{xc} gespeichert. Der Fehlerwert $e_{V_i} \in \mathbb{R}^+$wird für das Einfügen von neuen Knoten wie bei dem GNG-Algorithmus verwendet. Der Nützlichkeitswert $u_{V_i} \in \mathbb{R}^+$ wird während der Planung zur Validierung des Netzes verwendet.

Exploration

Bei einem Knoten in diesen Netzen ist es nicht sinnvoll eine Aktualisierung in jedem Zeitschritt durchzuführen, denn hier interessiert gerade die Dauer und der Energieverbrauch des gesamten Überganges. Die Aktualisierung erfolgt demnach erst nach dem vollständigen Übergang von einem Knoten zum nächsten. Ein Beispiel ist in Abbildung 3.14 auf der nächsten Seite dargestellt.

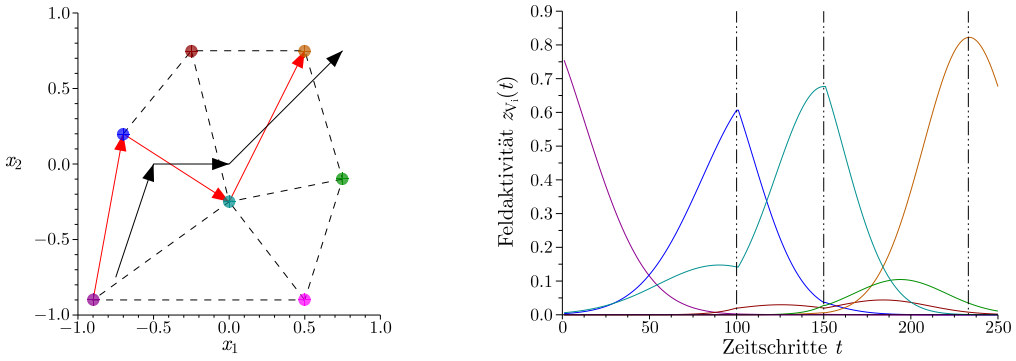

Abbildung 3.14: Aktualisierung der Kanten im KNN_{xm}: (links) Die Grafik zeigt einen Pfad (schwarze Pfeile) durch den sensorischen Raum von $\boldsymbol{x}_{abs}$. Weiterhin sind die Kanten dargestellt, die aktualisiert werden sollen (rot). (rechts) Die Grafik zeigt den zeitlichen Verlauf der Feldaktivität z_{V_i} der Neuronen (jeweils gleiche Farbe). Nach einem Wechsel des Neurons mit dem höchsten Ausgangswert wird abgewartet, bis dessen Ausgangswert sein Maximum erreicht hat. Genau dann wird die Kante aktualisiert (gestrichelte Linien).

In der linken Hälfte ist der sensorische Raum von $\boldsymbol{x}_{abs}$ mit einem neuronalen Netz mit sieben Knoten dargestellt, welches beispielsweise das Netz KNN_{xm} sein könnte. Weiterhin sind ein Pfad (schwarze Pfeile) und die zu aktualisierenden Kanten (rote Pfeile) abgebildet. An jeder Kante befindet sich ein weiteres neuronales Netz, beispielsweise KNN_m. Im rechten Abschnitt ist die Feldaktivität z_{V_i} jedes Neurons von KNN_{xm} dargestellt. Die Zugehörigkeit zu einem Knoten ist über die Farbe kodiert. Bei der linearen Bewegung durch den Phasenraum ist die exponentielle Transferfunktion sehr gut erkennbar. Während der ersten 50 Zeitschritte hat das erste Neuron (lila) den größten Ausgangswert und wird als Gewinner-Neuron bezeichnet. Danach findet der Wechsel zum zweiten Neuron (blau) statt. Es wäre jedoch zu früh, dieses Neuron nun schon zu aktualisieren, denn der Roboter bewegt sich immer noch auf das Neuron zu. Erst durch den Vorzeichenwechsel in der Steigung der Feldaktivität zum Zeitschritt 100 wird die Aktualisierung ausgelöst (gestrichelte Linien). Das gleiche Verhalten findet sich beim Übergang zu den nächsten Neuronen (türkis und braun) wieder.

Seien die Neuronen V_i und V_j zwei aufeinander folgende Gewinner-Neuronen im KNN_{xm} oder KNN_{xc}. Eine Aktualisierung findet also genau dann einmalig statt, wenn $z_{V_j}(t+1) < z_{V_j}(t)$ ist. Der Zeitpunkt der letzten Aktualisierung wird mit t^* bezeichnet. Eine darauf folgende Aktualisierung findet dann zwischen V_j und dem nächsten Gewinner-Neuron V_k statt. Der Vektor $\boldsymbol{\psi}_{V_i}(t^*)$ wird äquivalent zu Gleichung (3.51) auf Seite 57 durch

$$\boldsymbol{\psi}_{V_i}(t^*+1) = \tau_\psi * \boldsymbol{\psi}_{V_i}(t^*) + (1-\tau_\psi)\boldsymbol{\psi}(t) \tag{3.52}$$

aktualisiert. Der Vektor an Zusatzinformationen wird immer vom letzten Übergang an aufaddiert und ist zu jedem Zeitschritt durch

$$\boldsymbol{\psi}(t) := (\sum_{i=1}^{n} \tilde{x}_{ene,i}(t) \quad t - t^*)^T, \; \tilde{\boldsymbol{x}}_{ene}(t) := \sum_{t'=t^*}^{t} \boldsymbol{x}_{ene}(t') \tag{3.53}$$

definiert. Für die Aktualisierung wird hier ein festes τ_ψ aus dem Intervall $]0,1[$ verwendet. Die Aktualisierung der Parametervektoren jedes Neurons von KNN_m und KNN_c wird nach Gleichung (3.51) auf Seite 57 berechnet.

Planung

Die Planung wird bei Verwendung dieser Netze verändert. Wird ein Knoten ausgewählt, so wird dessen Motoraktion oder Parametersatz für mehr als einen Zeitschritt festgelegt. Die Zeitspanne richtet sich nach der durchschnittlichen Dauer für den entsprechenden Übergang, der in $\boldsymbol{\psi}$ gespeichert ist. Die Zusatzinformationen werden im neuronalen Netz nicht normiert abgespeichert und erst während des Bewertungsprozesses eines Knotens V_j durch

$$\psi_{V_j,i}^{norm} = \psi_i^{mask} \frac{\psi_{V_j,i}}{\max_{V_k}\{\psi_{V_k,i}\}},\ V_k \in \mathbb{V} \tag{3.54}$$

mit der Maske $\psi^{mask} \in [0,1]^2$ normiert. Die Maske dient der Selektion der zu verwendenden Zusatzinformationen. Jede Stelle k an der die Maske einen Wert ungleich 0 aufweist, wird für die Bewertung in Betracht gezogen. Der Term $\max_{V_k \in \mathbb{V}}\{\psi_{V_k,i}\}$ ist der Maximalwert für die Zusatzinformationen des Netzes und skaliert $\psi_{V_j,i}^{norm}$ in das Intervall $[0,1]$. Für die Bewertung eines Knotens durch

$$r_{V_i} := \exp^{-2\|\boldsymbol{\psi}_{V_i}^{norm} - \boldsymbol{\psi}_g\|^2 \sigma_\psi^{-2}} \tag{3.55}$$

wird weiterhin ein Zielvektor $\boldsymbol{\psi}_g \in [0,1]^2$ verwendet. Dieser Vektor ist an jeder Stelle j gleich 0, an der die Maske 0 ist. Alle anderen Werte können zwischen 0 und 1 frei gewählt werden. In der Regel wird der Wert 0 gewählt. Dieser Wert bedeutet, dass die Auswahl sich auf Neuronen mit möglichst kleinen Zusatzinformationen beschränkt. Nach Bewertung über eine bestimmte Maske und den Zielvektor kann der Knoten mit der maximalen Bewertung r ermittelt werden. Der Parametervektor dieses Knotens kann anschließend für die Planung verwendet werden.

Wurde ein Parametervektor ausgewählt und ausgeführt, so wird nach der Ausführung überprüft, ob der Zielknoten erreicht worden ist. Ist dies nicht der Fall, so wird der Nützlichkeitswert dieses Parametersatzes verringert:

$$u_{V_i}(t^* + 1) := \beta_u u_{V_i}(t^*), \tag{3.56}$$

mit $\beta_u \in]0,1[$. Unterschreitet die Nützlichkeit einen vorgegebenen Grenzwert $u_{min} \in]0,1[$ so wird der Knoten aus dem Netz gelöscht. Sind also Einträge vorhanden, die nur selten zum Erfolg führen, so werden diese rasch wieder gelöscht.

Die Zusatzvektoren und die neuronalen Netze ohne Kanten ermöglichen somit eine differenzierte Planung. Damit schließt der Algorithmus den Kreis von der initialen Exploration, über die Speicherung der Ergebnisse bis zur Auswertung und Verwendung der Ergebnisse. Die Einbettung des Algorithmus in eine objektorientierte Implementierung ist Thema des nächsten Kapitels.

Kapitel 4

Software-Architektur und Implementierung

I invented the term Object-Oriented,
and I can tell you I did not have C++ in mind.

Alan Curtis Kay (*1940), amerikanischer Informatiker

In diesem Kapitel wird die in dieser Arbeit entwickelte Software vorgestellt. Diese beinhaltet neben einer Ablaufsteuerung und drei Algorithmen auch eine Schnittstelle für reale und simulierte Welten. Die drei enthaltenen Algorithmen sind die in Abschnitt 3.2 ab Seite 34, Abschnitt 3.4 ab Seite 46 und Abschnitt 3.6 ab Seite 53 vorgestellten Algorithmen. Bei der Beschreibung der Software wird detailliert auf die grundlegenden Basisklassen, den Ablauf der parallelen Threads und deren Synchronisation eingegangen. Weiterhin wird neben den Welten der Simulation abschließend auf die Implementierung beschrieben.

4.1 Objektorientierte Software-Architektur

In Abbildung 4.1 auf der nächsten Seite ist die grundlegende Architektur in einem UML Klassen-Diagramm dargestellt. Die drei Basisklassen sind World, ExplorationApplication und Explorer. Weiterhin sind die abgeleiteten Klassen von World und Explorer aufgeführt. In der Mappe und deren Einträge sind die Parametersätze der Algorithmen gespeichert. Diese können in separaten Konfigurationsdateien festgelegt und anschließend bei Programmstart eingelesen werden. Im Folgenden werden die drei Basisklassen näher beschrieben.

4.1.1 Die Klasse Explorer

Die abstrakte Klasse Explorer repräsentiert die Hülle aller verwendeten Explorationsalgorithmen. Sie stellt lediglich drei öffentliche Methoden zur Verfügung, die jede abgelei-

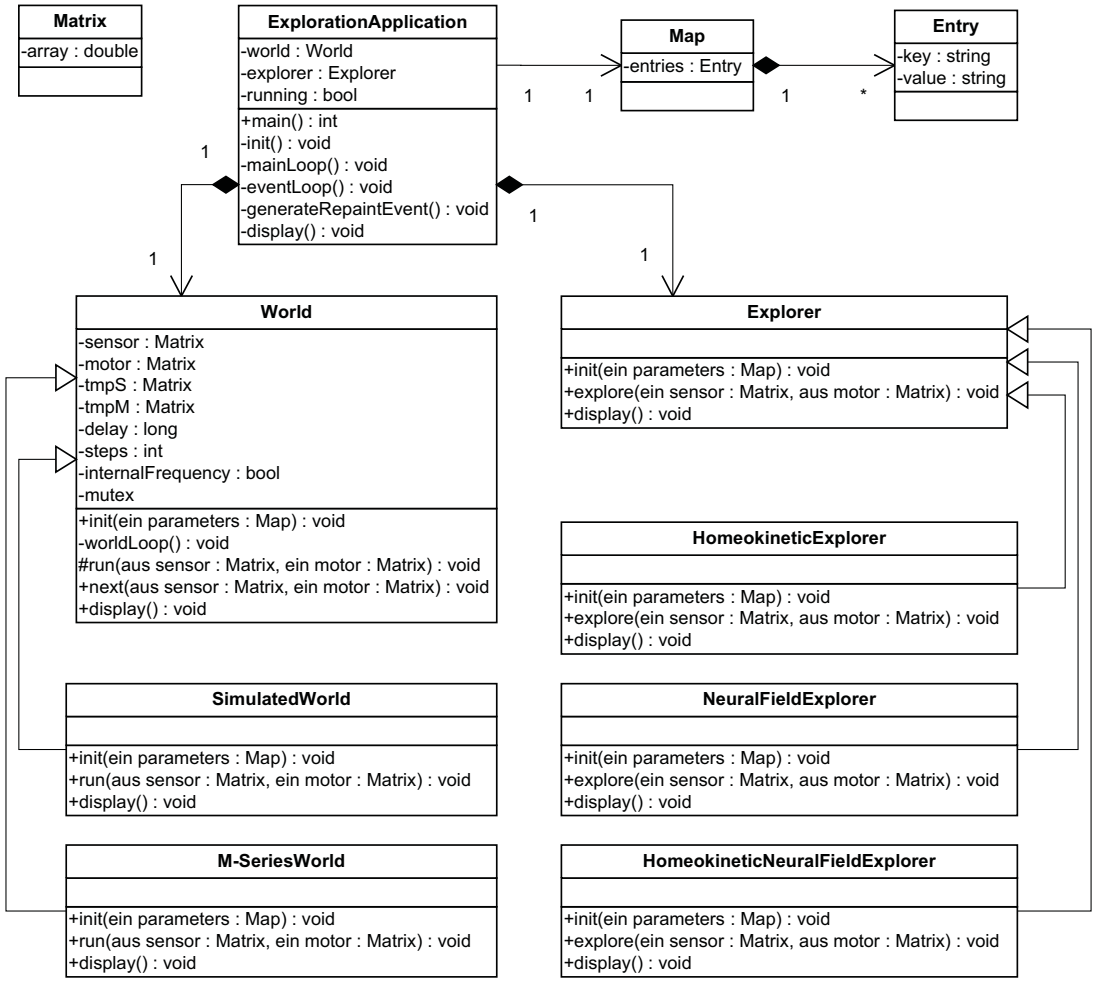

Abbildung 4.1: UML Klassen-Diagramm der Basisklassen: Die drei Basisklassen sind World, ExplorationApplication und Explorer. Abgeleitete Klassen von World können sowohl simulierte Welten als auch ein Roboter der M-Serie sein. Die Ablaufsteuerung ist in ExplorationApplication. Die Algorithmen der Exploration werden als abgeleitete Klassen von Explorer implementiert.

tete Klasse implementieren muss. Die Methode init (Map) wird nach dem Konstruieren einer Instanz aufgerufen. Hier werden alle vorhandenen Parameter mitgegeben und die Instanz entnimmt die entsprechenden Einträge. Weiterhin können hier interne Variablen initialisiert werden. In der Methode explore(Matrix∗, Matrix∗) wird der Explorationsalgorithmus zu einem Zeitschritt berechnet. Als Eingangsparameter dient der Sensorvektor sensor und die auszuführende Aktion wird in den Motorvektor motor geschrieben. Die Methode display () dient zur Darstellung in einem Fenster. Hier können Inhalte bis zur dritten Dimension angezeigt werden. Ein Beispiel wäre hier, dass der geschätzte und der erwartete Sensorwert angezeigt oder die Entwicklung bestimmter Parameter in einem Graph dargestellt werden.

4.1.2 Die Klasse World

Die Klasse World ist ebenfalls abstrakt, stellt jedoch neben abstrakten Methoden ebenfalls implementierte Routinen bereit. Auch hier wird nach dem Konstruieren einer Instanz die öffentliche Methode init (Map) mit den gleichen Eingangsparametern aufgerufen. Hier werden die internen Matrizen sensor, tmpS, motor und tmpM initialisiert und am Ende die private, abstrakte Methode init () aufgerufen. Hier können abgeleitete Klassen weitere Initialisierungen vornehmen.

Weiterhin verfügt die Klasse über einen eigenen aktiven Thread, der die Methode worldLoop() abarbeitet. Dieser Thread läuft so lange wie running auf **true** gesetzt ist. Innerhalb der Methode wird zyklisch run(Matrix∗, Matrix∗) aufgerufen, die den nächsten Sensorvektor sensor aus dem gegebenen Motorvektor motor berechnet. Mit der öffentlichen Methode next(Matrix∗, Matrix∗) kann der aktuelle Sensorvektor entnommen und ein neuer Motorvektor geschrieben werden. Der genaue Programmfluss und die Synchronisation mit den Parametern mutex, condition, internalFrequency, delay, steps werden in Abschnitt 4.2.4 ab Seite 65 beschrieben. Über display () können auch hier weitere Informationen, beispielsweise der aktuelle Sensorvektor in ein Fenster gezeichnet werden.

4.1.3 Die Klasse ExplorationApplication

Die Methode main() der Klasse ExplorationApplication ist der Eintrittspunkt der Software. Hier wird jeweils eine abgeleitete Klasse von World und Explorer instantiiert. Weiterhin werden das Multi-Thread-Umfeld, ein Darstellungsfenster und ein Timer eingerichtet. Der Timer generiert in einer niedrigen Frequenz ein Ereignis zum Neuzeichnen des Fensters. Hierfür wird die Methode generateRepaintEvent() aufgerufen. Das Ereignis wird in eine Warteschlange abgelegt, welche von der Endlosschleife eventLoop() abgearbeitet wird. Diese Schleife wird am Ende der main() Methode aufgerufen. Hier können weiterhin Tastatur-, Joystick- und Mauseingaben zur Programmsteuerung verarbeitet werden. Jedes Ereignis zum Neuzeichnen führt zu einem Aufruf der display () Methode, die den Aufruf an die beiden Instanzen von World und Explorer weiterleitet. Weiterhin wird ein zusätzlicher aktiver Thread instantiiert, der die mainLoop() abarbeitet. Hier wird immer abwechselnd next(Matrix∗, Matrix∗) und explore(Matrix∗, Matrix∗) von den Instanzen von World und Explorer aufgerufen.

4.2 Programmfluss der Software

Wie bereits erwähnt existieren drei parallele Threads und ein Timer im Hintergrund. Der Timer wird von der Multi-Thread Bibliothek bereitgestellt. Ihm werden die aufzurufende Methode und ein Intervall als Parameter übergeben. Nach Ablauf des Intervalls wird zyklisch die mitgegebene Methode, hier generateRepaintEvent(), aufgerufen.

4.2.1 Die Methode mainLoop()

Der Thread der die mainLoop() aus dem Quelltext 4.1 abarbeitet wird in der main() Methode erstellt. Zunächst werden zwei Vektoren Matrix* sensor und Matrix* motor mit

```
1 void mainLoop(){
2   Matrix* sensor = new Matrix(world->getSensorDimension(), 1);
3   Matrix* motor = new Matrix(world->getMotorDimension(), 1);
4   while(running){
5     world->next(sensor, motor);
6     explorer->explore(sensor, motor);
7 } }
```

Quelltext 4.1: Quelltext der mainLoop(): In Zeile 1 und 2 werden die Matrizen für den Sensor- und Motorvektor initialisiert. Die Schleife in den Zeilen 4-7 führt abwechselnd einen Zeitschritt in der Welt und anschließend einen Explorationsschritt aus. Über die Laufvariable running kann die Schleife beendet werden.

den entsprechenden Dimensionen der Welt instantiiert. Danach läuft die Schleife bis der Wahrheitswert running auf **false** gesetzt wird. Innerhalb der Schleife wird nun abwechselnd die Welt einen Zeitschritt inkrementiert und die Exploration durchgeführt.

4.2.2 Die Methode eventLoop()

Die eventLoop() aus dem Quelltext 4.2 auf der nächsten Seite wird als letzte Anweisung der main() Methode betreten. Hier werden alle Eingabeereignisse abgearbeitet. In dem Methodenaufruf WaitForEvent(Event*) wird solange gewartet, bis in der Ereignisschlange ein neues Ereignis vorhanden ist. Als Beispiel ist hier in den Zeilen 6 bis 10 das Loslassen der Taste „Q“ aufgeführt, welche zum Beenden der Software führt, indem die Laufvariable running auf **false** gesetzt wird. Weiterhin werden in den Zeilen 11 bis 15 die benutzerdefinierten Neuzeichnen-Ereignisse abgearbeitet. Diese führen zum Aufruf der Methode display().

Durch die Steuerung des Neuzeichnens über diesen Mechanismus ist es möglich, die eigentliche Simulation und deren Darstellung im Fenster asynchron mit unterschiedlichen Frequenzen zu betreiben. Die Simulation ist mit 100 Hz angestrebt. Diese Frequenz ist für das Neuzeichnen nicht sinnvoll, da zu viele Rechenressourcen verbraucht werden. Sinnvoll ist hier eine Frequenz von 10 - 20 Hz, die für den Timer ein Intervall von 100 - 50 ms bedeutet.

4.2.3 Die Methode worldLoop()

Der Thread, der die Methode worldLoop() aus dem Quelltext 4.3 auf Seite 66 abarbeitet, wird in der init(Map) Methode der Klasse World erzeugt. Auch hier wird die Schleife über den Wahrheitswert running gesteuert. Diese Schleife ist für das Einhalten eines vorgegebenen Taktes verantwortlich. Je nachdem ob eine reale oder simulierte Welt verwendet wird, wird dies unterschiedlich realisiert. Die M-Serie Roboter sind externe

```
1 void eventLoop(){
2   Event event;
3   while(running) {
4     WaitForEvent(&event);
5     switch (event.type) {
6       case KEYUP:
7         switch (event.key) {
8           case q: running = false;  break;
9         }
10        break;
11      case USEREVENT:
12        switch(event.user.code){
13          case REPAINT: display();  break;
14        }
15      break;
16 } } }
```

Quelltext 4.2: Die eventLoop() ist eine Endlosschleife, die über die Laufvariable running beendet werden kann. In Zeile 4 wird in der Methode WaitForEvent(Event∗) solange gewartet, bis ein Ereignis in der Warteschlange vorhanden ist. Über verschiedene **switch** Anweisungen wird nun der Typ des Ereignisses gefunden und eine entsprechende Aktion durchgeführt, z.B. das Beenden der Software (Zeilen 6-10) oder das Neuzeichnen des Fensters (Zeilen 11-15).

Taktgeber mit einer Frequenz von 100 Hz. Hier muss keine aktive Taktsteuerung erfolgen und der Wahrheitswert internalFrequency ist **false**. Wird eine Welt simuliert, so muss eine künstliche Pause eingehalten werden. Ob eine Pause notwendig ist und wie lange sie sein muss wird über die Werte t0, t1, delay und internalFrequency innerhalb der Methode wait(clock_t, clock_t) bestimmt. Bei eigener Taktgebung kann das Warteintervall auch auf 0 gesetzt werden, so dass der PC so schnell wie möglich simuliert.

Die Zeilen fünf bis neun sind eine einfache Synchronisation der Aktualisierung des Sensor- und Motorvektors. In jedem Zeitschritt wird die Zählvariable counter inkrementiert. Mit den Methoden mutexP(mutex) und mutexV(mutex) kann das mutex festgehalten bzw. wieder losgelassen werden. CondSignal(cond) sendet ein Signal an alle wartenden Threads auf dieser Bedingung.

4.2.4 Synchronisation der Exploration und der Umwelt

Die Methode next(Matrix∗, Matrix∗) aus dem Quelltext 4.4 auf der nächsten Seite wird zyklisch von der Methode mainLoop() aufgerufen. Die Methoden next(Matrix∗, Matrix∗) und worldLoop() greifen beide auf die gleiche Instanz der Sensor- und Motorvektoren im Speicher zu. Aufgrund dessen muss eine Synchronisation des Zugriffs erfolgen. Als Grundlage dient eine einfache Erzeuger-Verbraucher Synchronisation. Der Unterschied hier ist, dass der Erzeuger, also die Welt, nicht warten darf. Dies liegt an der externen Taktsteuerung der M-Serie. Wenn die Welt wartet, bis der aktuelle Zeitschritt vollständig verarbeitet worden ist, dann kann es sein, dass ein Datenpaket des Roboters während der Wartezeit verloren geht, oder dass die Daten zum falschen Zeitpunkt an den Roboter zurück gesendet werden und mit anderen Daten auf der Datenleitung kollidieren. Als Konsequenz muss die Exploration immer schneller sein als die Welt.

```
1 void worldLoop(){
2   while(running){
3     clock_t t0 = clock();
4     this->run(tmpX, y);
5     mutexP(mutex);
6     //update s and m
7     counter++;
8     mutexV(mutex);
9     CondSignal(cond);
10    clock_t t1 = clock();
11    wait(t0, t1);
12  }
13 }
```

Quelltext 4.3: Die worldLoop() ist eine Endlosschleife über running und leitet die Aktualisierung von sensor und motor an eine abgeleitete Klasse über run(Matrix∗, Matrix∗) weiter. Nach einer synchronisierten Aktualisierung wird ein Signal an die wartenden Threads von cond geschickt. Über die Methode wait(clock_t, clock_t) kann eine eigene Taktsteuerung erfolgen.

```
1 void next(Matrix* sensor,
2     Matrix* motor){
3   mutexP(mutex);
4   if (counter<steps){
5     while(counter<steps){
6       CondWait(cond);
7     }
8   } else { //error
9   }
10  counter = 0;
11  //update s and m
12  mutexV(mutex);
13 }
```

Quelltext 4.4: Die gesamte Methode next(Matrix∗ Matrix∗) ist synchronisiert. Zunächst wird in den Zeilen 4 bis 7 solange gewartet bis genau die richtige Anzahl an Zeitschritten in der Welt vergangen ist, wobei ein Fehler in der **else** Klausel bearbeitet werden kann. Abschließend wird der Zähler auf 0 zurückgesetzt und die Vektoren sensor und motor aktualisiert.

In der Methode next(Matrix∗, Matrix∗) wird nach dem Festhalten des mutex zunächst überprüft, ob alle vorhandenen Datenpakete abgearbeitet worden sind. Der Wert steps $\in \mathbb{N}$ gibt an über wie viele Zeitschritte ein Mittelwert gebildet werden soll. Ist steps $:= 1$, so wird jedes Datenpaket direkt abgearbeitet. Für steps > 1 ergibt sich für die Exploration eine Teilfrequenz, nämlich $100\,\mathrm{Hz}/\mathrm{steps}$. Aus dem eingegangenen Sensorvektor wird der Mittelwert über die Zeitschritte gebildet und der Motorwert über die Dauer konstant gehalten. Sind alle Datenpakete abgearbeitet, so kann gewartet werden, bis genügend Zeitschritte vergangen sind. Sind bereits Datenpakete eingegangen, so ist die Exploration zu langsam. Dies lässt sich über den **else** Fall erkennen und anzeigen. Bei normaler Abarbeitung werden dann sensor und motor aktualisiert und der counter auf 0 zurückgesetzt.

4.3 Welten der Simulation

Während der Entwicklung des Algorithmus wurden mehrere Welten entwickelt. Diese dienten der schnellen Validierung des Algorithmus, da Experimente mit diesen Welten mehr als zehn mal so schnell wie mit der M-Serie durchgeführt werden können. Im nächsten Kapitel wird der Algorithmus in Experimenten an diesen Welten getestet. In den nächsten beiden Abschnitten werden ein ebener Kreis und ein Gebirge beschrieben.

Aus beiden Welten kann direkt keine Information über die Energie erhalten werden. Aufgrund dessen wird der Energieverbrauch aus der aufgewendeten Motoraktion bestimmt:

$$\boldsymbol{x}_{ene}\left(t+1\right) = \boldsymbol{y}\left(t\right)^2 + \boldsymbol{b}_{ene} \tag{4.1}$$

Die aufgewendete Motoraktion geht quadratisch in den Energieverbrauch ein. Weiterhin wird in jedem Zeitschritt ein Grundenergieverbrauch $\boldsymbol{b}_{ene}$ angenommen, der durch das reine Betreiben eines realen Roboters anfallen würde. Neben der Energie müssen noch $\boldsymbol{x}_{abs}\left(t\right)$ und $\boldsymbol{x}_{rel}\left(t\right)$ bestimmt werden. Hier wird $\boldsymbol{x}_{rel}\left(t\right)$ in jedem Zeitschritt über Gleichung (3.41) auf Seite 53 bestimmt. Somit ist lediglich $\boldsymbol{x}_{abs}\left(t\right)$ von jeder einzelnen Welt abhängig.

4.3.1 Ebener Kreis

In Anlehnung an das Ziel des Selbstschutzes und zum Vergleich mit den Ergebnissen aus [HDH09] wird hier ebenfalls ein ebener Kreis als Welt verwendet. Die Motoraktivierung $\boldsymbol{y}\left(t\right)$ wird als Geschwindigkeit interpretiert und führt mit

$$\check{\boldsymbol{x}}_{abs}\left(t+1\right) = \boldsymbol{x}_{abs}\left(t\right) + \beta_y \check{\boldsymbol{y}}\left(t\right) + \boldsymbol{\xi}\left(t\right) \tag{4.2}$$

zu einem neuen Sensorwert $\boldsymbol{x}\left(t+1\right)$, mit $\beta_y := 0.01$ und einem normalverteilten Fehler $\boldsymbol{\xi}\left(t\right) \in [-0.1\beta_y; 0.1\beta_y]$. Abschließend wird die Position in den Einheitskreis projiziert:

$$\boldsymbol{x}_{abs}\left(t+1\right) = \frac{\check{\boldsymbol{x}}_{abs}\left(t+1\right)}{\|\check{\boldsymbol{x}}_{abs}\left(t+1\right)\|} \tag{4.3}$$

Bei einer maximaler Motoraktivierung von eins würde ein Roboter in Echtzeit 2 Sekunden benötigen um die Welt zu durchqueren. Es wird eine untere Schranke $b_y \in [0, 1]$ mit

$$b_y := 10\beta_y \tag{4.4}$$

für die Motoraktivierung $\boldsymbol{y}\left(t\right)$ eingeführt:

$$\check{\boldsymbol{y}}\left(t\right) = \begin{cases} \boldsymbol{y}\left(t\right) & \text{wenn } \boldsymbol{y}\left(t\right)^T \boldsymbol{y}\left(t\right) > b_y \\ \boldsymbol{0} & \text{sonst} \end{cases} . \tag{4.5}$$

Wird der ebene Kreis als dynamisches System mit einer Regelung betrachtet, so besteht der gesamte Sensorraum aus einer stabilen Ruhemenge. Diese Eigenschaft weist ebenfalls die Welt aus [Tou04] auf. Um die homöokinetische Regelung besser einbinden zu können wird im nächsten Abschnitt eine unebene Welt vorgestellt.

4.3.2 Gebirge

Das Gebirge ist eine unebene Welt und ist in Abbildung 4.2 auf der nächsten Seite dargestellt. Die Welt

$$f_g(x_1, x_2) = 0.5 * (\sin\left(8x_1x_2 - 1.6\right) + \cos\left(3x_1x_2 - 2.1\right)) \tag{4.6}$$

ist im Wertebereich $[-1,1]^2$ definiert und hat vier verborgene Bereiche. Jeweils zwei der Bereiche sind durch unterschiedlich hohe Gebirge vom Zentrum der Welt getrennt.

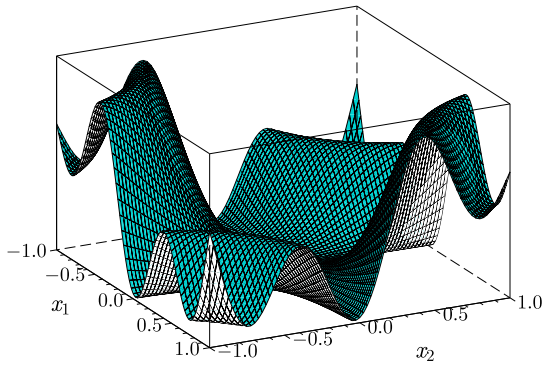

Abbildung 4.2: Unebene Welt: In der Welt gibt es vier versteckte Gebiete in den Ecken, die unterschiedlich schwer zu erreichen sind.

Hier wird die Motoraktivierung $\boldsymbol{y}(t)$ ebenfalls als Geschwindigkeit angesehen und führt mit der Geschwindigkeit aus der Umwelt zu

$$\boldsymbol{\vartheta}(t) = \frac{1}{2}\left(\beta_y \boldsymbol{y}(t) + \beta_g \frac{\delta}{\delta \boldsymbol{x}} f_g(\boldsymbol{x}(t))\right) \tag{4.7}$$

mit

$$\frac{\delta}{\delta \boldsymbol{x}} f_g(\boldsymbol{x}(t)) = \boldsymbol{x}(t) \begin{pmatrix} 4 & 1.5 \end{pmatrix} \begin{pmatrix} \cos(8\overline{x}(t) - 1.6) \\ \sin(3\overline{x}(t) - 2.1) \end{pmatrix}, \overline{x}(t) = \prod_{i=1}^{2} x_i(t). \tag{4.8}$$

Die Aktualisierung der Welt über einen Zeitschritt erfolgt durch

$$\check{\boldsymbol{x}}_{abs}(t+1) = \boldsymbol{x}_{abs}(t) + \boldsymbol{\vartheta}(t) + \boldsymbol{\xi}(t). \tag{4.9}$$

Die Parameter β_y und β_g sollten ungefähr im Verhältnis 1:10 gewählt werden. Damit ist gewährleistet, dass jeder Punkt der Welt erreicht werden kann. Da die Welt der Bewegung des Roboters entgegenwirkt, ist $\beta_y < 0$ und in der Implementierung ist $\beta_y := -0.0025$ und $\beta_g := 0.025$. Der Fehler $\boldsymbol{\xi}(t)$ ist wieder normalverteilt aus $[-0.1\beta_y; 0.1\beta_y]$. Abschließend wird die neue Position in den Wertebereich der Welt projiziert:

$$x_{abs,i}(t+1) = \begin{cases} 1 & \text{für } \check{x}_{abs,i}(t+1) \geq 1 \\ \check{x}_{abs,i}(t+1) & \text{für } -1 < \check{x}_{abs,i}(t+1) < 1 \\ -1 & \text{für } \check{x}_{abs,i}(t+1) \leq -1 \end{cases} \tag{4.10}$$

4.4 Implementierung der Software

Die Software wurde unter Microsoft Visual Studio 2008 und C++ entwickelt. Verwendete Bibliotheken sind das „lpzrobots“ Framework [MDH+09], die „Simple DirectMedia Layer“ (SDL) Bibliothek [SDL] und eine in der Arbeitsgruppe entwickelte COM-Anschluss Bibliothek.

Aus dem „lpzrobots“ Framework sind mehrere Klassen entnommen und wenn nötig angepasst und erweitert worden. Zur Berechnung von Matrizen sind zwei Klassen erweitert worden. Diese bieten unter anderem umfangreiche Möglichkeiten der Addition, Multiplikation, Invertierung oder elementweisen Operationen. Weiterhin wurden verschiedene Steuereinheiten ausprobiert und angepasst, um neben der aktiven Steuerung auch eine externe Planung zu ermöglichen.

Die SDL Bibliothek stellt Möglichkeiten zur graphischen Darstellung, zum parallelen Programmfluss und zum Abfangen von Eingaben verschiedenster Eingabegeräte. Es wird der OpenGL Standard für die zwei- oder dreidimensionale Anzeige der neuronalen Netze, der simulierten Welten und von verschiedenen Zeit-Diagrammen verwendet. Der in Abschnitt 4.2 ab Seite 63 beschriebene parallele Programmfluss wird über eine „Multi-Thread“ Umgebung der SDL Bibliothek umgesetzt. Weiterhin wird eine Ereignisschlange bereitgestellt, die sämtliche Systemereignisse von Eingabegeräten zur Verfügung stellt. Hierzu können neben Tastatur und Maus ebenfalls Joysticks oder virtuelle Tastaturen auf Touchscreen Monitoren gehören.

Für die Kommunikation und Steuerung eines Roboters der M-Serie wird ein virtueller COM Anschluss verwendet, der über eine separate Bibliothek initialisiert und ausgelesen beziehungsweise beschrieben werden kann. Die notwendige Hardware für die Kommunikation, eine kleine weiße Box, ist in Abbildung 5.8 auf Seite 80 abgebildet.

Die Implementierung stellt die notwendige Funktionalität bereit um den entwickelten Algorithmus sowohl in der Simulation als auch mit den Robotern der M-Serie zu testen. Die durchgeführten Experimente mit dieser Implementierung, sowie die gewonnenen Ergebnisse werden im nächsten Kapitel vorgestellt.

Kapitel 5

Experimente

No amount of experimentation can ever prove me right.
A single experiment can prove me wrong.

Albert Einstein (*1879, †1955), deutscher Physiker

In diesem Kapitel werden die durchgeführten Experimente des neuen Algorithmus' vorgestellt. In einem ersten Experiment wird überprüft, ob die erstellten neuronalen Netze des sensorischen Raumes bei konstanten Versuchsaufbau vergleichbar sind. Das Ziel des zweiten Experimentes ist es einen geeigneten Parametersatz für die Erstellung der Netze zu finden. In einem dritten Experiment werden die beiden Mechanismen zur Aktionsauswahl differenziert betrachtet und die Eigenschaften der jeweils erstellten neuronalen Netze analysiert. Im nächsten Experiment wird untersucht, ob der Mechanismus des Selbstschutzes über RBF-Neuronen erlernbar ist. Im letzten Experiment werden unterschiedliche Varianten der Planung analysiert. Hier werden vier Möglichkeiten zur Planung auf den erstellten neuronalen Netzen auf deren Eigenschaften untersucht.

5.1 Kolmogorow-Smirnow Anpassungstest

Mit dem Kolmogorow-Smirnow Anpassungstest (KSA) wird überprüft, ob eine empirisch ermittelte Wahrscheinlichkeitsverteilung einer vorgegebenen Verteilung entspricht. Dieser Test wird hier verwendet, um zu prüfen, ob die erstellten neuronalen Netze einer Regelmäßigkeit unterliegen. Dazu wird die maximale Differenz zwischen empirischer und theoretischer Wahrscheinlichkeitsdichtefunktion (WDF) ermittelt. Liegt dieser Wert innerhalb eines Grenzwertes, so kann eine Nullhypothese akzeptiert werden.

Hier wird überprüft, ob die Verteilung der Messgrößen des gleichen Tests über mehrere Simulationen hinweg einer Normalverteilung entspricht. Die Hypothesen für diesen Test können wie folgt formuliert werden:

H_0: Der empirisch ermittelten WDF liegt eine Normalverteilung zugrunde.

H_1: Der empirisch ermittelten WDF liegt eine andere Verteilung zugrunde.

Durchführung

Für den KSA werden sowohl der ebene Kreis als auch das Gebirge verwendet. Bis auf die Initialisierung des Zufallszahlengenerators startet jede Simulation mit den gleichen Bedingungen. Jede Welt wird 25 mal jeweils 100000 Zeitschritte simuliert. In jedem Zeitschritt werden die folgenden Messgrößen gespeichert:

- aktuelle Anzahl an Knoten und Kanten und die Anzahl aller eingefügten Kanten
- Mittelwert und Standardabweichung der Anzahl an Nachbarn, der Länge und des Alters der Kanten
- Mittelwert und Standardabweichung der Anzahl an Unterknoten, Dauer und Energieverbrauch einer Kante des Netzes

Auswertung

Die einzelnen Auswertungen der 25 Simulationen zeigen alle einen ähnlichen Verlauf. Die σ-Bereiche um die Mittelwerte aller erfassten Größen sind immer kleiner als 5% des Mittelwertes. Für diese Berechnungen wurden jeweils die letzten Messwerte der Simulationen herangezogen. Für die ersten drei Messgrößen ergibt sich beispielsweise:

Messgröße	aktuelle Knoten	aktuelle Kanten	erstellte Kanten
μ	80.638	401.558	515.013
σ	1.844	11.119	15.149
σ/μ	0.022	0.027	0.029

Tabelle 5.1: Mittelwerte μ und Standardabweichungen σ der ersten drei Messgrößen. Das Verhältnis von σ zu μ ist hier kleiner als 3%.

Wird die Entwicklung des Netzes über den zeitlichen Verlauf betrachtet, so zeigen die ersten Messgrößen den in Abbildung 5.1 auf der nächsten Seite dargestellten Verlauf. Aufgrund der unterschiedlichen Wertebereiche wurden die Messgrößen in zwei Diagramme aufgeteilt.

Die Diagramme zeigen auch über die Zeit hinweg einen stabilen Verlauf. Zu keinem Zeitpunkt ist das Verhältnis von σ-Bereich zu Mittelwert größer als 5%. Um zu überprüfen, ob die Ergebnisse dem Zufall unterliegen, oder ob eine Normalverteilung zugrunde liegt, wird der KSA verwendet.

Es wurde für jede Messgröße der Mittelwert μ und die Standardabweichung σ zum Zeitpunkt 100000 verwendet. Anhand dieser Werte wird die WDF jedes Parameters mit

$$\Phi(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) \mathrm{d}t \tag{5.1}$$

bestimmt. Es wird auf dem 5% Signifikanzniveau getestet. Aufgrund dessen ist die maximale Abweichung $d_{max} := 0.27$ für die vorgegebene Stichprobengröße an Simulationen. Ein repräsentatives Ergebnis dieses Tests für die Anzahl an Nachbarn ist in

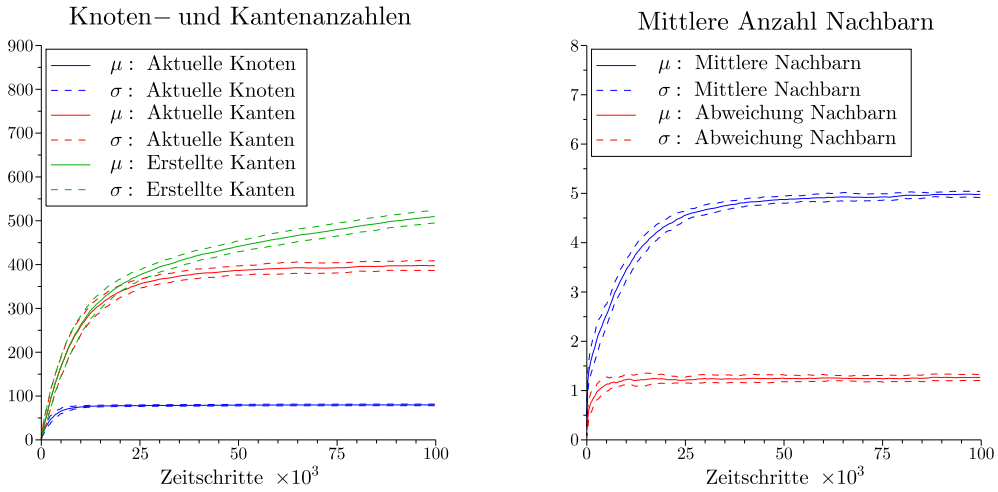

Abbildung 5.1: Mittelwerte und Standardabweichungen fünf unterschiedlicher Messgrößen: Die Grafiken weisen einen sehr kontinuierlichen Verlauf mit kleinen Standardabweichungen auf.

Abbildung 5.2 auf der nächsten Seite dargestellt. Weiterhin ist der Abstand relativ zu $\Phi(x)$ nach oben und unten angegeben. Die emprisch ermittelte WDF verlässt den so aufgespannten Bereich innerhalb des 5%-Quantils nicht und somit kann die Nullhypothese H_0 akzeptiert werden.

Allgemein verlässt keine empirisch ermittelte WDF das jeweilige 5%-Quantil, womit jeder Messgröße wahrscheinlich eine Normalverteilung zugrunde liegt. Unter dieser Annahme kann die Aussage getroffen werden, dass 99,7% aller möglichen Ergebnisse für jeden Parameter in einem Bereich von maximal $\pm 0.15\mu$ liegen. Für die ersten drei Parameter zeigt Tabelle 5.1 auf der vorherigen Seite, dass die Bereiche für die absoluten Zahlen sogar um $\pm 0.07\mu$ liegen. Das bedeutet konkret, dass beispielsweise die Anzahl an Knoten mit einer Wahrscheinlichkeit von 99,7% im Wertebreich $[75.106, 86.17]$ liegt.

Diese Schwankungen in den Messgrößen zeigen, dass auch einzelne Simulationen aussagekräftig sind, da die wesentliche Struktur erkennbar und die genaue Anzahl an einzelnen Elementen für spätere Tests nicht entscheidend ist. In den folgenden Experimenten werden daher die Stichproben deutlich verkleinert.

5.2 Parameteranalyse zur Feldkonstruktion

In diesem Experiment werden die vorhandenen Parameter zur Konstruktion der neuronalen Netze überprüft. In den meisten Arbeiten, wie beispielsweise in [Tou06], wird ein fester Parametersatz vorgegeben, der sich wahrscheinlich über die Entwicklungszeit des Algorithmus hin bewährt hat. Ein solcher Parametersatz wurde beispielsweise auch im eben beschriebenen KSA verwendet.

Um festzulegen, welcher Parameter welchen Wert haben wird, ob Parameter in einem bestimmten Verhältnis zueinander stehen oder ob Parameter variabel sein sollten, wer-

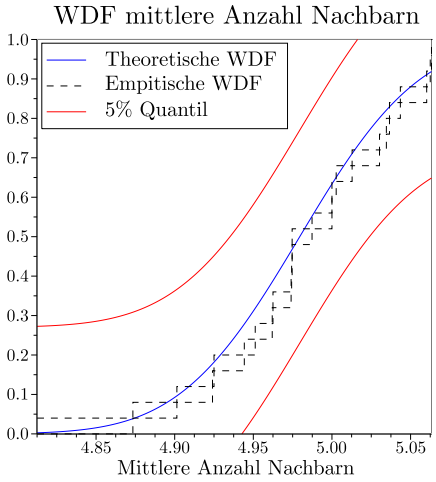

Abbildung 5.2: Theoretische Wahrscheinlichkeitsdichtefunktion (WDF) der Anzahl an Knoten (blau) anhand des empirischen Mittelwertes und Standardabweichung mit dem zugehörigen 5%-Quantil (rot). Weiterhin ist die empirische WDF (schwarz) über 25 Simulationen abgetragen, die das 5%-Quantil nicht verlässt.

den hier 640 Parametersätze überprüft und anschließend ein oder mehrere geeignete Parametersätze für die nächsten Experimente festgelegt.

Durchführung

Für diesen Test werden der ebene Kreis und das Gebirge herangezogen. Die zu untersuchenden Parameter sind in Tabelle 5.2 aufgeführt.

Parameterart		Parameterwert			
Aktualisierungsart Knoten		absolut		inkrementell	
Aktualisierung Knoten	τ_w	-		0.9	0.995
Aktualisierungsart Kanten		absolut		inkrementell	
Aktualisierung Kanten	τ_m	-		0.7	0.8
Aktualisierung Feldaktivität	τ_z	0.7	0.9	0.7	0.9
Grundaktivierung Feldaktivität	b_z	-0.05	0.05	-0.05	0.05
Lateraler Aktivierungsfaktor	η_z	0	0.1	0	0.1
Breite	σ_x	0.09	0.16	0.09	0.16
Aktualisierung Knotenfehler	τ_e	0.75	0.85	0.75	0.85
Vigilanz	h	0.55	0.75	0.55	0.75
Maximales Kantenalter	α_{max}	2	4	2	4

Tabelle 5.2: Alle für die Feldkonstruktion notwendigen Parameter sind mit den möglichen Belegungen aufgeführt und in zwei Gruppen eingeteilt. Die mittlere Spalte weist 128 mögliche Kombinationen der Parameter für eine absolute Aktualisierung der Knoten und Kanten auf. In der rechten Spalte ergeben sich 512 mögliche Kombinationen für eine inkrementelle Aktualisierung.

Alle für die Feldkonstruktion notwendigen Parameter sind mit den möglichen Belegungen in zwei Gruppen eingeteilt. Die insgesamt 640 Parametersätze teilen sich in 128 Kombinationen für die absolute Aktualisierung der Knoten und Kanten (mittlere Spalte) und in 512 Kombinationen für die inkrementelle Aktualisierung auf. Durch diese Aufteilung reduziert sich die Gesamtzahl der möglichen Parametersätze, da bei einer absoluten Aktualisierung keine besonderen Parameter τ_m und τ_w notwendig sind. Jeder Parametersatz wird mit beiden Welten je 10 mal je 50000 Zeitschritte berechnet.

Auswertung

Um die Datenmenge bei der Auswertung niedrig zu halten, wurde im Vorfeld von jedem Parametersatz eine repräsentative Simulation ausgewählt, deren Ergebnisse dem Mittelwert aller Ergebnisse dieses Parametersatzes am nächsten kamen. Weiterhin wurden für die folgenden Grafiken die Mittelwerte über 100 Zeitschritte der einzelnen Ergebniswerte berechnet.

Zunächst werden die Ergebnisse der absoluten Aktualisierung mit dem ebenen Kreis als Welt betrachtet. Der Verlauf aller 128 Repräsentanten für die folgenden aufgezeichneten Werte ist in Abbildung 5.3 dargestellt: Anzahl an Knoten und Kanten, mittlere Kantenlänge und Anzahl an Nachbarn.

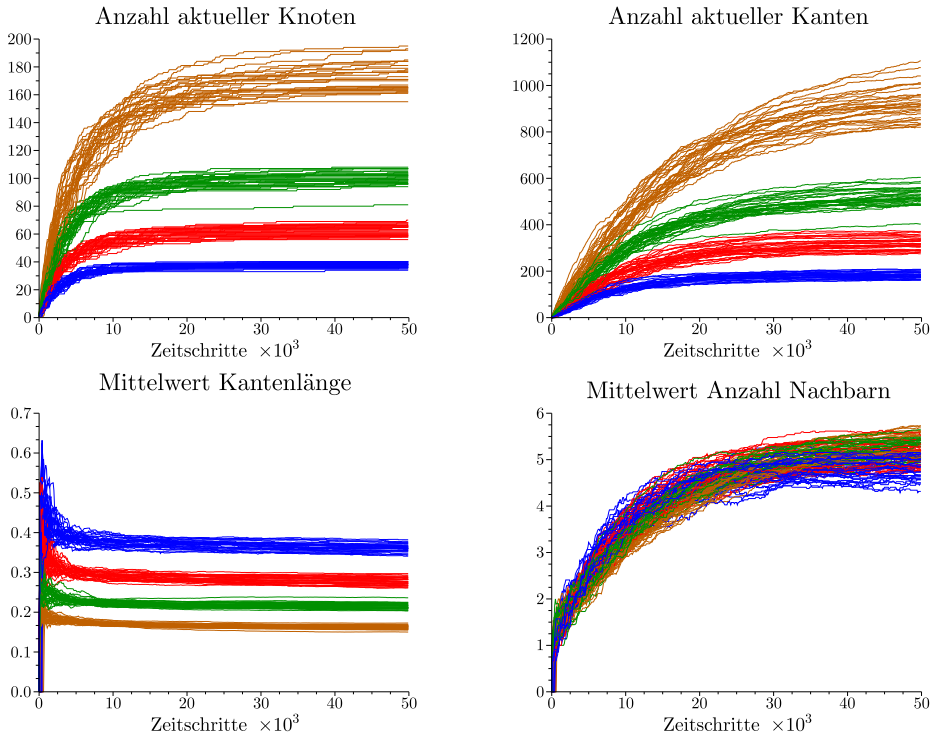

Abbildung 5.3: Darstellung eines Repräsentanten für jede absolute Parameterkombination der ebenen Welt: In den ersten drei Grafiken ist eine Separation in vier Kategorien erkennbar. Die Anzahl an Nachbarn weist einen Mittelwert von etwa 5 Nachbarn auf.

Auf allen vier Grafiken ist erkennbar, dass die einzelnen Linien sich einem Mittelwert nähern. Weiterhin ist in drei Grafiken eine Spaltung in vier Teilbereiche erkennbar. Diese vier Teilbereiche, im Folgenden Kategorien genannt, haben folgende Mittelwerte:

Kategorie	A	B	C	D
Anzahl Knoten	170	100	60	40
Anzahl Kanten	1000	500	300	190
Länge Kante	0.16	0.22	0.28	0.36

Tabelle 5.3: Einteilung der Testergebnisse in vier Kategorien. Zu jeder Kategorie sind die Mittelwerte der Anzahl an aktuellen Knoten und Kanten und die Länge der Kanten aufgeführt.

Interessant ist an dieser Stelle, welche Parameter einen Einfluss auf die Zugehörigkeit zu einer der vier Kategorien haben. Eine erste Prüfung ergab, dass das maximale Kantenalter keinen Einfluss auf die Zugehörigkeit zu einer bestimmten Gruppe hat. Das durchschnittliche Kantenalter liegt bei 0.5 und somit weit unter den beiden angegebenen Parametern von 2 und 4. Die übrigen 64 Ergebnisse sind in Tabelle Tabelle 5.4 aufgeführt.

	Vigilanz $h := 0.55$		Vigilanz $h := 0.75$	
Breite $\sigma_x := 0.09$	A A A A B A A A A A A A A A A A	B A A A A A A A A A A A A A A A	B B B B B B C B B B B B B B C B	B B B B B A B B B B B B B B C B
Breite $\sigma_x := 0.16$	B C C C B C C C C C C C C C C C	D C C C C C C A C C C C C A C C	D D D D D D D D D D D D D D D D	D D D D C D D D D D D D C D D D

Tabelle 5.4: Die Tabelle zeigt die übrigen 64 Zuordnungen zu einer Kategorie nach Entfernen des Alterskriteriums der Kanten. Im Wesentlichen sind für eine Zuordnung zu einer Gruppe die Vigilanz und der Aktivierungsradius verantwortlich.

Die Tabelle zeigt eine Gruppierung der Ergebnisse nach der Vigilanz und der Breite eines Knotens. Diese beiden Parameter sind maßgeblich für die Einordnung in eine Gruppe verantwortlich. Den stärksten Einfluss hat der Breite σ_x, da dieser nach Gleichung (2.20) auf Seite 13 quadratisch in die Gesamtaktivierung eines Knotens eingeht. Die Vigilanz ist die Obergrenze des Fehlers eines Knotens und hat somit direkt Einfluss auf die Entstehung eines neuen Knotens im Netz. Die übrigen Parameter dienen eher einer Feineinstellung.

Die Auswertung der 512 Parameterkombinationen für die inkrementelle Aktualisierung zeigt im Wesentlichen ähnliche Ergebnisse. Die Grafiken für die gleichen vier Variablen zur Auswertung sind in Abbildung 5.4 auf der nächsten Seite dargestellt.

Einen wesentlichen Unterschied zum vorher betrachteten Ergebnis zeigt die vierte Grafik mit der Auswertung des Mittelwertes der Nachbarn eines jeden Knotens. Hier sind deutlich zwei Tendenzen erkennbar: Der eine Teil der Linien nähert sich wie in Abbil-

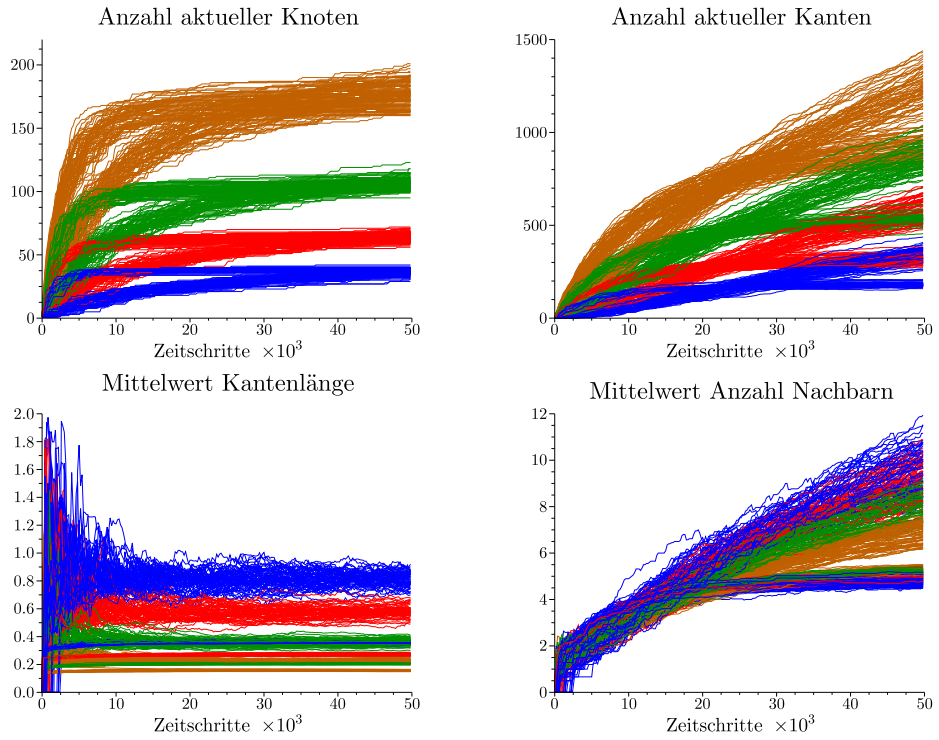

Abbildung 5.4: Darstellung eines Repräsentanten für jede inkrementelle Parameterkombination der ebenen Welt: Nur bei den Knotenzahlen sind die vier Kategorien noch zu erkennen. Die Anzahl an Nachbarn zeigt einen schlechten Verlauf, da Werte über 6 angenommen werden. Diese Grenze stellt eine optimale Struktur des Netzes aus gleichseitigen Dreiecken dar.

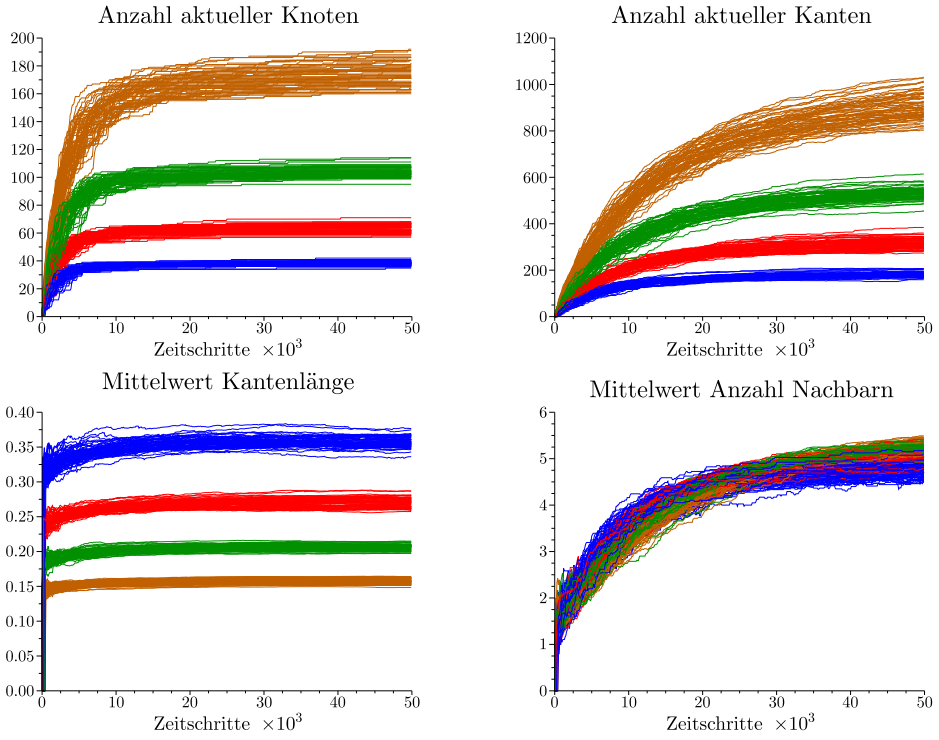

Abbildung 5.5: Darstellung eines Repräsentanten für die bereinigten inkrementelle Parameterkombination der ebenen Welt: Die Grafiken zeigen nun qualitativ den gleichen Verlauf wie in Abbildung 5.3 auf Seite 74

dung 5.3 auf Seite 74 einem Mittelwert von ca. 5.5 Nachbarn pro Knoten. Die andere Gruppe weist noch keinen waagerechten Verlauf auf und viele der Werte liegen bereits deutlich über 6. Dies ist eine wichtige Grenze, da eine optimale Gitteraufteilung durch genau sechs Nachbarn realisiert wird. Jedes Dreiertupel an benachbarten Knoten spannt dann ein gleichseitiges Dreieck auf, was die optimale Form des neuronalen Netzes darstellt. Höhere Werte weisen auf starke Unregelmäßigkeiten und einen hohen Grad an Überschneidungen hin. Ein Beispiel ist in Abbildung 5.6 dargestellt.

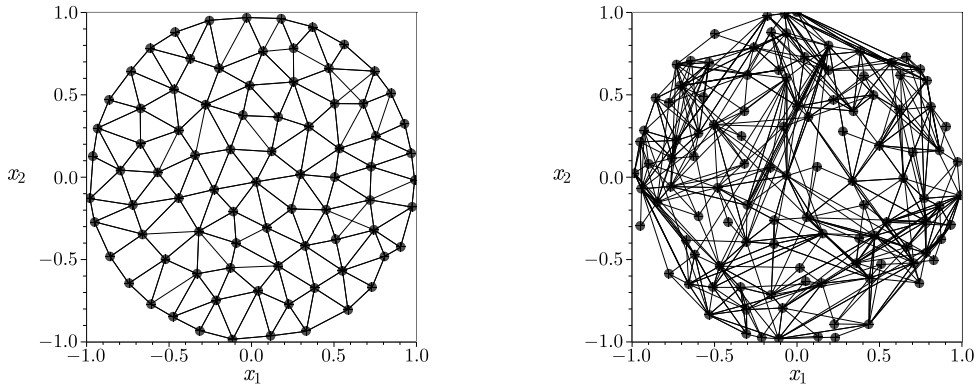

Abbildung 5.6: Darstellung von zwei explorierten neuronalen Netzen: (links) $\tau_w := 0.995$: Es baut sich eine regelmäßige Netzstruktur auf; (rechts) $\tau_w := 0.9$: Es zeigt sich keine Regelmäßigkeit in der Anordnung der Knoten und es gibt viele Überschneidungen der Kanten. Die Überschneidungen resultieren aus der starken Anpassung der Knoten.

Verantwortlich für die unkontrollierte Netzstruktur ist der Aktualisierungsparameter eines Knotens τ_w mit der Belegung 0.9 im Vergleich zu 0.995. Die kleinere Belegung weist eine höhere Netzdynamik auf. Jeder Gewinnerknoten wird sehr stark in Richtung des aktuellen Messwertes im sensorischen Raum verschoben. Diese starken Verschiebungen führen zu den unregelmäßigen Netzstrukturen. Nach einer Bereinigung der Testergebnisse um diesen Parameterwert ergeben sich die Grafiken in Abbildung 5.5 auf der vorherigen Seite, die wieder eine klare Einteilung in die vier Ergebniskategorien wie in Abbildung 5.3 auf Seite 74 zulassen.

Die restlichen 256 Testergebnisse nach der Bereinigung lassen sich wie in Tabelle 5.4 auf Seite 75 kategorisieren. Somit zeigt sich, dass die wesentlichen Parameter für die Netzstruktur der Breite σ_x und die Vigilanz h sind. Da es auf die Netzstruktur keinen Einfluss hat, ob die Knoten inkrementell oder absolut aktualisiert werden, wird auf die inkrementelle Variante zurückgegriffen. Diese hat den Vorteil, dass sie lebenslanges Lernen ermöglicht.

Die Auswertung der Testergebnisse für das Gebirge als Welt zeigt im Wesentlichen das gleiche Ergebnis. Einzig die Standardabweichungen sind größer, da durch die unterschiedlichen Steigungen in der Welt die einzelnen Knoten unterschiedliche Distanzen haben. Folglich ist eine größere Varianz bereits innerhalb einer einzelnen Simulation und somit auch für das Gesamtergebnis. Aufgrund der hier gefundenen Ergebnisse wird für alle weiteren Experimente der Standardparametersatz mit inkrementeller Aktualisierung in Tabelle 5.5 auf der nächsten Seite verwendet.

τ_w	τ_m	τ_z	b_z	η_z	σ_x	τ_e	h	α_{max}
0.995	0.85	0.9	0.05	0.1	0.13	0.8	0.6	2

Tabelle 5.5: Standardparametersatz für die nachfolgenden Experimente. Es wird auf die inkrementelle Aktualisierung zurückgegriffen.

5.3 Vergleich der aktiven Motorsteuerung und der homöokinetische Regelung

In diesem Abschnitt werden die explorierten neuronalen Netze der beiden Bewegungsvarianten gegenübergestellt. Der Vergleich basiert auf den beiden Welten aus der Simulation und einem Aufbau mit einem Arm eines M-Serie Roboters. Neben den Eigenschaften der neuronalen Netze wird auch auf die Dauer der Erstellung eingegangen.

Durchführung

Es wird für jeden Versuchsaufbau eine Testreihen mit jeweils beiden Bewegungsvarianten durchgeführt. Jede Testreihe besteht aus drei Simulationsläufen mit jeweils 100000 Zeitschritten. Für die Konstruktion der Netze wird einheitlich der im letzten Abschnitt definierte Parametersatz in Tabelle 5.5 verwendet. Während der Simulationen werden die gleichen Messgrößen wie im vorherigen Experiment protokolliert.

Auswertung

Zunächst erfolgt die Auswertung des ebenen Kreises als Welt. Einige Messwerte der entstehenden neuronalen Netze sind in Abbildung 5.7 auf der nächsten Seite dargestellt.

Die Eigenschaften der aktiven Motorsteuerung (NF) sind in roten Farben und die der Homöokinese (HK) in blauen und grünen Farben gekennzeichnet. Allgemein ist erkennbar, dass die Graphen der Homöokinese langsamer ansteigen, als die der aktiven Motorsteuerung. Dies ist auf die Aktualisierung der Gewichtsmatrix $\boldsymbol{W}_c$ nach Gleichung (3.39) auf Seite 48 zurückzuführen. Auf Möglichkeiten der Anpassung der Lernraten wird im letzten Teilabschnitt eingegangen.

Auf der linken Abbildung ist erkennbar, dass nach etwa 85000 Zeitschritten die Anzahl der vorhandenen Knoten gleich ist. Allerdings existieren am Ende des Versuches der Homöokinese nicht einmal die Hälfte an Kanten im Vergleich zur aktiven Motorsteuerung. Außerdem wurde keine einzige Kante gelöscht. Diese beiden Tatsachen zeigen, dass zwar die Gesamtheit des Raumes exploriert worden ist, dies jedoch meist auf den gleichen Pfaden geschah. Dies wird durch die rechte Abbildung gestützt in der klar erkennbar ist, dass bei der Homöokinese jeder Knoten im Mittel weniger als zwei Nachbarn besitzt.

Wird die zweite Testreihe betrachtet, zeigen sich ähnliche Resultate. Weiterhin wurde hier das Verhältnis aus rücktreibender Kraft des Gebirges und Ansteuerung auf den

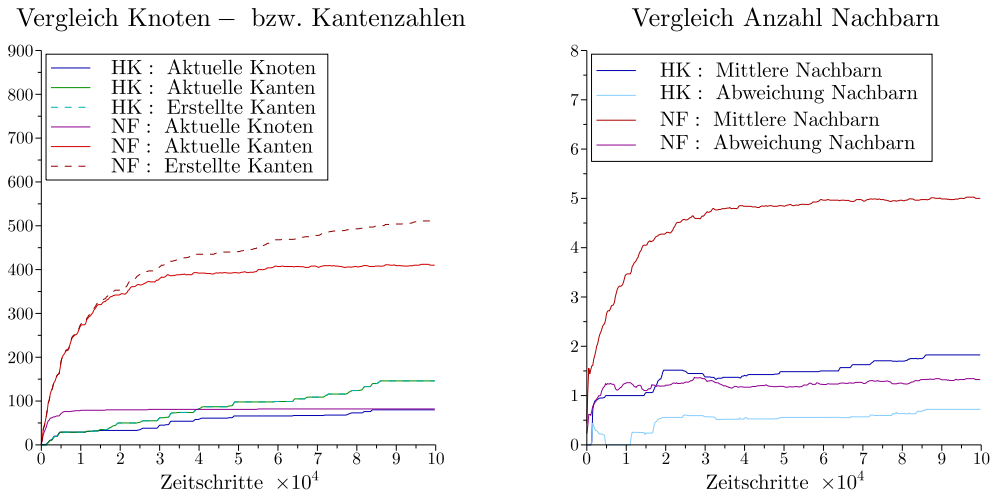

Abbildung 5.7: Vergleich der Eigenschaften der Netze unter Verwendung der aktiven Motorsteuerung (NF) beziehungsweise der homöokinetischen Regelung(HK): (links) Die drei Graphen zu (NF) zeigen den bereits bekannten Verlauf aus den vorherigen Experimenten. Für die Homöokinese ist die Anzahl der Knoten nach 85000 Zeitschritten ungefähr gleich, wobei die Anzahl der Kanten deutlich geringer ist. (rechts) Die deutlich geringere Anzahl an Kanten führt zu einem schwach verbundenen Feld. Im Schnitt hat jeder Knoten bei der Homöokinese weniger als zwei Nachbarn.

Motor so eingestellt, dass die Hügel nur bei einer Ansteuerung von mindestens 80% der maximalen Motorkraft überquert werden können. Die aktive Motorsteuerung überquert die Hügel im Laufe von 100000 Zeitschritten mehrmals. Bei der Verwendung der Homöokinese wurde eine Überquerung der Hügel bei dieser Einstellung nicht erreicht. Somit blieben wesentliche Teilgebiete des sensorischen Raumes unerforscht.

Für die Testreihe an einem Roboter der M-Serie wurde ein Arm, ohne das rotierende Schultergelenk, verwendet. Der Arm wurde, wie in Abbildung 5.8 auf der nächsten Seite dargestellt, senkrecht nach oben auf eine Haltevorrichtung montiert. Das entstehende zweidimensionale System weist eine instabile Ruhemenge und eine stabile Ruhemenge auf. Die beiden Bewegungsvarianten zeigen hier ähnliche Ergebnisse. Jedoch explorieren beide Varianten den Raum sehr spärlich.

Dieses Experiment führt im Wesentlichen zu zwei Erkenntnissen:

- Die homöokinetische Exploration benötigt mehr Zeit zur Exploration.
- Beide Bewegungsvarianten explorieren einen realen Sensorraum nur spärlich.

Speziell Abbildung 5.7 bekräftigt die erste Erkenntnis. Auch nach 500000 Zeitschritten ist die Anzahl der entstandenen Kanten bei der Homöokinese erst bei 70% aller Kanten bei der aktiven Motorsteuerung. Dies liegt an der langsamen Anpassung der Gewichtsmatrizen $\boldsymbol{W}_a$ und $\boldsymbol{W}_c$. Die verwendeten Lernraten können zwar angepasst werden, jedoch ergibt sich dann ein „Übersteuern“: Die Gewichtsmatrizen werden zu stark angepasst und diese Anpassung wird in den nächsten Zeitschritten umgekehrt. Dieses Verhalten führt zu einem ständigen Schwingen der Matrizen und dementsprechend auch

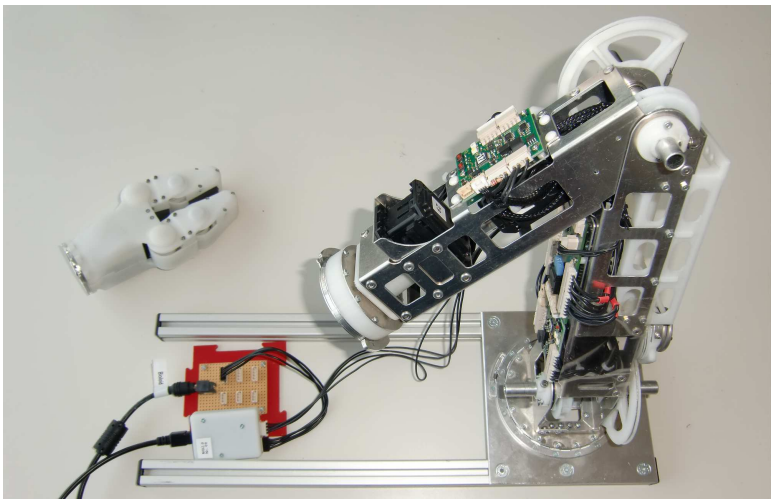

Abbildung 5.8: Aufbau des linken Armes eines M-Serie Roboters: Es sind der Arm in einer Haltevorrichtung, der Greifer und die notwendigen Anschlüsse für die Kommunikation mit dem PC abgebildet.

zu einer ständigen Schwingung in der Motoransteuerung. Pendeln die Werte zusätzlich zwischen positiven und negativen Werten, so wird ständig die Drehrichtung der Motoren verändert. Um diesem unnötigen Verschleiß vorzubeugen, müssen die Lernraten entsprechend angepasst werden, damit eine kontinuierliche Veränderung ohne Schwingung entsteht. Somit benötigt die Homöokinese um genauso viel Information wie die aktive Motorsteuerung zu explorieren, deutlich mehr Zeit.

Die zweite Erkenntnis, dass die realen Räume spärlich exploriert werden, zeigt sich bei der Homöokinese bereits bei der simulierten Welt des Gebirges. So werden hier zwei der vier versteckten Bereiche nicht gefunden, obwohl sie erreichbar sind. Bei dem Umstieg von der Simulation an einen Roboter der M-Serie ist das Verhalten der Motoren entscheidend. In der Simulation wird die Motorenergie linear in eine Änderung des Zustandes in der Welt überführt. Dies wird durch Gleichung (4.2) auf Seite 67 und Gleichung (4.7) auf Seite 68 realisiert. Die Servomotoren der M-Serie Roboter weisen folgendes Verhalten auf:

1. Die Kraftübertragung ist nicht linear.
2. Zwischen dem Servomotor und dem Gelenk ist eine Federkupplung verbaut.
3. Die Hsltekraft des Servomotors ist etwa fünf mal so groß wie die Aktionskraft

Die Bauteile und deren Verhalten führen dazu, dass bei beiden Bewegungsvarianten ähnliche Eigenschaften auftreten. Sehr viele Ansteuerungen führen zu einer Stabilisierung einer Position, da die Energie nicht ausreicht um eine Bewegung durchzuführen. Ist die Kraft groß genug, so wird der Sensorraum innerhalb weniger Sekunden komplett durchschritten. Die resultierende Bewegung führt annähernd zu einer Geraden im sensorischen Raum. Dadurch wird der sensorische Raum spärlich exploriert. Es entstehen also wenige Kanten und demnach auch wenige Möglichkeiten einer späteren Planung.

5.4 Selbstschutz durch RBF-Neuronen

Die Funktionalität des Selbstschutzes wurde in drei Schritten untersucht:

1. Festlegung des Wertebereiches des Lernsignals
2. Offline-Lernen der RBF-Neuronen auf Daten eines M-Serie Roboters.
3. Online-Lernen der RBF-Neuronen während der Exploration in der Simulation.

In einem ersten Schritt wird die offene Fragestellung aus Abschnitt 3.6.1 ab Seite 54 nach dem Wertebereich des Lernsignals für das überwachte Lernen beantwortet. Anhand dieses gefundenen Lernsignals wird im zweiten Schritt untersucht, ob das Lernen der RBF-Neuronen erfolgreich ist. Da dieser Schritt auf der Grundlage von bereits aufgenommenen Daten durchgeführt wird, kann keine Rückkopplung des Ausgangsvektors $\boldsymbol{\zeta}$ nach Gleichung (3.44) auf Seite 55 der RBF-Neuronen erfolgen. Diese Rückkopplung ist Teil des dritten Schrittes. Hier wird das Experiment aus [HDH09] in ähnlicher Weise wiederholt.

5.4.1 Festlegung des Wertebereiches des Lernsignals

In diesem Vorexperiment wird untersucht, aus welchem Wertebereich das Lernsignal für den Gradientenabstieg sein sollte. Für den Gradientenabstieg werden die Gleichungen (2.32), (2.35), (2.36) und (2.37) für RBF-Neuronen aus Abschnitt 2.4.1 ab Seite 18 verwendet. Das Ausgangssignal eines RBF-Neurons ist im Intervall $]0, 1]$ definiert. Das Lernsignal für den Gradientenabstieg ist die Abweichung der geschätzten von der gemessenen Geschwindigkeit. Wird kein Hindernis erreicht, so wird dieser Fehler meist nahe 0 sein. Wird ein Hindernis erreicht, kann der maximale Fehler in einer Umkehrung der gemessenen Geschwindigkeit, beispielsweise durch eine Feder, auftreten. Jedoch wird dieser Fehler deutlich unter dem Wert 1 liegen, so dass eine Skalierung des Fehlers ohnehin notwendig ist.

Durchführung

Für dieses Vorexperiment wird ein eindimensionales dynamisches System betrachtet, das im geschlossenen Intervall $[-1, 1]$ definiert ist. Das dynamische System wird mit der homöokinetischen Regelung exploriert. Für das Lernverfahren des RBF-Neurons wird der entstehende Fehler in folgende Wertebereiche skaliert:

Kategorie	A	B	C	D	E
Wertebereich	$[0.0, 1.0]$	$[0.1, 0.9]$	$[0.2, 0.8]$	$[0.3, 0.7]$	$[0.4, 0.6]$

Tabelle 5.6: Wertebereiche des Lernsignals für den Gradientenabstieg eines RBF-Neurons

Die Skalierung erfolgt über eine ständige Anpassung der Ober- und Untergrenze des Fehlers. Für jeden Wertebereich werden zehn Experimente mit jeweils

35000 Zeitschritten berechnet. Anschließend werden der Erwartungswert und die Standardabweichung über die letzten 25000 Zeitschritte jedes Experimentes für den Gewichtsvektor und die Breite des RBF-Neurons berechnet. Aus diesen Werten wird anschließend über die zehn Experimente jeweils ein Mittelwert gebildet.

Auswertung

Das Vorexperiment mit der anschließenden Mittelwertbildung führt zu den in Tabelle 5.7 dargestellten Ergebnissen.

Kategorie	A	B	C	D	E
$\mu(w_V)$	-0.028	-0.011	-0.039	0.005	0.008
$\sigma(w_V)$	0.048	0.029	0.035	0.018	0.034
$\mu(\sigma_V)$	1.240	1.181	1.092	1.034	0.924
$\sigma(\sigma_V)$	0.098	0.067	0.071	0.057	0.076

Tabelle 5.7: Auswertung der Wertebereiche des Lernsignals: Die geringste Abweichung von den erwarteten Werten weist der Wertebereich D auf. Weiterhin hat dieser auch die kleinsten Standardabweichungen

Alle Wertebereiche führen zu guten Ergebnissen. Die Abweichungen der gefundenen Mittelwerte für den Gewichtswert $\mu(w_V)$ von der Vorgabe sind immer kleiner als 4%. Bei den Mittelwerten $\mu(\sigma_V)$ ist erkennbar, dass sich diese zusammen mit dem Wertebereich verkleinern. Diese Tendenz muss für die Rückkopplung der Ausgangswerte der RBF-Neuronen beachtet werden. Dies betrifft vor allem Gleichung (3.43) auf Seite 55 und Gleichung (3.49) auf Seite 57, da hier die Ausgangswerte der Neuronen eingehen. Werden die Standardabweichungen der Mittelwerte betrachtet, so weist der Wertebereich D die kleinsten Werte auf. Aufgrund dessen wird in den folgenden Abschnitten dieser Wertebereich zugrunde gelegt.

5.4.2 Offline-Lernen der RBF-Neuronen

Im zweiten Schritt werden zwei RBF-Neuronen auf drei aufgenommenen Datensätzen eines M-Serie Roboters trainiert. Der Vorteil ist, dass hier der Fehlerwert $\boldsymbol{\xi}$ optimal in den Wertebereich D skaliert werden kann, da bereits alle Sensordaten vorliegen.

Durchführung

Für die Aufnahmen der Datensätze wurde der linke Arm verwendet. Der Aufbau entspricht der Abbildung 5.8 auf Seite 80. Die beiden Dimensionen des Systems sind der Winkel des Schulter- und des Ellenbogengelenkes. Von den Aufnahmen werden jeweils die ersten 35000 Zeitschritte für das Lernen verwendet. Zusätzlich wurde der Beuge- und der Streckanschlag der Gelenke für eine Skalierung notiert.

Auswertung

Für die Auswertung wurden die Wertebereiche der Winkel in das Intervall $[-1, 1]$ skaliert. Dadurch sind die folgenden Ergebnisse mit den Ergebnissen des Vorexperimentes vergleichbar. Die Auswertung der drei Aufnahmen ist in Tabelle 5.8 dargestellt.

Datensatz	A	B	C
$\mu(w_V)$	-0.011	-0.007	0.015
$\sigma(w_V)$	0.024	0.029	0.018
$\mu(\sigma_V)$	1.096	1.075	1.014
$\sigma(\sigma_V)$	0.079	0.084	0.078

Tabelle 5.8: Auswertung des Offline-Lernens: Alle drei Datensätze weisen ähnliche Ergebnisse zu Tabelle 5.7 auf der vorherigen Seite auf.

Wie bei dem Vorexperiment wurden über die letzten 25000 Zeitschritte die Mittelwerte und Standardabweichungen des Gewichtswertes und der Breite bestimmt. Alle drei Datensätze zeigen, dass kaum eine Abweichung der Ergebnisse untereinander und zum Vorexperiment vorhanden ist.

5.4.3 Online-Lernen der RBF-Neuronen

Nach dem erfolgreichen Lernen der RBF-Neuronen auf aufgenommenen Daten wird nun untersucht, ob eine Beeinflussung der Richtungswahl möglich ist, so dass die Anschläge der Gelenke vermieden werden. Für dieses Experiment wird das eindimensionale System aus dem Vorexperiment herangezogen.

Durchführung

Die Aktionsauswahl im System erfolgt hier zum einen über die homöokinetische Regelung und zum anderen über die aktive Motorsteuerung. Die Steuerung wird um den Prädiktor aus der homöokinetischen Regelung erweitert, da eine Vorhersage des nächsten Zustandes für das Lernen der RBF-Neuronen notwendig ist.

Es werden mit beiden Varianten jeweils zwanzig Experimente durchgeführt. Davon sind jeweils zehn Läufe mit und ohne Rückkopplung des Ausgangswertes des RBF-Neurons. Jedes Experiment dauert 35000 Zeitschritte. Für jeden Lauf wird ab dem Zeitschritt 10000 die absolute Anzahl an Positionen bestimmt, in denen das Gelenk einen Anschlag erreicht hat und weiterhin in diese Richtung strebt.

Auswertung

Aus den jeweils zehn Experimenten wurde der Mittelwert μ_G und die Standardabweichung σ_G der Gelenkanschläge berechnet. Die Ergebnisse sind in Tabelle 5.9 auf

der nächsten Seite dargestellt. Durch die Rückkopplung des Ausgangswertes des RBF-Neurons konnte die Gesamtzahl der Gelenkanschläge um 67% bei der aktiven und um 45% bei der homöokinetischen Aktionauswahl verringert werden. Die Standardabweichungen sind ebenfalls niedriger, wobei dieser Reduzierung weniger Gewicht beizumessen ist.

Aktionsauswahl	Rückkopplung	μ_G	σ_G
Aktive Motorsteuerung	nein	2589	340.5
	ja	848	217.5
Homöokinese	nein	2271	270.3
	ja	1246	143.8

Tabelle 5.9: Auswertung des Online-Lernens: Reduzierung des Mittelwertes der absoluten Anzahl an Gelenkanschlägen μ_G durch die Rückkopplung des Ausgangssignals des RBF-Neurons um 67% bei der aktiven und um 45% bei der homöokinetischen Aktionauswahl.

Dieser Test zeigt, dass nicht nur die Gelenkanschläge gelernt werden konnten, sondern dass auch bei beiden Ansteuerungen der Motoren die Anzahl der Anschläge durch eine geeignete Rückführung des Ausgangssignals des RBF-Neurons reduziert werden konnte. Diese Eigenschaft kann maßgeblich zu einer schonenden Exploration beitragen.

5.5 Analyse der unterschiedlicher Planungsmöglichkeiten

Nachdem die ersten Experimente den Aufbau des neuronalen Netzes und das Explorationsverhalten betrachteten, wird hier ein exploriertes Netzwerk auf die Möglichkeiten der Planung hin untersucht. Es werden die Eigenschaften der verschiedenen Planungsalgorithmen auf dieses Netz in Bezug auf Erfolg, Zeitdauer und Energieverbrauch betrachtet.

Durchführung

Gemeinsame Basis für dieses Experiment ist ein neuronales Netz nach 100000 Zeitschritten in der simulierten Welt des ebenen Kreises. Jeder Planungsalgorithmus startet mit diesem neuronalen Netz und durchläuft einen festgelegten Zyklus 1000 mal. Dieser Zyklus und das neuronale Netz sind in Abbildung 5.9 auf der nächsten Seite dargestellt.

Sollte der nächste Knoten innerhalb eines Zyklus nicht nach spätestens 500 Zeitschritten erreicht sein, so gilt dieser Teilpfad als nicht erfolgreich und der nächste Knoten im Zyklus wird als Ziel ausgewählt. Nach jedem Wechsel im nächsten Zielknoten werden die benötigte Zeit und die verbrauchte Energie gespeichert.

Die Planungsalgorithmen unterliegen nicht einem Zufallszahlengenerator, sondern sind deterministische Algorithmen. Einzig das Rauschen der Sensorwerte kann zu einem Unterschied der Ergebnisse bei dem gleichen Algorithmus führen. Aus diesem Grund werden jeweils fünf Testläufe für jeden Planungsalgorithmus berechnet.

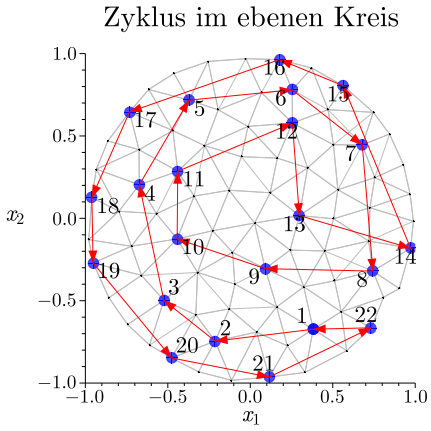

Abbildung 5.9: Der rote Pfad zeigt den Zyklus über ausgewählte Knoten des neuronalen Netzes. Das gesamte Netz ist im Hintergrund grau dargestellt.

Die verwendeten Planungsalgorithmen sind zum einen der ursprüngliche Algorithmus wie er in [Tou04] und [Tou06] verwendet wird und in Abschnitt 3.2.2 ab Seite 37 beschrieben ist und zum anderen der neu entwickelte Planungsalgorithmus unter Verwendung der einzelnen neuronalen Netze des motorischen Raumes an jeder Kante. Hier werden drei Varianten verwendet, die jeweils eine andere Priorität in der Planung haben:

- $\psi^{mask} := (1 \quad 0)^T$: energieeffiziente Planung
- $\psi^{mask} := (0 \quad 1)^T$: zeiteffieziente Planung
- $\psi^{mask} := (1 \quad 1)^T$: kombinierte Planung

Der Zielvektor ist für alle Varianten $\psi_g := \mathbf{0}$, so dass minimale Werte für die Kriterien angestrebt werden.

Auswertung

Bei der Auswertung der wird die Farbgebung zu den einzelnen Planungsmethoden konsistent gehalten. Den einzelnen Varianten sind folgende Farben zugeordnet:

- schwarz: ursprünglicher Algorithmus
- blau: kombinierte Planung
- rot: zeiteffieziente Planung
- grün: energieeffiziente Planung

In Abbildung 5.10 auf der nächsten Seite sind der Energie- und Zeitaufwand jedes einzelnen Zyklus der vier Varianten dargestellt. Die Planung mit dem ursprünglichen Algorithmus ist nahezu immer gleich, da sich das zugrunde liegende Netz nicht ändert.

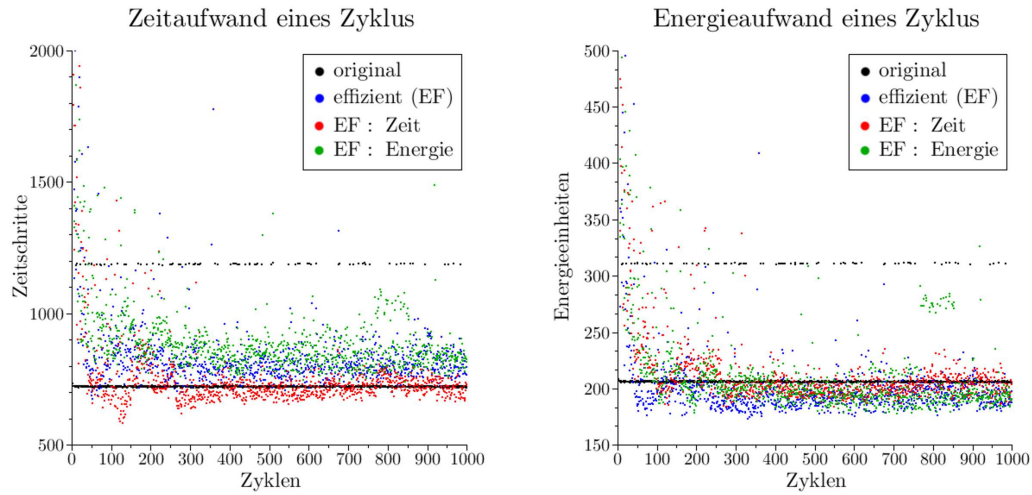

Abbildung 5.10: Zeit- und Energieaufwand jedes einzelnen Zyklus: (links) Der zeiteffiziente Algorithmus weist niedrigere Werte gegenüber dem ursprünglichen Algorithmus auf. (rechts) Nahezu jeder Zyklus der effizienten Planungsvarianten benötigt weniger Energie als die ursprüngliche Variante.

Nur wenn eine bestimmte Konstellation im Netz erreicht wird scheitert der Algorithmus. Die anderen drei Varianten der effizienten Planung werden über die Zeit hinweg besser. Hier werden nach und nach nicht erfolgreiche Knoten aus dem Netz der Motoraktionen gelöscht. Dadurch wird die Planung erfolgreicher und die Durchführung eines Zyklus benötigt weniger Zeit und Energie.

Wird speziell die Zeit betrachtet (linke Grafik), so kann eine klare Trennung der drei effizienten Algorithmen durchgeführt werden. Es ist die Priorität der einzelnen Varianten klar erkennbar. Je wichtiger die zeitliche Komponente bei der Planung ist, desto geringer sind die benötigten Zeitschritte. Die Grafik des Energieverbrauches (rechts) zeigt keine deutliche Trennung, jedoch werden alle drei Varianten meistens besser als der ursprüngliche Algorithmus.

In Abbildung 5.12 auf Seite 88 sind für jeden einzelnen Übergang des Zyklus der Mittelwert und die einfache Standardabweichung des Zeit- und Energieverbrauches dargestellt. Die geringen Abweichungen in der Planung des ursprünglichen Algorithmus zeigen sich auch hier wieder. Die effizienten Planungsalgorithmen weisen deutlich höhere Standardabweichungen auf. Werden die einzelnen Übergänge mit hohen Standardabweichungen mit dem Zyklus in Abbildung 5.9 auf der vorherigen Seite verglichen, so zeigt sich, dass die hohen Abweichungen dort auftreten, wo es keine offensichtlichen Kanten entlang des Zyklus gibt. Übergänge wie $9 \rightarrow 10$ oder $14 \rightarrow 15$ zeichnen sich dadurch aus, dass die kürzeste Verbindung nahezu senkrecht auf eine Kante des Netzes trifft. Da die effizienten Algorithmen keine Mittelwerte bei der Planung bilden, muss hier ein Umweg in Kauf genommen werden. Der ursprüngliche Algorithmus hat hier keine Probleme, da die Mittelwerte aus den nächsten Motoraktionen der Kanten gebildet werden.

Zusammenfassend werden die Ergebnisse in Abbildung 5.11 auf der nächsten Seite in einem Zeit-Energie-Diagramm dargestellt. Die Pfeile deuten an, in welche Richtung sich der Verbrauch über die Zeit hinweg entwickelt. Links ist das gesamte Diagramm und rechts der Ausschnitt des Verbrauches der letzten Zyklen dargestellt. Es wurden jeweils 100 Zyklen zusammengefasst und somit zehn Werte für die Grafik pro Planungsalgorithmus berechnet.

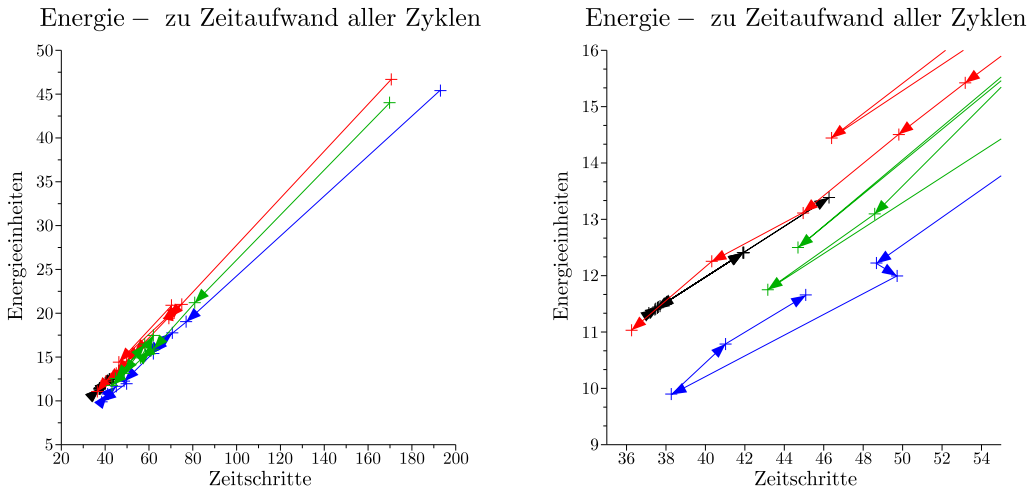

Abbildung 5.11: Energie-Zeit-Diagramm der vier Planungsalgorithmen: (links) gesamtes Diagramm; (rechts) Ausschnitt der letzten Zyklen; Der ursprüngliche Planungsalgorithmus ist zeitlich gleich oder besser, jedoch wird mehr Energie benötigt

Die effizienten Algorithmen zeigen einen unregelmäßigeren Verlauf als der ursprüngliche Algorithmus und verschlechtern sich auch wieder. Der zeiteffiziente Algorithmus ist dem ursprünglichsten Algorithmus am ähnlichsten. Die beiden anderen Varianten zeichnen sich durch einen niedrigeren Energieverbrauch aus.

Dieses Experiment führte zu folgenden Ergebnissen:

- Die effizienten Planungsalgorithmen benötigen weniger Energie als der ursprüngliche Algorithmus.
- Die effizienten Algorithmen korrigieren das Netz nach der Explorationsphase.

Das Experiment hat gezeigt, dass die effizienten Planungsalgorithmen einen niedrigeren Energieverbrauch aufweisen. Die zusätzlichen Informationen an den Neuronen, die eine Motoraktion speichern machen dies möglich.

Weiterhin wurde gezeigt, dass eine Nachoptimierung des Netzes während der Planung und Ausführung erfolgen kann. Motoraktionen, die selten erfolgreich sind, oder deren Zusatzinformationen von den wirklich benötigten abweichen, werden aus dem Netz der Motoraktionen gelöscht. Dadurch bleiben nur die passendsten Motoraktionen für einen entsprechenden Übergang übrig.

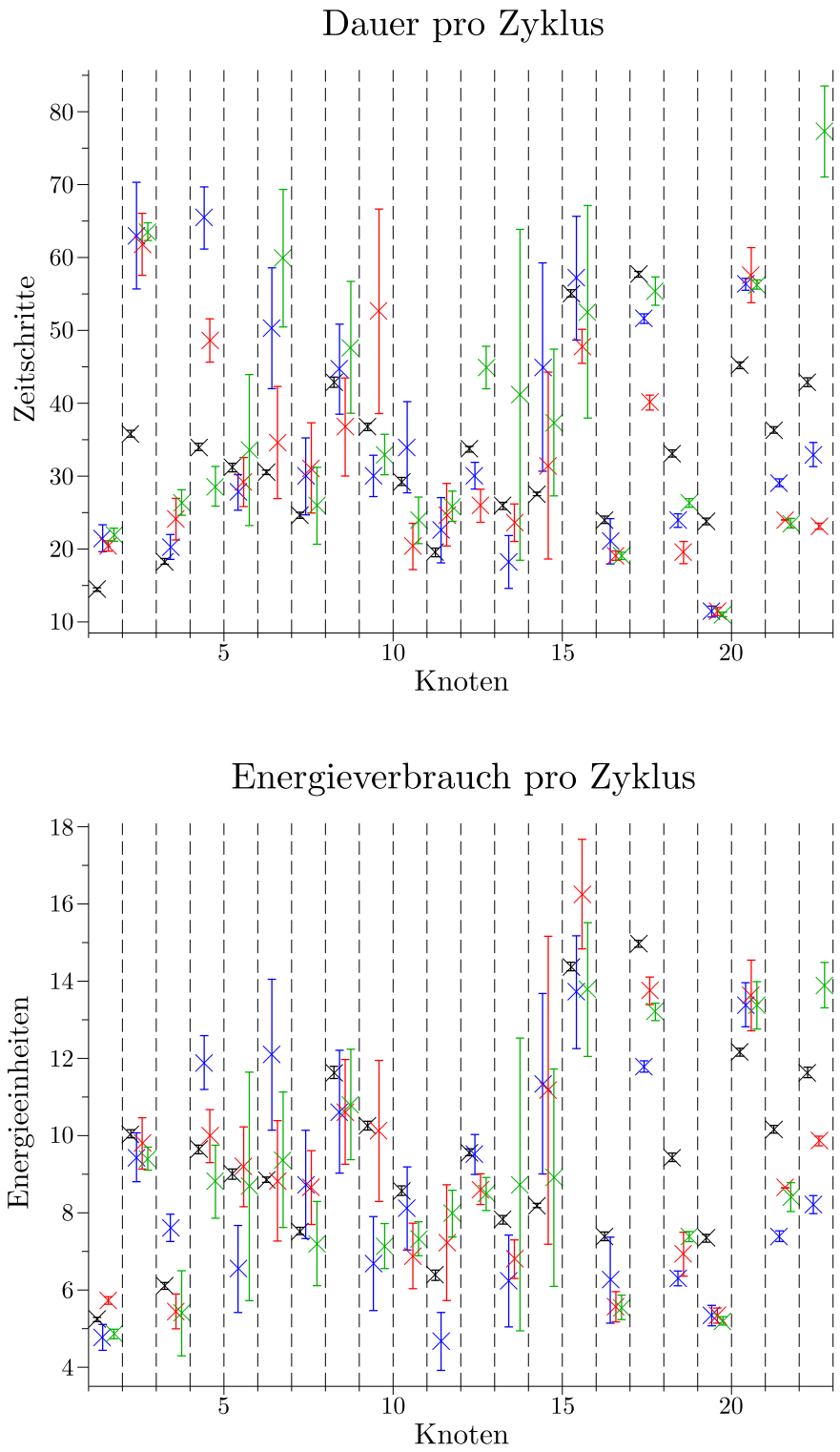

Abbildung 5.12: Mittelwert und einfache Standardabweichung für jeden einzelnen Übergang innerhalb der Zyklen für den Zeit- und Energieverbrauch: Der ursprüngliche Algorithmus weist kaum eine Standardabweichung auf, wohingegen die effizienten Algorithmen an bestimmten Übergängen große Abweichungen aufweisen.

Kapitel 6

Fazit und Ausblick

Intelligenz lässt sich nicht am Weg, sondern nur am Ergebnis feststellen.

Garri Kimowitsch Kasparow (*1963), russischer Schachgroßmeister

Abschließend werden in diesem Kapitel die Ergebnisse der Experimente mit Rückblick auf die vorherigen Kapitel zusammengefasst. Nach diesem Fazit werden Erweiterungsmöglichkeiten und andere Anwendungen des Algorithmus' vorgestellt.

6.1 Fazit

Das Experiment in Abschnitt 5.3 ab Seite 78, in dem die Auswahlmechanismen für die Motoraktion untersucht wurden, hat gezeigt, dass diese für die Roboter der M-Serie nicht gut geeignet sind. Die entstehenden Netze decken zwar den gesamten Sensorraum ab, jedoch weist jedes Neuron im Schnitt weniger als zwei Nachbarn auf. Weiterhin ist eine Exploration mit dem Roboter sehr langsam. Diese beiden Ergebnisse haben im Wesentlichen einen Grund: das Verhalten der Servomotoren. Diese haben ein sehr hohes Halte- aber ein geringes Aktionsdrehmoment. Das bedeutet, dass eine große Kraft aufgebracht werden muss, bis eine Aktion durchgeführt wird. Ist eine Aktion entstanden, so hat der Roboter den sensorischen Raum in wenigen Sekunden durchquert, so dass eine Änderung der Richtung kaum möglich ist. Der Sensorraum wird demnach meist gerade durchlaufen und so ergeben sich wenige Nachbarn an den Neuronen.

Dieses Verhalten ist der maßgebliche Grund dafür, warum im letzten Experiment der Planung in Abschnitt 5.5 ab Seite 84 keine Experimente mit einer Extremität des Roboters durchgeführt wurden. Die entstehenden Netze waren zu spärlich und wenig exploriert, als dass eine erfolgreiche Planung möglich gewesen wäre.

Neben diesen beiden Punkten kann jedoch festgehalten werden, dass ein Explorationsalgorithmus mit der Möglichkeit einer Planung nach den Zielvorstellungen dieser Arbeit entwickelt worden ist. Weiterhin wurde eine objektorientierte Implementierung angefertigt, die in Kombination mit den M-Serie Robotern funktionsfähig ist.

Die Simulation hat gezeigt, dass regelmäßige Netze erstellt werden können. Ein Beispiel ist in Abbildung 5.6 auf Seite 77 dargestellt. Weiterhin konnte der Basisalgorithmus zur Erstellung dieser Netze erfolgreich erweitert werden, so dass an jeder Kante mehrere Aktionen zur Auswahl stehen. Diese Auswahl wurde rekursiv durch ein weiteres neuronales Netz an der Kante implementiert.

Diese Auswahl hat es schließlich ermöglicht einen Planungsalgorithmus zu entwickeln, der das neuronale Netz ausnutzt und nach bestimmten Kriterien planen kann. Das letzte Experiment hat gezeigt, dass dadurch der Energieverbrauch im Gegensatz zum Basisalgorithmus gesenkt werden konnte.

Abschließend sei hier der Selbstschutz erwähnt, der sowohl in der Simulation als auch auf aufgenommenen Daten eines Roboters der M-Serie implementiert werden konnte. Weiterhin zeigte sich, dass die Möglichkeit, den Selbstschutz bei der Aktionsauswahl zu berücksichtigen, mit der Homöokinese und der aktiven Motorsteuerung möglich ist.

Zusammenfassend können demnach die Ziele dieser Arbeit als erreicht angesehen werden. Im nächsten Abschnitt werden einige Erweiterungen des Algorithmus und neue Anwendungen aufgeführt.

6.2 Ausblick

Eine wichtige Komponente des Explorationsalgorithmus' ist die Homöokinese. Die hier vorgestellte Regelung verwendet einschichtige neuronale Netze ohne Rückkopplungen und ohne zeitverzögerte Eingänge. Das bedeutet, dass die Regelung keine Möglichkeit hat, die Qualität des Eingangssignals zu verändern. Für die Regelung wird die relative Winkeländerung der letzten Zeitschritte verwendet. Dementsprechend sind die Ausgänge auch als Winkeländerungen oder Drehmomente zu interpretieren. Durch dieses Verhalten wird nie eine instabile Ruhemenge stabilisiert. Die reine Homöokinese wird diese Zustände direkt wieder verlassen. Um einen solchen Zustand zu stabilisieren müsste aus der homöokinetischen eine homöostatische Regelung werden. Dies könnte beispielsweise durch die Einkopplung der absoluten Winkelpositionen des Roboters geschehen. Mit diesen Signalen sollte auch eine Stabilisierung dieser Zustände möglich sein.

Dieses Erweitern der Dimensionen könnte noch erweitert werden, in dem lineare Kombinationen der Sensorwerte oder höhere Ableitungen und Integrale in den Eingangsvektor eingehen. Auch könnten die Struktur des Reglers verändert und beispielsweise Rückkopplungen auf die Eingangsneuronen zugelassen werden.

Basierend auf dieser Exploration könnte dann auch der Parameterraum kategorisiert werden. Neben Oszillationen und Bewegungssteuerungen sollten dann auch Stabilisierungen bestimmter Zustände vorhanden sein. Zu beachten ist jedoch die Dimension des Parametervektors, da jede zusätzliche Dimension exponentiell in die Anzahl an Parametervektoren eingeht.

Als letzte Erweiterung wird hier der Planungsalgorithmus betrachtet. Die strikte Trennung in Planung und Exploration sollte bei der hier beschriebenen Planung aufgelöst werden. Durch die Aktualisierung des Nützlichkeitswertes, kann die Planung auch als Nachoptimierung der neuronalen Netze angesehen werden. Somit ist abzuwägen, ob nicht auch planerische Aspekte in die Exploration mit eingehen sollten. So könnte ein ständiges Erweitern und Konsolidieren des Netzes stattfinden. Die Homöokinese baut das Netz an bestimmten Stellen auf, die anschließend planerisch gefestigt oder revidiert werden. So würde eine anschließende reine Planung bereits auf einem validen und robusten Netz arbeiten. Dadurch würde sich die Zeitspanne des Einpendelns einer gute Route verkürzen. Das Einpendeln kann in Abbildung 5.10 auf Seite 86 nochmals vor Augen geführt werden.

Abbildung 6.1: Zweidimensionaler Roboter: Der Roboter verfügt über zwei Servomotoren an einem „Bein“. Im Kopf ist ein Abstandssensor und ein AccelBoard3D verbaut.

Abschließend wird hier ein kleiner Roboter skizziert, der in der letzten Woche dieser Arbeit im Labor für Neurorobotik fertig gestellt worden ist. Der Roboter sit in Abbildung 6.1 dargestellt. Es handelt sich um einen Roboter mit zwei Gelenken, die eine Art Bein darstellen. Dieses Bein ist an einer tropfenförmigen Konstruktion befestigt, so dass es komplett dahinter verschwinden kann. Durch gezielte Bewegungen soll der Roboter aufstehen können und sich in einer bestimmten Pose hinstellen ohne zusätzlich Energie zu verbrauchen. Weiterhin sollte durch die Konstruktion eine Art hinkendes Gehen möglich sein, mit einem festen und einem motorisierten Bein.

Mit diesem Roboter könnte der aktuelle Explorationsalgorithmus getestet werden. Hier sollten sich ausschließlich kreisende Bewegungen ergeben und der stehende Zustand sollte nicht erreicht werden. Auch eine Belohnung dieses Zustandes sollte zu keiner Stabilisierung führen. Erst durch die Erweiterung des Eingangsvektors der Homöokinese könnte diese Stabilisierung erfolgen. Außerdem sind weitere Belohnungsstufen möglich, so dass beispielsweise das „Gehen“ oder ein bestimmter Abstand zu einem Objekt nochmals belohnt wird. Dieser Roboter stellt einen einfachen Aufbau bereit, der viele verschiedene Zustände annehmen kann. Zudem ist es eine spannende Aufgabe den hier entwickelten Algorithmus an diesem Roboter weiter zu erforschen.

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Literaturverzeichnis

[Ama77] Amari, Sun-ichi: *Dynamics of pattern formation in lateral-inhibition type neural fields.* In: *Biological Cybernetics*, Band 27, Seiten 77–87, 1977.

[CRGU02] Carreras, Marc, Pere Ridao, Rafael Garcia und Zoran Ursulovici: *Learning Reactive Robot Behaviors with a Neural-Q Learning Approach*, 2002.

[DBJ98] Dahm, Percy, Carsten Bruckhoff und Frank Joublin: *A Neural Field Approach to Robot Motion Control*, 1998.

[Der99] Der, Ralf: *Self-Organized Robot Behavior from the Principle of Homeokinesis.*, 1999.

[DHL04] Der, Ralf, Frank Hesse und Rene Liebscher: *Self-Organized Exploration and Automatic Sensor Integration from the Homeokinetic Principle.* In: *Proc. 3rd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE'04)*, Fortschritt-Berichte VDI, Reihe 10, Nr. 743, Seiten 220–230. VDI-Verlag, 2004.

[DHM05] Der, Ralf, Frank Hesse und Georg Martius: *Learning to Feel the Physics of a Body.* In: *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, Seiten 252–257, Washington, DC, USA, 2005. IEEE Computer Society.

[DHM06] Der, Ralf, Frank Hesse und Georg Martius: *Rocking Stamper and Jumping Snake from a Dynamical System Approach to Artificial Life.* Adaptive Behavior, 14(2):105–115, 2006.

[DL02] Der, Ralf und Rene Liebscher: *True autonomy from self-organized adaptivity.* In: *Proc. Workshop Biologically Inspired Robotics*, Bristol, 2002.

[DM06] Der, Ralf und Georg Martius: *From Motor Babbling to Purposive Actions: Emerging Self-exploration in a Dynamical Systems Approach to Early Robot Development.* In: Nolfi, Stefano, Gianluca Baldassarre, Raffaele Calabretta, John C. T. Hallam, Davide Marocco, Jean-Arcady Meyer, Orazio Miglino und Domenico Parisi (Herausgeber): *From Animals to Animats 9, 9th International Conference on*

Simulation of Adaptive Behavior, SAB 2006, Rome, Italy, September 25-29, 2006, Proceedings, Band 4095 der Reihe *Lecture Notes in Computer Science*, Seiten 406–421. Springer, 2006.

[DMH06] Der, Ralf, Georg Martius und Frank Hesse: *Let It Roll – Emerging Sensorimotor Coordination in a Spherical Robot.* In: Rocha, L. M., L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone und A. Vespignani (Herausgeber): *Artificial Life X : Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, Seiten 192–198. International Society for Artificial Life, MIT Press, August 2006.

[DSP99] Der, Ralf, Ulrich Steinmetz und Frank Pasemann: *Homeokinesis - A new principle to back up evolution with learning.* In: *Computational Intelligence for Modelling, Control, and Automation*, Band 55 der Reihe *Concurrent Systems Engineering Series*, Seiten 43–47, Amsterdam, 1999. IOS Press.

[DW01] Dayan, Peter und Christopher JCH Watkins: *Reinforcement Learning.* In: *Encyclopedia of Cognitive Science*. MacMillan Press, 2001.

[ES02] Erlhagen, Wolfram und Gregor Schöner: *Dynamic field theory of movement preparation.* In: *Psychological Review*, Band 109, Seiten 545–572. American Psychological Association, 2002.

[Fri93] Fritzke, Bernd: *Growing Cell Structures - A Self-organizing Network for Unsupervised and Supervised Learning.* Neural Networks, 7:1441–1460, 1993.

[Fri95] Fritzke, Bernd: *A Growing Neural Gas Network Learns Topologies.* In: *Advances in Neural Information Processing Systems 7*, Seiten 625–632. MIT Press, 1995.

[Fri97] Fritzke, Bernd: *A Self-Organizing Network That Can Follow Non-Stationary Distributions*, 1997.

[Gal07] Galor, Oded: *Discrete Dynamical Systems.* Springer Verlag, 2007.

[GSK98] Gross, H.-M., V. Stephan und M. Krabbes: *A Neural Field Approach to Topological Reinforcement Learning in Continuous Action Spaces.* In: *In Proc. of WCCI-IJCNN'98, Anchorage*, Seiten 1992–1997. IEEE Press, 1998.

[GWZZ99] Gaskett, Chris, David Wettergreen, Alexander Zelinsky und Er Zelinsky: *Q-Learning in Continuous State and Action Spaces.* In: *In Australian Joint Conference on Artificial Intelligence*, Seiten 417–428. Springer-Verlag, 1999.

[Haf05] Hafner, Verena V.: *Cognitive Maps in Rats and Robots.* Adaptive Behavior, 13:87–96, 2005.

[Hay08] Haykin, Simon: *Neural Networks and Learning Machines*. Prentice Hall, 3. Auflage, 2008.

[HDH07] Hesse, Frank, Ralf Der und J. Michael Herrmann: *Reflexes from Self-Organizing Control in Autonomous Robots*. In: Berthouze, Luc, Christopher G. Prince, Michael Littman, Hideki Kozima und Christian Balkenius (Herausgeber): *7th International Conference on Epigenetic Robotics: Modelling Cognitive Development in Robotic Systems, Rutgers University, Piscataway, NJ, USA*, Band 134 der Reihe *Cognitive Studies*, Seiten 37–44. Lund University, 2007.

[HDH09] Hesse, Frank, Ralf Der und Michael Herrmann: *Modulated Exploratory Dynamics can shape self-organized Behaviour*. Advances in Complex Systems, 12(3):273–291, 2009.

[Hea08] Hild, Manfred und et al.: *Workshop Sensomotorik*. Humboldt-Universität zu Berlin, 2008.

[Heu03] Heun, Voler: *Grundlegende Algorithmen - Einführung in den Entwurf und die Analyse effizienter Algorithmen*. Friedr. Vieweg & Sohn Verlag, 2 Auflage, 2003.

[Hil07] Hild, Manfred: *Neurodynamische Module zur Bewegungssteuerung autonomer mobiler Roboter*. Doktorarbeit, Humboldt-Universität zu Berlin, April 2007.

[HK03] Hagen, Stephan Ten und Ben Kröse: *Neural Q-Learning*. In: *In Neural Computing & Applications*, Seiten 81–88, 2003.

[HMDH09] Hesse, Frank, Georg Martius, Ralf Der und J. Michael Herrmann: *A Sensor-Based Learning Algorithm for the Self-Organization of Robot Behavior*. Algorithms, 2(1):398–409, 2009.

[Hol08] Holoch, Wolfgang: *Eine biologisch inspirierte Navigation für mobile Roboter*. Diplomarbeit, Universität Ulm, August 2008.

[KM97] Karayiannis, Nicolaos B. und Glenn Weiqun Mi: *Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques*. IEEE Transactions on Neural Networks, 8:1492–1506, 1997.

[Koh90] Kohonen, Teuvo: *The self-organizing map*. In: *Proceedings on the IEEE*, Band 78, Seiten 1464–1480, September 1990.

[MDH+09] Martius, Georg, Ralf Der, Frank Hesse, Frank Güttler, Jörn Hoffmann, Marcel Kretschmann, Dominic Schneider und Claus Stadler: *Software „lpzrobots-0.5“*. `http://robot.informatik.uni-leipzig.de/software/`, Oktober 2009.

[MFH08] Martius, Georg, Katja Fiedler und J. Michael Herrmann: *Structure from Behavior in Autonomous Agents*. In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[MHD07] Martius, Georg, J. Michael Herrmann und Ralf Der: *Guided Self-organisation for Autonomous Robot Development*. In: Costa, Almeida e und Francesco (Herausgeber): *Advances in Artificial Life 9th European Conference, ECAL 2007, Lisbon, Portugal*, Band 4648 der Reihe *Lecture Notes in Computer Science*, Seiten 766–775. Springer, 2007.

[OK07] Oudeyer, Pierre-Yves und Frédéric Kaplan: *What is intrinsic motivation? A typology of computational approaches*. Frontiers in Neurorobotics, 1, November 2007.

[OKH07] Oudeyer, Pierre-Yves, Frédéric Kaplan und Verena V. Hafner: *Intrinsic motivation systems for autonomous mental development*. IEEE Transactions on Evolutionary Computation, 11:265–286, 2007.

[OKHW05] Oudeyer, Pierre-Yves, Frédéric Kaplan, Verena V. Hafner und Andrew Whyte: *The playground experiment: Task-independent development of a curious robot*. In: *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, Seiten 42–47, 2005.

[SB98] Sutton, Richard S. und Andrew G. Barto: *Reinforcement Learning*. MIT Press, März 1998.

[SDL] *Simple DirectMedia Layer*. `http://www.libsdl.org/`.

[Ste10] Stephan, André: *Visualisierung und Neuronales-Gas-Lernen von Sensordaten des humanoiden A-Serie Roboters*. Diplomarbeit, Humboldt-Universität zu Berlin, 2010.

[Tou04] Toussaint, Marc: *Learning a world model and planning with a self-organizing dynamic neural system*. In: *Advances in Neural Information Processing Systems 16*, Seiten 929–936. MIT Press, 2004.

[Tou06] Toussaint, Marc: *A sensorimotor map: Modulating lateral interactions for anticipation and planning*. In: *Neural Computation 18*, Seiten 1132–1155, 2006.

[Tou07] Toussaint, Marc: *Sensorimotor Map Videos*. `http://user.cs.tu-berlin.de/~mtoussai/04-smm/index.html`, November 2007.

[Wat89] Watkins, Christopher JCH: *Learning from Delayed Rewards*. Doktorarbeit, University of Cambridge, England, 1989.

[Zel97] Zell, Andreas: *Simulation neuronaler Netze*. R. Oldenburg Verlag, 2 Auflage, 1997.