Diplomarbeit

Integration lokaler Navigationsstrategien mittels Reinforcement Learning

Denise Peters

03. August 2006

Betreut von

Prof. Dr. Hanspeter A. Mallot
Lehrstuhl Kognitive Neurowissenschaft
Fakultät für Biologie

Prof. Dr. Wolfgang Rosenstiel
Lehrstuhl für Technische Informatik
Fakultät für Kognitions- und Informationswissenschaften

Zusammenfassung

Navigation ist eine alltägliche Aufgabe für Menschen und Tiere, dennoch sind viele Fragen bezüglich der grundlegenden Mechanismen der Navigation noch nicht beantwortet. Eine essentielle Navigationsleistung ist das Zurückfinden zu bekannten Orten („Heimfindung", „Homing“'). Für eine robuste Heimfindung werden im Tierreich oft unterschiedliche Navigationsstrategien kombiniert, wodurch mögliche Fehler minimiert werden. In dieser Diplomarbeit erhält ein virtueller Roboter die Navigationsaufgabe zu einem bekannten Ort zurück zu kehren. Hierzu stehen ihm drei lokale Navigationsstrategien zur Verfügung: visuelles Homing, Wegintegration und Hindernisvermeidung. Diese Strategien liefern in der Regel inkonsistente Bewegungsentscheidungen, weshalb eine Integration der Strategien erfolgen muß. Der in dieser Diplomarbeit beschriebene Ansatz ermöglicht es dem Roboter die Gewichtung zwischen den einzelnen Navigationsstrategien kontinuierlich zu bestimmen. Der Roboter soll dabei eigenständig die Gewichtungen der einzelnen Navigationsstrategien situations- und zeitabhängig setzen. Das Ziel der dieser Arbeit ist es, eine Entscheidungsstrategie zu ermitteln, welche es ermöglicht die Gewichtung der Navigationsstrategien zeit- und situationsabhängig zu wählen. Hierzu wird ein Reinforcement Learning Ansatz verwendet, der den kontinuierlichen Zustands-Aktionsraum mittels eines Radialen-Basisfunktionen Netzwerks approximiert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Navigation ist eine Leistung, die Menschen und Tiere täglich erbringen. Viele Tiere - sogar Ameisen mit einem Gehirn nicht größer als ein Stecknadelkopf - zeigen ein erstaunliches Navigationsverhalten. Da Tiere sehr auf zuverlässige Navigation angewiesen sind, haben sich mit der Zeit robuste Navigationsstrategien herausgebildet. Diese ermöglichen es, die zur Navigation nötige Information aus der Umgebung zu extrahieren und zu verarbeiten, so dass das Tier optimal an seine Umgebung angepasst ist. Trotz erheblicher Forschung in diesem Bereich sind noch viele Fragen bezüglich der zugrundeliegenden Mechanismen unbeantwortet.
Zur Beantwortung dieser Fragestellung werden neben Methoden der Verhaltensbiologie auch autonome Roboter als Modell zur Überprüfung biologischer Hypothesen verwendet. Die Robotik stellt dabei eine gute Möglichkeit dar, die kognitiven Mechanismen der Navigation zu untersuchen. Ein möglicher Ansatz besteht darin komplexes Verhalten (komplexe kognitive Leistungen) als Kombination von einfachen „elementaren“ Verhaltensweisen (Modul) zu beschreiben. Als Modul bezeichnet man einen Teil des Systems, der auch isoliert betrachtet eine eigenständige Funktion erfüllt. Ausgangspunkt für diese inverse Herangehensweise - Aufbau eines kognitiven Systems zur Untersuchung von Hypothesen - sind die Gedankenexperimente von V. Braitenberg [3]. In diesem Experiment stellt er einfache „Vehikel“ vor und diskutiert ihr Verhalten und die Schlüsse, die ein Beobachter aus ihrem Verhalten ziehen könnte. Aus dieser Diskussion geht hervor, dass sich auf Grund von Beobachtung des Verhaltens kein Bauplan ableiten lässt. Daher sollte man umgekehrt (invers) beginnen, indem man einen Bauplan auf einem Roboter realisiert und überprüft, ob der Roboter das gewünschte Verhalten zeigt. Hierbei geht man davon aus, dass die einfachste Lösung die wahrscheinlichste Lösung darstellt. Dieses Prinzip wird auch Prinzip von Ockham genannt.
Die einzelnen Verhaltensmodule werden in der Robotik oft über eine Subsumptionsarchitektur integriert [4]. Hierbei erfolgt die Verhaltensauswahl durch eine diskrete Fallabfrage. Allerdings besteht das Problem, dass die Anpassung des Verhaltens an die jeweilige Situation über einen Außenstehenden (den Programmierer) erfolgt. Dies entspricht nur bedingt dem biologischen Vorbild. Es wäre näher am biologischen Vorbild, wenn der Roboter lernen würde, wann er welcher Navigationsstrategie folgt. Dieser Ansatz wurde in den letzten Jahren immer stärker berücksichtigt. Hierbei stand nicht zwangsläufig die bio-

logische Motivation im Vordergrund, sondern oftmals das Problem, dass die Feineinstellung/Feinjustierung des Kontrollsystems eines autonomen Roboters eine sehr aufwendige Aufgabe darstellt [18]. Ein Lernansatz, der für Navigationsaufgaben oftmals eingesetzt wird, ist das Reinforcement Learning [18], [14], [12], [13]. Der Vorteil dieses Lernansatzes ist, dass der Roboter über Erfahrungen selbstständig lernt. Allerdings bezogen sich die bisherigen Ansätze mehr auf die Verwendung einzelner Navigationsstrategien als auf die Integration der Strategien, beispielsweise die Hindernisvermeidung [18], [14]. In dieser Diplomarbeit soll nun ein Ansatz vorgestellt werden, in dem ein autonom navigierender Roboter die Gewichtung seiner vorhandenen Navigationsstrategien situations- und zeitabhängig lernen soll. Dadurch soll die Subsumptionsarchitektur ersetzt werden und somit auch potentielle Problemquellen dieses Verfahrens verhindert werden.

Kapitel 2 - Grundlagen In diesem Kapitel wird zunächst die Navigation, insbesondere die hierarchische Einteilung der Navigationsstrategien, vorgestellt. Der zweite Abschnitt beschäftigt sich sowohl mit der biologisch/psychologischen Seite als auch mit der informatischen Seite des Lernens. Der Schwerpunkt liegt hierbei auf dem Lernverfahren Reinforcement Learning. Die Ausführung der psychologischen Sichtweise soll dabei helfen die Ideen und Herangehensweise des informatischen Reinforcement Learning besser zu verstehen. Abschließend wird die Funktionsapproximation mit Hilfe von einem Radiale-Basisfunktionen-Netz (RBF-Netz) vorgestellt.

Kapitel 3 - Problemstellung Im dritten Kapitel wird die eigentliche Problemstellung der Diplomarbeit vorgestellt. Als Navigationsaufgabe wurde in dieser Arbeit das Heimfindungsverhalten ausgewählt. Nachdem die lokalen Navigationsstrategien - das visuelle Homing, die Wegintegration und die Hindernisvermeidung - kurz vorgestellt wurden, wird die Integration dieser Strategien mittels der Subsumptionsarchitektur erläutert und deren Problemquellen ausgeführt. Am Ende des Kapitels wird der in dieser Diplomarbeit neu entwickelte Ansatz zur situations- und zeitabhängigen Gewichtung der Navigationsstrategien dargestellt und die Gründe der Verwendung des Reinforcement Learnings aufgezeigt.

Kapitel 4 - Reinforcement Learning Problem Die situations- und zeitabhängige Gewichtung soll nun durch den Roboter selbst erfolgen. Es soll eine kontinuierliche Entscheidungsstrategie gelernt werden. Zur Verwendung von Reinforcement Learning muß die Lernaufgabe dazu in ein Reinforcement Learning Problem umformuliert werden. Hierzu werden die Grundelemente des Reinforcement Learning für die vorliegende Aufgabenstellung definiert und die verwendete Methodik besprochen.

Kapitel 5 - Radiale-Basisfunktionen Netz (RBF-Netz) Die Verwendung eines Funktionsapproximators ist aufgrund des kontinuierlichen Zustands-Aktionsraumes unabdingbar. Zur Funktionsapproximation wird ein RBF Netzwerk verwendet. Die genaue Implementierung und die entwickelten Algorithmen zur Bestimmung der Koeffizienten und sowie der Suche der Maxima wird in diesem Kapitel vorgestellt und anhand von ein- und zweidimensionalen Fallbeispielen ausgeführt. Am Ende des Kapitels wird auf die

Problematik der Messpunktauswahl eingegangen.

Kapitel 6 - Lernexperiment und Ergebnisse In Kapitel 6 wird das Lernexperiment vorgestellt. Hierzu wird zunächst die Simulationsumgebung beschrieben. Im Anschluss daran wird die Parametrisierung des Reinforcement Learning Problems erläutert. Auf die Einstellung und Anpassung des RBF Netzwerkes wird danach ausführlicher eingegangen, da die Approximation in einem hochdimensionalen Raum eine effiziente Implementierung erfordert. Nachdem der Ablauf des Lernexperimentes erläutert ist, werden die Ergebnisse vorgestellt.

Kapitel 7 - Diskussion und Ausblick Im letzten Kapitel erfolgt die abschließende Diskussion der Ergebnisse. Zudem wird erläutert, weshalb die in diesem Ansatz realisierte Approximation der Q-Funktion gegenüber bisherigen Approximationsversuchen viele Vorteile aufweist. Im Ausblick werden Vorschläge für mögliche Verbesserungen und weiterführende Arbeiten vorgestellt.

Kapitel 2

Grundlagen

2.1 Navigation

Die Navigationsleistungen, die Menschen und Tiere vollbringen, sind erstaunlich. Jedoch sind bis jetzt viele Fragen zu den grundlegenden Mechanismen noch unbeantwortet. Eine Möglichkeit biologische Hypothesen über Navigation zu testen, ist, sie mit Hilfe von biologisch-motivierten mobilen Robotern zu überprüfen. Zugleich helfen die biologisch begründeten Mechanismen bei der Konstruktion neuer Methoden in der Robotik autonom navigierende Agenten zu entwickeln oder zu verbessern.
Navigation ist definiert als *"ein Prozess der Bestimmung und Beibehaltung eines Kurses oder einer Trajektorie zu einem Zielpunkt"* [9][1]. Notwendige Fähigkeiten für die Navigation sind die Fortbewegung und die Fähigkeit bestimmen zu können, ob der Zielpunkt schon erreicht wurde. Für die Wiedererkennung des Zielpunktes muss man Merkmale zur Erkennung dieses Ortes abspeichern können, d.h. das Tier oder der Roboter muss eine Ortsrepräsentation besitzen. Wenn das Tier oder der Agent zudem fähig ist sich mehrere Orte merken zu können und diese Orte durch - im einfachsten Fall - Aktionen miteinander zu verbinden, spricht man von einer kognitiven Karte. Bestimmte Grundfähigkeiten der Navigation sind beispielsweise die Hindernisvermeidung, die Exploration oder die Kursstabilisierung.

2.1.1 Navigationsstrategien

Die Einteilung der Navigationsstrategien richtet sich nach der Klassifizierung von M.O. Franz & H.A. Mallot [7]. Im Folgenden wird der Navigierende, ob Mensch, Tier oder Roboter, als Agent bezeichnet. Grundsätzlich kann man zwischen zwei Gruppen von Navigationsstrategien unterscheiden: lokale Navigationsstrategien und globale Navigationsstrategien oder auch Wegfindungsstrategien genannt. Der Unterschied zwischen diesen beiden Gruppen ist, dass bei den lokalen Strategien der Agent in der Lage sein muss sich nur an einen Ort zu erinnern, um diesen wieder zu erkennen. Bei Wegfindungsstrategien

[1] Aus dem Englischen: *„Navigation is the process of determining and maintaining a course or trajectory to a goal location."*

hingegen muss der Agent mehrere Orte speichern können und zudem eine Verbindung zwischen den unterschiedlichen Orten aufbauen. Die Verbindung könnte beispielsweise aus Regeln der Form: „Am roten Haus rechts“ bestehen. Die Navigation von einem Ort zu einem anderen wird durch lokale Strategien bewältigt, und durch die Erweiterung mit globalen Strategien ist es möglich, dass mehrere Zielpunkte (Teilziele) dargestellt und auch die Zusammenhänge zwischen den Orten gespeichert werden können.
Sowohl die lokalen als auch die globalen Navigationsstrategien lassen sich nach der Komplexität der Aufgabenstellung einteilen. Die lokalen Navigationsstrategien lassen sich vier Komplexitätsstufen zuordnen: Suche („Search“), Zielanfahrt („Direction-following“), Zielführung („Aiming“) und Wegweisung („Guidance“). Die globalen Navigationsstrategien gliedern sich in drei Ebenen: „Recognition-triggered response“, „Topological navigation“ und „Survey navigation“. Hierbei stellt die unterste Ebene die Strategie mit der niedrigsten Komplexität dar. Durch diese Einteilung in die unterschiedlichen Ebenen folgt, dass ein Agent, der die Fähigkeiten einer Strategie besitzt, auch in der Lage ist, die Strategien auszuüben, die unter dieser Ebene liegen. Dennoch benötigt man bei den globalen Strategien nicht notwendigerweise alle lokalen Navigationsstrategien.

2.1.1.1 Lokale Navigationsstrategien

Suche Diese Strategie stellt die einfachste Form der Navigation dar. Der Agent benötigt hierfür nur die Fähigkeit der Fortbewegung und Zielerkennung, aber keine Orientierung oder räumliche Repräsentation des Raumes. Die Suche kann nun nach einer systematischen Strategie erfolgen oder auch zufällig sein.

Zielanfahrt Diese Strategie basiert entweder auf externen Informationen, wie beispielsweise Orientierung nach einem Magnetfeld oder an einer Pheromonspur, oder auf internen Informationen, wie zum Beispiel die Stellung des Körpers im Raum durch propriozeptive und vestibuläre Signale. Diese Strategie wird wesentlich effizienter, wenn zudem noch die Distanz zum Zielort bekannt ist. Eine Strategie, für die man sowohl die Information der Richtung als auch der Distanz benötigt, ist die Wegintegration.

Wegintegration Unter Wegintegration versteht man die kontinuierliche Schätzung der Eigenbewegung. Dadurch ist der Agent in der Lage, kontinuierlich die Veränderungen der Distanz und der Richtung bezüglich des Zieles zu integrieren und kann so zu jedem Zeitpunkt seine relative Position zum Startpunkt bestimmen. Die Schätzung der Eigenbewegung erfolgt in der Robotik über die Odometrie. Hierbei werden über die Radstände die Bewegungen des Roboters berechnet.
Allerdings eignet sich die Wegintegration nur für kurze Stecken, da sich fehlerhafte Schätzungen der Eigenbewegung schnell akkumulieren, was den Heimvektor stark verfälscht.

Zielführung Bei der Zielführung muss der Agent in der Lage sein, seine Körperachse nach einem Hinweis auszurichten, so dass das Ziel vor ihm liegt. Dieser Hinweis kann beispielsweise eine Duftspur sein, ein Geräusch oder auch ein visueller Reiz. Nach diesem Hinweis richtet sich das Tier aus und versucht die Quelle des Hinweises zu erreichen. Der Bereich um den Zielpunkt, in dem das Erreichen des Zielpunktes durch diesen Hinweis möglich ist, wird auch Fangbereich oder „*Catchment Area*“ genannt.

Wegweisung Bei dieser Strategie ist das Ziel nicht durch einen Hinweis markiert, sondern der Zielort wird durch die räumliche Anordnung von sich in der Umgebung befindenden Objekten identifiziert. Der Agent muss seine eigene Position nun relativ zu diesen Objekten ausrichten. Um den Zielpunkt wiederfinden zu können, muss er dann die Position einnehmen, die er bei der Speicherung der Objektkonfiguration des Zielpunktes inne hatte. Die Objekte, die zur Wiederfindung des Ortes genutzt werden, werden auch Landmarken genannt. Ein Beispiel für eine solche Navigationsstrategie ist das visuelle Homing.

Visuelles Homing Bei der visuellen Navigation ist der Zielpunkt durch eine Anordnung von lokalen, sichtbaren Landmarken definiert. Man konnte experimentell nachweisen, dass Insekten, wie beispielsweise Bienen und Ameisen die Konfiguration der Landmarken um den Zielpunkt nutzen um diesen wiederzufinden. Wird nun diese Konfiguration von Landmarken umgesetzt, sucht das Tier an dem durch die Landmarken bestimmten Ort nach seinem Nest.
Ein Modell hierzu wurde von Cartwright & Collett [5] formuliert. Ein Ort in diesem Modell wird charakterisiert durch ein 360° Panoramabild (Snapshot). Von einem besuchten Ort wird ein Bild aufgenommen. Das Ziel ist nun, den Zielpunkt zu erreichen, indem man die aktuelle Aufnahme mit dem gespeicherten Bild vergleicht und auf Ähnlichkeit analysiert. Ein Algorithmus, der für das visuelle Homing genutzt werden kann, ist der „*Image Warp*“-Algorithmus von M. Franz [8]. Dieses Modell sagt voraus, wie sich das aktuelle Bild verändern wird, wenn man eine bestimmte Bewegung ausübt. Dies wird für den aktuellen Standpunkt in diskreten Richtungen bestimmt und die Richtung gewählt, die die höchste Ähnlichkeit mit den Zielbild besitzt. Die genaue mathematische Formulierung wird in Abschnitt 3.1.1.1 noch ausführlich vorgestellt.

2.1.1.2 Globale Navigationsstrategien

”Recognition-triggered response” Diese Strategie wird auch Routennavigation genannt. Hierbei werden zwei Orte, die mit Hilfe einer lokalen Navigationsstrategie gefunden wurden, durch eine Aktion miteinander verbunden, beispielsweise: „Bin an Ort 3, muss jetzt rechts gehen“. Dadurch entsteht eine Liste von Orten, die mit solchen Anweisungen eine Route ergeben. Das bedeutet, man erhält an einem bestimmten Ort eine sensorische Eingabe und reagiert darauf mit einer bestimmten Aktion. Dieses Verhalten ist unflexibel und nicht in der Lage, auf neue Hindernisse zu reagieren.

"Topological navigation" Bei der topologischen Navigation benötigt der Agent eine Repräsentation der Orte mit ihren Verbindungen untereinander. Durch diese Repräsentation ist der Agent in der Lage, eine geplante Route abzuändern, falls diese blockiert ist und einen alternativen Weg zu nehmen oder die gespeicherte Repräsentation auch für andere Routen zu nutzen.

"Survey navigation" Die letzte vorgestellte Navigationsstrategie erfordert nicht nur wie bei der topologischen Navigation eine Repräsentation der Orte und ihrer Verbindungen, sondern zudem wird bei dieser Strategie ein Überblickswissen benötigt. Überblickswissen bedeutet hierbei, dass der Agent in seiner Raumrepräsentation alle bekannten Orte in einen räumlichen Bezug setzt (in das gleiche Koordinatensystem). Diese Repräsentation ermöglicht es, eine Abkürzung über einen noch nicht erkundeten Bereich zu nehmen.

2.1.1.3 Subsumptionsarchitektur vs. flexible Verhaltensauswahl

Zur Lösung einer Navigationsaufgabe können mehrere lokale Navigationsstrategien genutzt werden. Durch die Kombination von Navigationsstrategien können so Fehlerquellen der einzelnen Strategien durch die anderen ausgeglichen werden. Ein Beispiel aus dem Tierreich stellt die Wüstenameise *Cataglyphis fortis* dar, die durch die Kombination von Wegintegration und visuellem Homing ihren Fangbereich vergrößert. Als erstes folgt diese Ameise ihrer Wegintegration. Wenn der Wegintegrationsvektor abgelaufen ist und der Zielpunkt noch nicht erreicht wurde, wird versucht das Ziel über ein visuelles Homing zu erreichen. Misslingt auch dies, benutzt die Ameise eine spezielle Suchstrategie um ihren Zielpunkt wieder zu finden [23].
Die Integration der Navigationsstrategien erfolgt bei autonomen mobilen Robotern über ein Kontrollsystem. Das Kontrollsystem muss dabei in der Lage sein unterschiedliche sensorische Informationen auszuwerten und die entsprechende Verhaltensweise auszuwählen. Allgemein werden diese komplexen Systeme, die auch ein mobiler Roboter darstellt, modular aufgebaut. Als ein Modul fasst man dabei eine Teilverarbeitung auf, die auch isoliert betrachtet und verstanden werden kann. Bei der Integration dieser Module kann man zwischen zwei Ansätzen unterscheiden: der horizontalen Modularisierung a.) und der vertikalen Modularisierung b.), die in Abbildung 2.1 dargestellt ist [4].

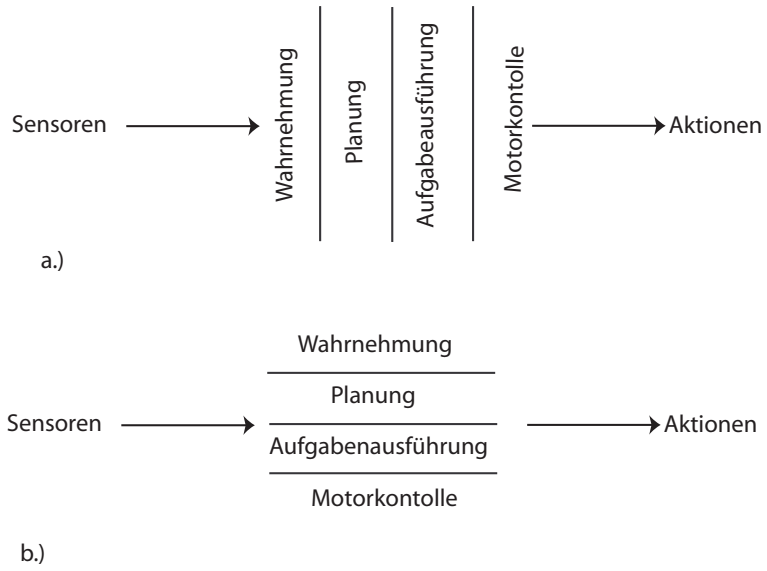

Abbildung 2.1: **Modularisierung [4]** a.) zeigt die schematische Darstellung der horizontalen Modularisierung. In b.) wird eine vertikale Modularisierung dargestellt (Bsp.: Subsumptionsarchitektur)

Die Subsumptionsarchitektur stellt eine vertikale Modularisierung dar [4]. Bei dieser Architektur werden die einzelnen Module in Form einer Fallunterscheidung miteinander verknüpft, beispielsweise: „Wenn ein Hindernis detektiert wird, dann wird die Hindernisvermeidung aktiv, wenn nicht, aktiviere die Wegintegration“. Die Fallabfrage ist dabei hierarchisch aufgebaut, so dass „wichtige“ Aufgaben, zum Beispiel die Hindernisvermeidung eine höhere Priorität habe. Durch diese Fallabfrage kann man ein sehr robustes Kontrollsystem aufbauen, das auch mit den unterschiedlichsten Situationen zurecht kommt.
In dieser Diplomarbeit wird ein Ansatz vorgestellt, der die diskrete Fallunterscheidung der Subsumptionsarchitektur durch eine kontinuierliche Integration ersetzt. Die kontinuierliche Integration der Navigationsstrategien soll dabei vom Agenten selbständig gelernt werden. Hierzu wird ein Reinforcement Learning Ansatz verwendet. Der Agent soll nun lernen, die gegebenen Navigationsstrategien, das visuelle Homing, die Wegintegration und die Hindernisvermeidung, bezogen auf die jeweilige Situation unterschiedlich zu gewichten und so die Strategien zu integrieren. Hierzu sollen im folgenden Kapitel zunächst die Grundlagen des Lernens und insbesondere des Reinforcement Learning erläutert werden, sowohl aus biologisch/psychologischer Sichtweise, als auch vom informatischen Ansatz her.

2.2 Lernen

Das Forschungsgebiet Lernen als Gegenstand der Psychologie entwickelte sich vor ungefähr 120 Jahren. Sowohl aus dem Bereich der Philosophie als auch aus der Biologie kamen hierbei Anstöße. Gerade die Bewegungen der Empiriker und der Rationalisten aus dem siebzehnten und achtzehnten Jahrhundert und die Entwicklung der Evolutionstheorie im neunzehnten Jahrhundert begründeten das Forschungsinteresse am Thema Lernen. Einer der zentralen Punkte war dabei die Frage, ob ein Verhalten „angeboren oder gelernt“ („nature-nurture“) ist, das heißt: Wieviel wird durch unsere Natur festgelegt und wieviel ist anerzogen durch unsere Gesellschaft? Viele Philosophen wie zum Beispiel Rousseau oder Candland beschäftigten sich mit dieser Fragestellung. Aus dieser Gedankenwelt kristallisierte sich die Epistemologie heraus, die Frage nach dem Wissenserwerb. Dies stellt noch heute in einem weitergefassten Rahmen den zentralen Punkt des Thema Lernens dar [21]. Erlangen wir unser Wissen nun durch logische Schlussfolgerungen, wie es schon Descartes vorgeschlagen hat (Rationalismus) oder erlangen wir es über Erfahrungen, so dass wir eine Kausalität zwischen Ereignissen erlernen (Empiriker), wie es beispielsweise Locke postuliert hat? Diese philosophische Fragestellung beeinflusste die frühe psychologische Forschung stark und spiegelt sich beispielsweise deutlich bei den Experimenten von Iwan R. Pavlov oder auch Edward L. Thorndike wieder [21].
Biologisch betrachtet, brachte Darwin eine entscheidende Wende. In seinem Werk „On the Origin of Species“ aus dem Jahre 1859 präsentierte er eine Theorie über die Evolution, die beschrieb, dass Organismen über Generationen hinweg besser an ihre Umgebung adaptieren. Er war einer der Ersten, der betonte, dass es Unterschiede zwischen Individuen einer Generation gab. Manche dieser Unterschiede führten zu besseren Überlebens- und Fortpflanzungschancen, was über Generationen hinweg zu einer Spezialisierung führte. Die Fähigkeit zu Lernen wird zu einer solchen Spezialisierung gezählt, und viele Psychologen vertreten die Meinung, dass Lernen eine Adaption an die Umgebung bedeutet [21].
In der heutigen Zeit beschäftigen sich die unterschiedlichsten Disziplinen mit dem Thema Lernen. Hierunter fallen nicht nur die Psychologie und die Biologie, sondern auch andere Bereiche wie die Informatik. In der Informatik beschäftigen sich Bereiche wie zum Beispiel künstliche Intelligenz und maschinelles Lernen stark mit abstrakteren Formen des Lernens. Über Bereiche wie Neuroinformatik wird die Zusammenarbeit mit den Kognitionswissenschaften auf der biologisch-psychologischen Seite immer stärker.
So breit wie die Interessensfelder beim Thema Lernen gestreut sind, so unterschiedlich sind auch die möglichen Definitionen von Lernen. So spricht ein Kognitionswissenschaftler von Lernen als dem Erwerb von Wissen. Psychologisch betrachtet spricht man von Lernen als einer relativen Verhaltensänderung oder Veränderung des Verhaltensrepertoirs als basierend auf Erfahrung. In der Biologie ist Lernen allgemein gesehen als Adaptation an die Umgebung/Reiz definiert [21]. Die Diskussion über die Definition von Lernen kann man je nach Bereich des Forschungsinteresses noch ausführlicher behandeln. Aus diesem Grund möchte ich im Folgenden einen kurzen Abriss über die Teilbereiche wie Konditionierung geben und im Besonderen auf das Reinforcement Learning eingehen.

2.2.1 Biologische/psychologische Sichtweise

Die Forschung im Bereich Lernen versucht die fundamentalen Prozesse des Lernens zu ergründen. Hierbei steht im Vordergrund aufzuklären welche Kausalität zwischen den Schlüsselvariablen besteht. Bei den klassischen Versuchen werden aus diesem Grunde alle Variablen bis auf eine kontrolliert, um so die Einflussnahme dieser einen zu untersuchenden Variable zu erforschen.
Die Forschungsbereiche spalten sich dabei auf in Verhaltensforschung, Kognitionswissenschaft und neuropsychologische Forschung, die aus Teilbereichen von Disziplinen wie Neurologie und Neurophysiologie zusammengesetzt sind.
Lernen lässt sich in zwei Hauptgruppen unterteilen, nicht-assoziatives Lernen und assoziatives Lernen [6]. Unter einfachem, nicht-assoziativem Lernen versteht man eine Gewöhnung an einen wiederholten, gleichartigen Reiz. Kommt es zu einer Abnahme der Reaktion auf diesen schwachen Reiz spricht man von einer Habituation. Durch starke und bedeutungsvolle Reize kann die Habituation rückgängig gemacht werden (Dishabituation). Im Falle einer verstärkten Reaktion und höheren Reizempfindung auf einen starken und bedeutungsvollen Reiz handelt es sich um eine Sensitisierung. Am Beispiel der Meeresschnecke *Aplysia californica* sind die zellulären Grundlagen der Habituation, Dishabituation und Sensitisierung weitgehend aufgeklärt worden. Habituation und Sensitisierung werden als nicht-assoziative Lernformen bezeichnet, weil keine neuen Bedeutungsverknüpfungen zwischen verschiedenen Reizen oder Reizen und Reaktionen entstehen, sondern existierende Verknüpfungen ihre Wirkung vorübergehend verlieren oder verstärken [6].
Wenn man von assoziativem Lernen spricht, versteht man darunter beispielsweise die Konditionierung eines Tieres auf einen Reiz. Man unterscheidet dabei zwischen der klassischen (Pavlov´schen) Konditionierung und der operanten (instrumentellen) Konditionierung [6]. Bei der klassischen Konditionierung wird ein "neutraler" Stimulus (CS = conditioned stimulus) mit einem bedeutungsvollen Stimulus (US = unconditioned stimulus) gepaart. Der US dient dabei als Belohnung oder Bestrafung. Nach der Konditionierung wird zwischen dem CS und dem US eine neue Beziehung assoziiert. Das klassische Beispiel hierfür ist der Pavlov´sche Hund, der auf einen Ton konditioniert wurde. Ein hungriger Hund reagiert mit Speichelfluss (unkonditionierte Reaktion, UR), wenn man ihm Fleischpulver in den Mund bläst (unkonditionierter Stimulus, US). Wird mehrmals der US kurz nach dem Erklingen eines Tones gegeben (zu konditionierender Stimulus, CS), dann löst später der CS alleine den Speichelfluss aus [6].
Die operante oder instrumentelle Konditionierung setzt im Unterschied zur klassischen Konditionierung eine aktive Beteiligung des Tieres voraus. Im Englischen wird diese Form des Lernens auch als Reinforcement Learning bezeichnet. Entscheidend dabei ist, dass die Aktion eines Tieres zu einem Ergebnis führt. Das Ergebnis sollte für das Tier ein angestrebtes Ziel darstellen und erhält damit eine Bewertung. Das Tier lernt dabei die Beziehung zwischen der eigenen Aktion, Bewertung und die darauf hinweisenden Stimuli. Da die grundlegenden Konzepte des informatischen Reinforcement Learnings aus der Psychologie stammen, werden die psychologischen Grundlagen im folgenden Kapitel ausführlicher erläutert [6].

Höhere Formen assoziativen Lernens haben gemeinsam, dass die Antriebe und bewertenden Ergebnisse nicht äußere Stimuli, sondern innere Zustände sind, wie beispielsweise Erwartungen und Neugier. Hierunter fallen Formen des Lernens wie Orientierungslernen, beobachtendes Lernen und spielendes Lernen. In einer neuen Umgebung zeigen Tiere sehr häufig ein Erkundungsverhalten, durch das sie lernen sich besser zu orientieren und zielsicher zu einem Ausgangspunkt zurückzukehren, was auch als Homing bezeichnet wird. Beim beobachtenden Lernen und beim Nachahmungslernen, das vor allem bei Primaten hoch entwickelt ist, wird durch die gerichtete Aufmerksamkeit auf die Tätigkeit eines anderen Tieres ein Verhalten gelernt. Ähnlich verhält es sich auch beim spielerischen Lernen, das bei Primaten von besonderer Bedeutung für die Entwicklung sozialen Verhaltens ist. Häufig manifestiert sich die Sozialkompetenz, gelernt durch das Spiel im Kindesalter, später im adulten Alter [6].
Auf der obersten Ebene - kognitiv gesehen - lassen sich einsichtiges Lernen, bewußtwerdendes und sprachabhängiges Lernen einordnen. Einsichtiges Lernen ist bislang nur bei Primaten nachgewiesen. Darunter versteht man, dass das Tier in der Lage ist eine schwierige Aufgabe, zum Beispiel eine hoch hängende Banane, nicht durch Ausprobieren, sondern durch „Nachdenken“ zu lösen, indem es beispielsweise Stühle aufeinander stellt oder Rohre zusammensteckt [6].
Bewußtwerdendes und sprachabhängiges Lernen sind charakteristisch für den Menschen. Man spricht hierbei von der Bewußtwerdung von Lerninhalten, so dass dieses Wissen in einem anderen Kontext wieder angewendet werden kann. Lerninhalte können auch über die Sprache vermittelt werden [6].
Natürlich stellt diese kurze Einordnung nur einen sehr knappen Abriss dar, was Lernen bedeutet und in welche Bereiche es einzuordnen ist. Das folgende Kapitel beschäftigt sich ausführlich mit dem Gebiet des Reinforcement Learnings (RL) und führt die psychologischen Hintergründe dieses Bereiches ein. Später werden diese Grundlagen im informatischen Kontext wieder aufgegriffen und für maschinelles Lernen genutzt.

2.2.1.1 Reinforcement Learning - in der Psychologie

Man versteht unter Reinforcement Learning oder auch instrumentellem Lernen das Lernen einer Verbindung von Verhalten und seiner Konsequenz. Geschichtlich gesehen, bedeutete die Entdeckung dieser Form von Lernen einen Umstoß der vorhergehenden Forschung [21]. Bislang nahm man an, dass nur bereits bestehende Verhaltensweisen oder Reflexe mit einem zusätzlichen Reiz konditioniert werden können, wie beispielsweise im Experiment von Pavlov. Eines der ältesten und bekanntesten Experiment zu diesem Thema, von Thorndike Anfang des 20 Jahrhundert entworfen, zeigte jedoch, dass auch neue Verhaltensweisen gelernt werden können.
In dem Experiment von Thorndike wurde eine Katze in eine hölzerne Kiste gesetzt, deren Tür nur mit einem versteckten „Tip“-Mechanismus geöffnet werden konnte. Diese Kiste wird auch „Puzzle-Box“ genannt. In dem Versuch wird die Katze in diese Kiste gesetzt, und kann immer dann fliehen, wenn sie den Hebel zur Öffnung der Tür in Bewegung gesetzt hat. Während des Versuches wurde die Zeit genommen, bis die Katze geflohen

war. Nach jedem Durchlauf wurde die Katze wieder in die Kiste gesetzt. Man sollte dabei hervorheben, dass Thorndike ganz gezielt ein Verhalten ausgewählt hat, dass nicht zum natürlichen Verhaltensrepertoire einer Katze gehört.
Nachdem die Katze am Anfang an den Wänden der Kiste kratzte und versuchte die Kiste irgendwie zu verlassen, berührte sie durch Zufall den Hebel und die Tür öffnete sich. Aus diesem Grunde wird diese Form auch als„Trial and Error"- Lernen bezeichnet. Nach einer gewissen Zeit wurde dieses Verhalten immer häufiger von der Katze gezeigt und wurde stereotyp.
Thorndike untersuchte die Lernkurven der Katze nach mehren Durchläufen dieses Versuches und stellte dabei fest, dass die Katze nicht wie vorher angenommen einen „Aha"-Moment erlebte und dann das gewünschte Verhalten (Hebel drücken) ausübte, sondern über „Trial and Error" den Hebel berührte und durch die gelungene Flucht (Belohnung) dieses Verhalten immer häufiger zeigte [21]. Er erklärte dieses Verhalten über - wie er es nannte - das „law of effect". Hierunter verstand er, dass unterschiedliche Verhalten, die in einer bestimmten Situation ausgeübt werden, häufiger auftreten, wenn nach der Ausübung dieser Verhalten eine Belohnung/positive Bewertung erfolgt. Je größer die Belohnung, desto stärker ist die Verbindung zur jeweiligen Aktion in einer bestimmten Situation. Das „law of effect" gehört zu den grundlegenden Prinzipien des Reinforcement Learnings. Die Prinzipien lassen sich wie folgt zusammenfassen: Ein Verhalten in seiner Ausprägung, Auftreten und Wahrscheinlichkeit des Auftretens lässt sich verändern durch die Konsequenz des Verhaltens [21].
B.F. Skinner setzte diese Forschung 1938 fort und entwickelte Techniken um die Grundlagen des Reinforcement Learnings aufzuklären. Am berühmtesten ist wohl die von ihm entwickelte „Skinner-Box", in der er Tiere wie zum Beispiel Ratten und Tauben konditionierte. Dieser Aufbau erlaubt es, Konditionierungsexperimente sehr genau zu kontrollieren und zu überwachen. Häufig wurden in dieser Box „bar press-to-food" Experimente durchgeführt, in denen das Versuchstier einen Hebel bewegte und daraufhin Futter bekam. Nach einer gewissen Zeit zeigt das Tier dieses Hebel-Drück- Verhalten häufiger. Man spricht hierbei von positivem Reinforcement Learning oder auch Belohnungstraining.
Skinner legte die Grundlage für die theoretische Formulierung von Reinforcement Learning. Er postulierte, dass man das Reinforcement Learning als eine Funktion zwischen der Häufigkeit der Antwort (ein bestimmtes Verhalten) und der Höhe der Belohnung (positives Reinforcement) oder seiner Verzögerung oder seines Ablaufs ansehen kann [21]. Bevor nun die wichtigsten Parameter des Reinforcement Learnings eingeführt werden, sollte man die Belohnung oder auch das positive Reinforcement Signal genauer definieren. Eine positive Bestärkung (Reinforcement) liegt dann vor, wenn zwischen der Verhaltensantwort und der Präsentation der Bestärkung eine höhere Auftretenswahrscheinlichkeit vorliegt. Man sollte sich dabei vor Augen führen, dass eine Belohnung nicht unbedingt etwas materielles wie beispielsweise Futter sein muss, sondern dass auch ein Verhalten ein positives Reinforcement Signal darstellen kann. Diesen speziellen Fall nennt man auch soziales Reinforcement. Es ist bei sehr vielen sozialen Verhaltensweisen der grundlegende Lernmechnanismus. Auch bei uns Menschen werden bestimmte Verhaltensweisen von der Gesellschaft bevorzugt und dadurch mit mehr Aufmerksamkeit, Liebe oder Anerkennung

belohnt.
Die wichtigsten Parameter beim Reinforcement Learning sind Stärke der Belohnung, die Motivation des Lernenden und in welcher Zeitrelation zum Verhalten die Belohnung gegeben wurde. Die Stärke und die Qualität der Belohnung stehen natürlich in einem starken Zusammenhang mit der Motivation des Tieres. Wenn zum Beispiel eine Ratte ihren Weg in einem Labyrinth zum Ausgang finden soll, so wurde gezeigt, dass sie schneller am Ziel ist, wenn ihr das dargebotene Futter zusagt und sie zudem hungrig ist. Die wichtigste Variable bei dieser Form von Lernen ist jedoch der zeitliche Ablauf. Die besten Lernergebnisse erzielt man, wenn der Reward immer genau nach dem zu lernenden Verhalten gegeben wird. Sobald die Belohnung mit einer Verspätung gegeben wird oder regelmäßig nach dem erwünschten Verhalten erfolgt, sinkt die Häufigkeit des gewünschten Verhaltensmusters.
Wie bestärkt eine Belohnung das Lernen? Wenn man beachtet, wie unterschiedlich die Belohnungen hierbei sein können, fällt die Erklärung nicht leicht. Ob es nun Futter bei Tieren, das Spielzeug bei Kindern oder bei Erwachsenen soziale Anerkennung ist, alle diese Dinge können als Belohnung im Reinforcement Learning eingesetzt werden. Aber worin besteht ihre Gemeinsamkeit? Thorndike meinte dazu, dass als Reinforcer alles eingesetzt werden kann, was den Lernenden in einen "zufriedenen" Zustand versetzt. Skinner brachte eine noch allgemeinere Erklärung und postulierte, dass all das, was die Häufigkeit des Verhaltens erhöht, einen Reinforcer darstellt.
Zudem sollte man sich vor Augen führen, dass nicht alles durch instrumentelles Lernen gelernt werden kann. Zum Beispiel konnte gezeigt werden, dass man zwar einem Hamster beibringen kann einen Hebel zu drücken, um an Futter zu gelangen, aber es konnte nicht konditioniert werden, dass das Tier sich öfter das Gesicht wäscht [21].
Es gibt mehrere Theorien, die versuchen dieses Form von Lernen zu erklären. Die größten Unterschiede bei diesen Theorien findet man in der Kategorisierung der Belohnung an sich. Ob nun die Belohnung als reiner Stimulus gesehen wird, wie beispielsweise Futter oder Spielzeug, oder als eine Aktivität eingestuft wird, wie das Konsumieren oder Spielen, oder wiederum als eine Information, durch die man sein Verhalten einschätzen kann, war es gut oder schlecht, was man gemacht hat. Bis heute konnte keine Theorie bewiesen werden. Zudem konnte Tolman zeigen, dass auch ohne Belohnung die Ratten das Labyrinth durch Erkundung lernten.
Es sei noch erwähnt, dass in diesem Kapitel nicht auf negatives Reinforcement Learning eingegangen wurde. Allerdings sind die grundlegenden Theorie für beide Lernmethoden gleich, nur dass bei negativem Reinforcement Learning dieses Verhalten nach einer bestimmten Zeit vermieden wird.

2.2.2 Maschinelles Lernen

Im Allgemeinen spricht man von maschinellem Lernen, wenn man anhand von Beispieldaten oder Erfahrungen die Leistung des Rechners (Programms) optimiert [1]. Dies ist immer dann von Vorteil, wenn zwar Beispiele oder Erfahrungen existieren, aber nicht genau beschrieben werden kann, wie ein Problem zu lösen ist. Ein Beispiel hierfür ist die Spracherkennung. Wir alle verstehen gesprochene Sprache unabhängig davon wer die-

se Wörter gesprochen hat, wie schnell er sie gesprochen hat oder wie alt der Sprecher war. Dennoch können wir nicht erklären, wie genau dieser Prozess des Verstehens abläuft und wie man ein Programm entwickeln könnte, das ebenfalls diese Fähigkeit besitzt. Ein weiterer Grund für maschinelles Lernen ist, dass das Programm oder der Roboter flexibel auf Änderungen seiner Umgebungen reagieren sollte, das bedeutet, ein System sollte sich an veränderte Umstände anpassen können. Das Ziel bei maschinellem Lernen ist es, ein Modell mit Hilfe von Beispieldaten oder Erfahrungen zu entwickeln, das in der Lage ist auch unbekannte Daten oder Erfahrungen richtig zu bearbeiten oder in einer neuen Situation adäquat zu handeln. Abstrahiert betrachtet, wird das Wissen, das man aus der Welt erlangt hat, durch die erhaltenen Modelle dargestellt. Maschinelles Lernen ist aus unterschiedlichen Disziplinen zusammengesetzt. Zu diesen Disziplinen gehören beispielsweise Mustererkennung, künstliche Intelligenz, Neuronale Netzwerke, Signalverarbeitung und Data Mining. Grundlegend für all diese Disziplinen ist die statistische Analyse der gegebenen Daten [1]. Prinzipiell kann man maschinelles Lernen in drei Lernansätze aufteilen: überwachtes Lernen, unüberwachtes Lernen und bestärkendes Lernen (Reinforcement Learning). Bei überwachtem Lernen ist immer eine Menge von Eingabedaten gegeben und zu diesen auch eine entsprechende Ausgabe. Die Ausgabe kann bei einer Interpolation einer Funktion beispielsweise ihr Funktionswert sein, es kann aber auch eine Klassenzuweisung sein. In einer Klassifizierungsaufgabe soll über maschinelles Lernen eine Regel gefunden werden, die verschiedene Klassen unterscheiden kann. Diese Regel oder die Menge an gefundenen Regeln wird Diskriminante genannt. Durch sie soll es möglich sein auch Daten, die nicht zur Extrahierung der Regel genutzt wurden ihren Gruppen nach richtig einzuteilen. Bei einer Regression versucht man eine Funktion zu finden, die die gegebenen Datenpunkte möglichst genau beschreibt und darüber hinaus in der Lage ist Funktionswerte an neuen Orten vorauszusagen.
Im Gegensatz dazu steht das unüberwachte Lernen, bei dem zwar eine Menge von Eingabedaten bereitsteht, zu denen jedoch keine Ausgaben vorliegen. Das Ziel ist hierbei eine Regelmäßigkeit, Ähnlichkeit oder ein Muster in den Daten zu entdecken [1]. Ein Beispiel hierfür wäre ein Clusteralgorithmus, der die Eingabemenge in Gruppen aufteilt.
Beim Reinforcement Learning liegt eine Menge von Eingaben vor, wobei zu diesen nicht wie bei überwachtem Lernen ein exakter Ausgabewert existiert, sondern nur die Einordnung, ob das, was beispielsweise der Roboter gemacht hat, gut oder schlecht war. In diesem Bereich ist die Eingabemenge meist eine Sequenz von Aktionen und der Roboter erhält am Ende eines Durchlaufes eine Bewertung, ob sein gezeigtes Verhalten gut oder schlecht war. Durch dieses Verfahren soll der Roboter sein Wissen durch Erfahrungen mit seiner Umgebung erlangen [1]. Auf Reinforcement Learning wird im folgenden Abschnitt noch ausführlich eingegangen.
Viele Ideen und Grundlagen für des maschinelle Lernen entstammen Modellen aus der Psychologie oder haben ihr Vorbild in den Neurowissenschaften, beispielsweise das Reinforcement Learning oder die Neuronalen Netzwerke. Nicht alle Methoden haben ihr Vorbild in der Biologie oder Psychologie oder versuchen gar die Grundlage des Lernens auf diese Weise zu erforschen. Aber die Erforschung der Grundlagen des Lernens war schon immer ein Antrieb auch im maschinellen Lernen.

2.2.2.1 Reinforcement Learning - in der Informatik

*„Es erscheint leichter und viel intuitiver für einen Programmierer zu spezifizieren **was** der Roboter tun sollte und ihn die genauen Details (**wie**) er es tun soll, lieber selbst lernen zu lassen.“*[18][2].
Im Gegensatz zum überwachten Lernen muss der „Lehrer“ nicht alle Aktionen des Roboters bewerten und dem Roboter das Eingabe-Ausgabe-Paar zum Lernen vorgeben, sondern den Erfolg einer Sequenz von Aktionen bewerten. Dies ist bei manchen Problemstellungen wesentlich einfacher. Ein gutes Beispiel hierfür sind Spiele. Bei einem Spiel wie Schach ist es sehr schwer einen einzelnen Spielzug zu bewerten, da er im Kontext von bereits ausgeführten und den noch kommenden Spielzügen steht. Beim Reinforcement Learning ist die Bewertung der Sequenz von Aktionen sehr intuitiv, wenn der Roboter das Spiel gewonnen hat, erhält er eine Belohnung. Ähnlich wie die Katze in der „Puzzle Box“ lernt der Roboter über Ausprobieren („Trial-and-Error Learning“). Jeder erlaubte Spielzug stellt eine Aktion dar. Wenn das Spiel gewonnen wird, erhält der Roboter eine Belohnung. Durch seine Erfahrung lernt der Roboter, welche Aktionen in welcher Situation erfolgreich waren. Hierzu ordnet er jedem Zustands-Aktionspaar einen numerischen Wert zu, der angibt, wie gut diese Aktion in diesem Zustand war.
Die Idee dahinter ist, wie auch schon in der psychologischen Sichtweise erläutert, dass Verhalten, die durch eine Belohnung bestärkt werden, häufiger auftreten. Im Reinforcement Learning lernt man eine Entscheidungsstrategie - welche Aktion in welcher Situation erfolgen soll - um die maximale Belohnung zu bekommen.
Die nun folgenden Ausführungen über Reinforcement Learning orientieren sich stark am Buch ”Reinforcement Learning” von Sutton und Barto [20]]. Falls andere Quelle genutzt wurden, wird explizit darauf hingewiesen.
Um nun diese Grundelemente und -methoden vorstellen zu können, sollen vorweg einige Definitionen erfolgen. Hierzu wird das allgemeine Reinforcement Problem definiert. Anschließend werden die einzelnen Grundelemente und Methoden erläutert.

Reinforcement Learning Problem Unter dem Reinforcement Learning Problem versteht man die Formulierung der Lernaufgabe in einer solchen Form, dass die Aufgabe mit Reinforcement Learning gelöst werden kann. Hierzu ist es notwendig das Problem so zu formulieren, dass es möglich ist, über Interaktion mit der Umgebung Erfahrungen zu sammeln und damit die Lernaufgabe zu lösen.
Der Entscheidungstreffende wird Agent genannt und der Raum seiner ausgeübten Aktionen Umgebung. Die Interaktion mit der Umgebung erfolgt über Aktionen, die vom Agenten ausgeübt werden. Durch jede Aktion erreicht der Agent einen neuen Zustand. Die Umgebung liefert je nach gewählter Aktion eine Bewertung der Aktion. Diese Belohnung einer Aktion wird im Folgenden als Returnwert r_t bezeichnet. Durch diesen Returnwert r_t wird die erwünschte Lernaufgabe definiert.
Ein Durchlauf erfolgt in diskreten Zeitschritten $t = 0, 1, 2, 3....$ In jedem Zeitschritt t er-

[2]Aus dem Englischen: *„It seems easier and much more intuitive for the programmer to specify what the robot shoud be doing, and to let it learn the fine details of how to do it”*

hält der Agent die Information über den jetzigen Zustand $s_t \in S$ durch die Umgebung, wobei S die Menge der möglichen Zustände ist. Auf Grund dessen übt der Agent eine Aktion, $a_t \in A(s_t)$ aus, wobei $A(s_t)$ die Menge der möglichen Aktionen in diesem Zustand s_t ist. Einen Zeitschritt später erfolgt die Konsequenz der Folgeaktion a_t, der Returnwert $r_{t+1} \in R$ für diese Aktion. Der Agent befindet sich nun in Zustand s_{t+1}. Abbildung 2.2 zeigt schematisch diesen Ablauf.

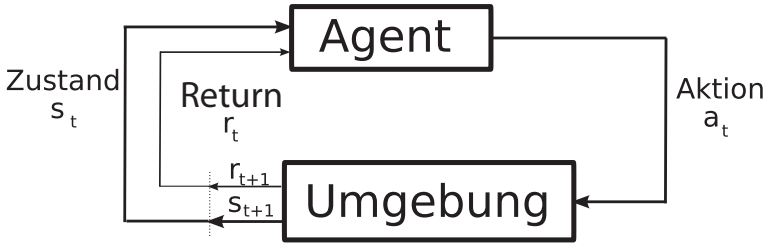

Abbildung 2.2: **Agent-Umgebung Interaktion**

In jedem Zeitschritt muss sich der Agent nun entscheiden, welche Aktion er im jetzigen Zustand ausüben will. Die Auswahl der Aktion ist abhängig von der Entscheidungsstrategie $\pi_(s, a)$. $\pi_t(s, a)$ gibt dabei die Wahrscheinlichkeit an die Aktion a zu wählen, wenn man in Zustand s ist. Die Strategie ändert sich während der Lernphase und passt sich den Erfahrungen des Agenten an.
Die Formulierung des Reinforcement Learning Problems ist so abstrakt und flexibel, dass viele Probleme entsprechend beschrieben werden können. Aktionen können je nach Problemstellung einfache Kontrollaufgaben, wie die Spannung für einen Motorarm, darstellen oder anspruchsvolle Entscheidungen, wie zum Beispiel die Interaktion mehrerer Roboter beim Robocup. Auch die Wahl, wie ein Zustand dargestellt ist, ist in diesem Rahmen sehr flexibel wählbar. Ob es nun einzelne Sensordaten oder wie im Fall dieser Diplomarbeit die Auswertung von Navigationsstrategien sind, obliegt dem Entwickler. Allgemein kann man sagen, alles was nützlich sein könnte, kann als Zustand definiert werden und alles was eine Entscheidung darstellt, kann als Aktion gewählt werden. Für die Anwendung von Reinforcement Learning müssen somit drei Informationssignale, die Aktion, der Zustand und die Belohnung definiert sein, die die Lernaufgabe möglichst gut beschreiben.

Grundlegende Elemente des Reinforcement Learnings Die Grundelemente des Reinforcement Learnings sind: die Strategie, die Belohnung, die Rewardfunktion und die Valuefunktion. Für den theoretischen Hintergrund wird zudem die Markov-Eigenschaft vorgestellt. Dabei werden die englischen Fachtermini verwendet, wenn eine adäquate Übersetzung schwierig ist.

Strategie Die Strategie beim Reinforcement Learning definiert das Verhalten des Agenten. Allgemein kann man sagen, dass die Strategie eine Zuordnung darstellt, die beschreibt welche Aktion der Agent in welchem Zustand wählen sollte, um eine Belohnung zu erhalten. Korrespondierend mit dem psychologischen Ansatz wäre die Strategie die Assoziationsregel zwischen Verhalten und Konsequenz. Die Strategie stellt somit den

Kern des Reinforcement Learnings dar, da sie darüber entscheidet ob der Agent das richtige Verhalten gelernt hat. Ziel der Lernaufgabe ist es die optimale Strategie π^* zu bestimmen. Beschrieben wird die Strategie mit π_t. Sie gibt die Wahrscheinlichkeit an, wie wahrscheinlich es ist im Zeitschritt t und im Zustand s_t die Aktion a_t auszuführen, $\pi_t(s, a)$.

Belohnung Beim Reinforcement Learning ist es entscheidend, was das Ziel der Lernaufgabe darstellt und welches Verhalten der Agent somit lernen soll. Die einzige Information, die der Agent zur Lösung seiner Lernaufgabe erhält, ist die Belohnung. Hierzu liefert die Umgebung in jedem Zeitschritt einen Zahlenwert $r_t \in R$, der Return genannt wird. Der Agent hat nun das Ziel die Summe an Belohnungen über die Zeit zu maximieren.
Die Beschreibung des Lernziels durch Belohnung stellt in vielen Fällen eine Vereinfachung dar. Zum Beispiel kann man so das Durchqueren eines Labyrinths leicht als Lernaufgabe formulieren. Immer wenn der Roboter den Ausgang findet, erhält er eine Belohnung von $r_t = +1$. Läuft er gegen eine Mauer erhält er eine Bestrafung von $r_t = -1$ und ansonsten ist $r_t = 0$.
Allerdings sollte man bei dem Versuch Vorwissen in die Belohnung mit einfließen zu lassen vorsichtig sein. Beim Schach beispielsweise könnte man als geübter Schachspieler bestimmte Zwischenziele definieren und diese belohnen. Dies könnte allerdings dazuführen, dass der Agent nur lernt die Zwischenziele zu erreichen, dennoch aber nicht gut Schach spielt.

Rewardfunktion Die Rewardfunktion ordnet jedem Zustands-Aktionspaar einen Wert zu, der die erwartete Belohnung dieses Paares beschreibt. Dieser numerische Wert wird Reward R_t genannt. Das Ziel des Agenten ist nun den über eine Sequenz summierten Reward zu maximieren. Die Rewardfunktion beschreibt somit den Wert eines Zustands-Aktionspaares, bezogen auf die gesamte Sequenz.
Die Berechnung des Rewards hängt davon ab, ob die Lernaufgabe in Episoden unterteilt werden kann, wie beispielsweise beim Schach, oder ob es sich um eine zeitlich unbegrenzte Lernaufgabe handelt. Ein Beispiel einer zeitlich unbegrenzten Lernaufgabe ist das Balancieren eines Stabes (pole-balancing task). Bei dieser Aufgabe soll ein beweglicher Roboter eine auf ihm befestigte Stange balancieren. Diese Aufgabe ist auch dann nicht abgeschlossen, wenn der Agent die Stange über sich balanciert, da die Stange immer wieder, je nach Bewegung des Roboters, instabil werden kann. Somit kann man zwar ein Zeitfenster definieren, wie lange der Agent diese Aufgabe lösen soll, aber man kann nicht, wie bei der Durchquerung eines Labyrinths davon sprechen, dass die Aufgabe abgeschlossen ist. Um auch zeitlich unbegrenzte Aufgaben durch die Rewardfunktion zu beschreiben, wird eine weitere Variable eingeführt, die *Discounting Rate* γ. Somit lässt sich die Rewardfunktion wie folgt beschreiben:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad 0 \leq \gamma \leq 1 \tag{2.1}$$

Dieser Parameter definiert ob der Agent "kurzsichtig" in die Zukunft schaut, das bedeutet, dass später erhaltene Returns nicht in der Beurteilung berücksichtigt werden oder ob der Agent eher weitsichtig auch nachfolgende Returns in seine Bewertung mit aufnimmt. Wird die Discounting Rate auf eins gesetzt, fließen alle Returns gleich stark in die Berechnung des Rewards ein. In diesem Fall handelt es sich um einen „normalen" Zeithorizont.

Markov-Eigenschaft & Markov-Entscheidungsprozess Für die theoretische Analyse des Reinforcement Learnings ist die Markov Eigenschaft essentiell. In dieser kurzen Beschreibung wird hierbei nur der diskrete Fall berücksichtigt. Die Entscheidungsdynamik im Reinforcement Learning besitzt dann die Markov Eigenschaft, wenn gilt, dass in einer gegebenen Umgebung das Erreichen des Zustands s_{t+1} im Zeitschritt $t+1$ nur von der Übergangswahrscheinlichkeit Pr des letzten Zustands s_t mit der gewählten Aktion a_t abhängt. Die komplette Übergangswahrscheinlichkeit ist definiert durch:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1} \cdots, r_1, s_0, a_0\} \tag{2.2}$$

wenn die Markov-Eigenschaft gilt, heißt dass:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{2.3}$$

Wenn die Markov Eigenschaft gilt, bedeutet das, dass der nächste Zustand und der nächste erwartete Reward nur durch den jetzigen Zustand und die gewählte Aktion bestimmt wird. Das hat zur Konsequenz, dass nicht die gesamte Historie benötigt wird, um den nächsten erwarteten Reward vorauszusagen.
Gilt die Markov Eigenschaft in einem Entscheidungsprozess kann dieser auch als ein Markov-Entscheidungsprozess verstanden werden. Sind sowohl Zustands- als auch Aktionsmenge endlich, spricht man von einem endlichen Markov Entscheidungsprozess. Dies ist wichtig für die Theorie hinter dem Reinforcement Learning. Ein Markov Entscheidungsprozess ist definiert durch eine Menge an Zuständen und Aktionen. Bei einem gegebenen Zustand s und einer Aktion a, wird die Übergangswahrscheinlichkeit für den nächstmöglichen Zustand wie folgt beschrieben:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.4}$$

Durch einen Markov Entscheidungsprozess ist nicht nur die Übergangswahrscheinlichkeit des nächsten Zustandes gegeben, sondern es lässt sich auch der Erwartungswert E für den Reward berechnen:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.5}$$

Value/Q-Funktion Die Valuefunktion beschreibt den erwarteten Reward eines Zustandes unter einer gegebenen Entscheidungsstrategie in einer bestimmten Sequenz und stellt somit das erworbene Wissen des Agenten dar. Durch sie bestimmt der Agent die Wahl der nächsten Aktion, so dass die Summe der Rewards maximiert wird. Die Berechnung des Values ist im Gegensatz zur Berechnung des Rewards schwierig, da die

Valuefunktion nur über Erfahrungen des Agenten geschätzt werden kann. Nur eine gute Schätzung der Valuefunktion führt zu einem Lernerfolg.
Die Valuefunktion ist stets über die gelernte Strategie π definiert und wird wie folgt beschrieben:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \tag{2.6}$$

Hierbei ist $E_{\pi}\{\}$ der erwartete Rewardwert des Zustands s_t, der durch die Strategie π gegeben ist. Man bezeichnet somit V^{π} als Zustands-Valuefunktion der Strategie π.
Man kann auch diese Funktion für Zustands-Aktionspaare definieren. Diese Form der Valuefunktion wird Q-Funktion genannt. Auch hier gilt, dass sie für eine Strategie π definiert ist. Die Q-Funktion ist definiert als:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \tag{2.7}$$

Q^{π} wird auch als Aktions-Valuefunktion der Strategie π bezeichnet.
V^{π} oder Q^{π}, werden über Erfahrungen des Agenten mit seiner Umgebung abgeschätzt. Zur Bestimmung der Value/Q-Funktion können unterschiedliche Methoden wie Monte-Carlo oder dynamisches Programmieren benutzt werden. Diese Methoden werden im folgenden Abschnitt vorgestellt.
Die Lernaufgabe ist gelöst, wenn man eine Strategie gefunden hat, die den maximalen Reward liefert. Mit Hilfe von $V^{\pi}(s)$ lassen sich alle möglichen Strategien π durch ihre Abschätzung des maximalen Rewards ordnen, da gilt, dass eine Strategie π nur dann besser ist als π' wenn und nur wenn $V^{\pi}(s) \leq V^{\pi'}(s) \forall s \in S$ ist. Die optimale Strategie wird als π^* bezeichnet, somit ist die optimale Zustands-Valuefunktion V^* folgendermaßen definiert:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.8}$$

Entsprechend hierzu ist auch die optimale Aktions-Valuefunktion definiert:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.9}$$

Über die Bellmann-Optimalitätsgleichung kann man beweisen, dass die optimale Strategie über iteratives Verbessern der Valuefunktion gefunden werden kann. Die Bellmann-Optimalitätsgleichung besagt, kurz formuliert, dass eine optimale Lösung aus optimalen Teillösungen zusammengesetzt werden kann. Dieser Optimalitätsbeweis ist für die theoretische Analyse der Reinforcement Learning-Verfahren unabdingbar. Für die genaue Ausführung dieses Beweises wird auf das Buch „Reinforcement Learning“ von R. Sutton und A. Barto [20] verwiesen.

Lösungsmethoden Um das Reinforcement Learning-Problem zu lösen können drei grundlegende Methoden verwendet werden: das dynamische Programmieren (DP), die Monte Carlo-Methode (MC-Methode) und das Temporal-Difference Learning (TD(λ)-Learning). Diese drei Methoden sollen im Folgenden kurz beschrieben und ihre Vor- und Nachteile herausgearbeitet werden.

Dynamisches Programmieren Voraussetzung für die Anwendung von Dynamischem Programmieren ist die Markov-Eigenschaft der Umgebung. Für die theoretische Analyse waren und sind die DP-Algorithmen sehr wichtig. Allerdings ist die praktische Anwendung auf Grund ihres hohen Rechen- und Speicheraufwands sehr schwierig.
Die Grundidee bei allen DP-Ansätzen ist, dass man durch die Valuefunktion die Suche nach der optimalen Strategie organisiert. Alle DP-Ansätze basieren auf der Bellmann-Optimalitätsgleichung. Ein grundlegender Algorithmus ist hierbei die *Strategie Evaluation* (E), mit der man iterativ die Valuefunktion für eine gegebene Strategie berechnen kann:

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}|s_t = s)\} \tag{2.10}$$
$$\tag{2.11}$$

Durch diese iterative Formel ist es möglich die Valuefunktion für diese Strategie anzunähern. Hierbei ist V_0 frei wählbar.

$$V_0 \to V_1 \to V_2 \cdots \to V_k \to V_{k+1} \to \cdots V^\pi \tag{2.12}$$

Für $k \to \infty$ kann gezeigt werden, dass die Funktion gegen V^π konvergiert. Wenn man nun die Valuefunktion V^π für eine gegebene Strategie π berechnet hat, stellt sich die Frage, ob es noch eine bessere Strategie π' gibt. Um dies zu berechnen, wendet man die *Strategie Überprüfung* (I) an. Hierbei wird überprüft, ob es in einem gegebenen Zustand besser ist, eine andere Aktion zu wählen. Um nun die verbesserte Strategie π' zu finden, überprüft man für jeden Zustand s, ob es eine bessere Aktion gibt:

$$\pi'(s) \quad = \quad \arg\max_a Q^\pi(s, a) \tag{2.13}$$
$$\tag{2.14}$$

Werden nun *Strategie Evaluation* und *Strategie Überprüfung* hintereinander ausgeführt, spricht man von der *allgemeinen Strategie Iteration* (engl: Generalized Policy Iteration = GPI). Hierbei erfolgt zunächst die Auswertung der gegebenen Strategie π, woraufhin überprüft wird, ob es eine bessere Strategie π' gibt. Wenn dies der Fall ist, wird mit der neuen Strategie weiter gerechnet.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \pi^* \xrightarrow{E} V^* \tag{2.15}$$

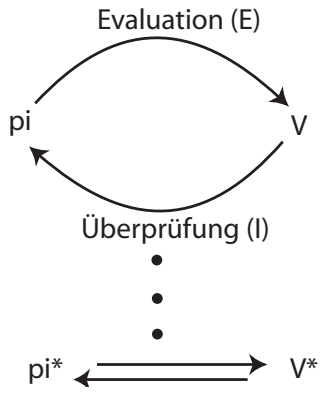

Abbildung 2.3: **Schematischer Ablauf der Strategie Iteration**

Abbildung 2.3 zeigt den schematischen Ablauf der Strategie Iteration.
Das Problem bei DP ist, dass die Berechnung der optimalen Strategie polynominal mit Anzahl der Zustände zunimmt und somit für große Zustandsräume auf Grund der hohen Rechenzeit und des großen Speicheraufwands nicht genutzt werden kann. Ein weiterer großer Nachteil ist, dass diese Methode darauf basiert, dass man einen perfekten Markov-Entscheidungsprozess vorliegen hat, da durch das iterative Verfahren immer die Schätzung des k-ten Schrittes von der Schätzung des $k-1$-ten Schrittes abhängt. Wird nun in einem Iterationsschritt der Wert ungenau geschätzt, summiert sich dieser Fehler auf. Dies ist oft der Fall wenn kein Markov-Entscheidungsprozess vorliegt.

Monte Carlo-Methode Im Unterschied zum Dynamischen Programmieren, erfordert die Monte Carlo-Methode kein Wissen über die Dynamik der Umgebung. Die Monte Carlo-Methode benötigt nur Erfahrungen, die der Agent bislang gesammelt hat. Die Lösung des Reinforcement Learning-Problems basiert nun auf der Mittelung der gesammelten Returns. Um sicher zu stellen, dass die erhaltenen Returns auch wohl definiert sind, ist die Monte Carlo-Methode nur für episodische Aufgaben definiert.
Ähnlich wie beim Dynamischen Programmieren erfolgt auch bei dieser Methode zuerst die *Strategie Evaluation*. Die Grundidee bei dieser Methode ist nun, die Valuefunktion über die Erfahrungen abzuschätzen, was in diesem Falle bedeutet, die einzelnen Returns in einem Zustand zu mitteln. Je häufiger dieser Zustand besucht wurde, desto genauer beschreibt das gemittelte Ergebnis den Wert der Valuefunktion für diesen Zustand.
Es gibt zwei Ansätze die Returns zu mitteln. Der erste wird *"Every-Visit MC"* genannt. Bei diesem Ansatz werden alle Returns, die einem Zustand zugeordnet sind, gemittelt. Bei *"First-visit MC"* werden nur die Returns für die Mittelung benutzt, die erhalten wurden, als der Zustand in einer Episode das erste Mal besucht wurde. Für beide Ansätze kann gezeigt werden, dass sie asymptotisch gegen den Mittelwert konvergieren. Im Gegensatz zum DP ist hier die Schätzung der erwarteten Returns für ein Zustand unabhängig von den Schätzungen der anderen Zustände. Dies stellt einen großen Vorteil gegenüber dem DP dar, da hierfür die Markov-Eigenschaft nicht notwendig ist.
Bei der MC-Methode ist das Ziel die Abschätzung der Q-Funktion. Hierdurch kann wiederum, wie auch beim DP, das Prinzip der *allgemeinen Strategie Iteration* (GPI) verwen-

det werden. Ähnlich wie beim DP ergibt sich daraus das Wechselspiel zwischen *Strategie Evaluation* - und *Überprüfung*:

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \pi^* \xrightarrow{E} Q^* \tag{2.16}$$

Bei der *Strategie Überprüfung* wird für jeden Zustand die Aktion gewählt, die den Wert der Q-Funktion maximiert („greedy-policy“):

$$\pi(s) = \arg\max_a Q(s,a) \tag{2.17}$$

Temporal-Difference Learning Temporal-Difference Learning (TD-Learning) stellt eine neuere Entwicklung des Reinforcement Learnings dar. Es beinhaltet Ideen sowohl aus dem Dynamischen Programmieren als auch von der Monte Carlo-Methode. Wie bei MC lernt diese Methode nur aufgrund von Erfahrungen. Allerdings werden, wie beim DP, die Schätzungen über die Valuefunktion über einen Lernvorgang berechnet und nicht wie bei MC durch eine Mittelung der Returnwerte.
Wie bei MC und DP steht auch hier das Zusammenspiel zwischen *Strategie Evaluation* und *Überprüfung* im Vordergrund. Bei der *Strategie Evaluation* wird ähnlich wie bei MC die benötigte Information aus den gewonnenen Erfahrungen des Agenten extrahiert. Allerdings muss nicht wie bei MC die Episode abgewartet werden bevor man den Durchgang auswerten kann, sondern die Auswertung im nächsten Zeitschritt. Bei der einfachsten Form von TD Learning, auch TD(0) Learning genannt, wird die Auswertung wie folgt berechnet:

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.18}$$

Dadurch hat dieses Verfahren den Vorteil, dass es nicht eine Episode abspeichern muß, sondern schon während des Lernens die Erfahrung auswerten kann. Auch hier konnte gezeigt werden, dass dieses Vorgehen gegen die Valuefunktion konvergiert. Die Konvergenzgeschwindigkeit ist dabei von der Wahl des Lernschrittes α abhängig.
Ein bekannter Algorithmus für das online Lernen der Q-Funktion wird Sarsa genannt. Der Name stammt von dem benötigten Quintupel, das für das Update der Q-Funktion nötig ist, $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. Die Q-Funktion wird folgendermaßen berechnet:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.19}$$

Ein anderes bekanntes offline Verfahren wird Q-Learning genannt. Hierbei wird die Q-Funktion wie folgt berechnet:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.20}$$

Nach Berechnung der Q-Funktion, erfolgt die Überprüfung der Strategie. Beim Temporal-Difference Learning wird hierbei der TD-Fehler zur Formalisierung der Lernregel verwendet:

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.21}$$

Wenn der Fehler positiv ist, zeigt dies, das die ausgewählte Aktion a_t im Zustand s_t bevorzugt in diesem Zustand genommen werden sollte, wenn er negativ ist sollte die Präferenz $p(s_t, a_t)$ für diese Aktion sinken. Über diese Angabe kann man die gegebene Strategie verbessern:

$$\pi_t(s,a) = Pr\{a_t = a | s_t = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}} \tag{2.22}$$

Die Präferenz $p(s_t, a_t)$ wird am Anfang zufällig initialisiert und während des Lernens wie folgt neu umgeändert:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t \tag{2.23}$$

Die Lernrate β definiert hierbei, wie stark ein neuer Wert berücksichtigt werden soll.
Die Vorteile dieser Methode sind zum einen, dass sie gegenüber MC einen geringeren Speicheraufwand besitzt, da man nicht einen gesamten Durchlauf abspeichern muss. Zum anderen konnte gezeigt werden, dass diese Verfahren schneller konvergieren. Allerdings tritt ähnlich wie beim DP das Problem auf, dass die Schätzungen eines Schrittes vom vorherigen abhängen, so dass sich durch fehlerhafte Schätzungen ein systematischer Fehler akkumuliert.

Erweiterung auf den kontinuierlichen Raum Im traditionellen Reinforcement Learning wurde bislang nur mit diskreten Zustands- und Aktionsräumen gearbeitet. Dies stellt für viele Anwendungen ein Problem dar. Gerade im Bereich Navigation, wo sich Reinforcement Learning sehr gut als Kontrollsystem eignet, sind die Zustands- und Aktionsräume häufig kontinuierlich. Eine mögliche Lösung ist den kontinuierlichen Raum zu diskretisieren. Allerdings sollte man hierbei bedenken, dass man durch eine "schlechte" bzw. zu grobe Diskretisierung des Zustands-Aktionsraumes Zustände verdecken kann, was dazu führt, dass die optimale Strategie nicht gefunden werden kann [17]. Wenn der Raum zu genau abgetastet wird, verliert das System die Fähigkeit der Generalisierung und die Menge an Trainingsdaten wächst sehr stark an [17]. Gerade bei multidimensionalen Räumen ist das der Fall. Hier steigt die Anzahl der diskretisierten Zustände exponentiell mit den Dimensionen des Raumes.
Eine andere Möglichkeit besteht darin, die Valuefunktion kontinuierlich zu approximieren. Im Folgenden soll nun ein Verfahren zur Funktionsapproximation detailliert vorgestellt werden. Für die Approximation wird ein Radiale-Basisfunktionen-Netzwerk verwendet.

2.2.2.2 Radiale-Basisfunktionen Netzwerk (RBF-Netzwerk)

Maschinelles Lernen wird häufig als Funktionslernen beschrieben. Gerade in der Approximation von Funktionen werden sehr häufig künstliche neuronale Netze eingesetzt. Man versucht durch diese Netzwerke herauszuarbeiten, welche Zusammenhänge zwischen den Eingabe- und Ausgabepaaren bestehen.
Ein Radiale-Basisfunktionen-Netzwerk ist ein vorwärts gerichtetes Netzwerk, das nur eine Schicht verdeckter Neurone besitzt. Diese Neurone besitzen radialsymmetrische Aktivierungsfunktionen. Ein RBF-Netz kann das Interpolationsproblem lösen:

Interpolationsproblem: Gegeben N unterschiedliche Eingabepunkte $\{\boldsymbol{x}_j \in I\!R^N |\ j = 1, \cdots n\}$ und N Ausgabewerte $\{y_j \in I\!R | j, \cdots N\}$. Gesucht ist nun die Funktion F aus $I\!R^N \rightarrow I\!R$ die die folgende Interpolationsbedingung erfüllt:

$$F(\boldsymbol{x}_j) = y_j \tag{2.24}$$

Zur Lösung des Interpolationsproblems benötigt man die gleiche Anzahl N von Basisfunktionen wie Datenpunkte. Das radiale Basisfunktionsnetzwerk besteht dann aus einer Linearkombination von Basisfunktionen:

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \phi_i(\boldsymbol{x}) \tag{2.25}$$

Man hat nun ein Gleichungssystem mit N Gleichungen zu lösen:

$$y_j = \sum_{i=1}^{N} c_i \phi_i(\boldsymbol{x_j}) \quad j = 1, \cdots, N \tag{2.26}$$

Definiert man nun die Vektoren $\boldsymbol{y}$ und $\boldsymbol{c}$ und die Matrix H wie folgt:

$$(\boldsymbol{y}) = y_j, \quad (\boldsymbol{c}) = c_i, \quad (H_{ij}) = \phi_i(\boldsymbol{x_j}) \tag{2.27}$$

Dann läßt sich das Gleichungssystem somit folgendermaßen beschreiben:

$$\boldsymbol{c} = H^{-1}\boldsymbol{y} \tag{2.28}$$

Abbildung 2.4 zeigt eine Beispielapproximation. Man sieht in dieser Abbildung, wie die Linearkombination der Basisfunktionen (dünne schwarze Linien) die ursprüngliche Funktion (in rot dargestellt) approximiert.

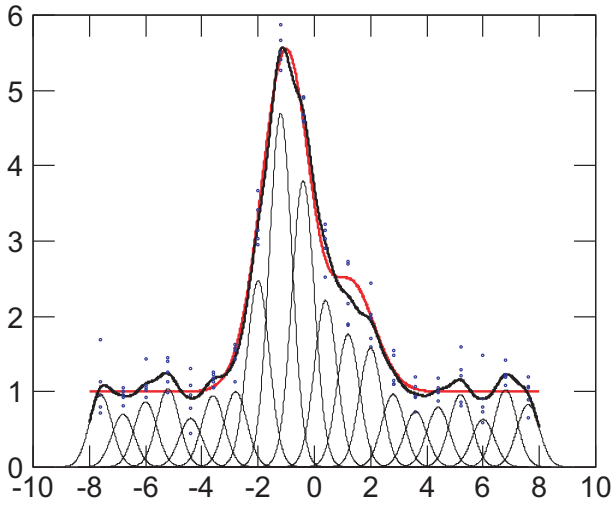

Abbildung 2.4: **Beispiel für ein Approximation** Die gegebenen Datenpunkte sind als blaue Punkte dargestellt. Die Ursprungsfunktion ist rot, die approximierte Funktion schwarz gekennzeichnet. Zudem sind die einzelnen Basisfunktionen (Gaußkurven) durch dünnere schwarze Linien eingezeichnet.

Die Basisfunktionen müssen die Eigenschaft besitzen, dass man die Inverse der Matrix H berechnen kann. Dies erreicht man beispielsweise, wenn man verlangt, dass die Basisfunktionen positiv definite Funktionen sind. Die nun aufgelisteten Funktionen können somit als Basisfunktion eingesetzt werden:

$$\phi(r) = \frac{1}{\sigma\sqrt{2\,\pi}}\, e^{\frac{||\mu - r||}{2\sigma^2}} \tag{2.29}$$

$$\phi(r) = \frac{1}{(c^2 - r^2)^\alpha} \quad \alpha > 0 \tag{2.30}$$

$$\phi(r) = (c^2 + r^2)^{beta} \quad 0 < \beta < 1 \tag{2.31}$$

$$\phi(r) = \sqrt{r^2 + c^2} \tag{2.32}$$

$$\phi(r) = r \tag{2.33}$$

Allerdings liegt oft der Fall vor, dass der gegebene Datensatz viel größer als die Anzahl an Basisfunktionen ist. Ein weiterer Grund ist, dass die Datenpunkte verrauscht sind, so dass eine Interpolation nicht erwünscht ist. In diesem Falle kann man das RBF auch für eine Approximation nutzen. Hierzu wird ein überbestimmtes Gleichungssystem aufgestellt, wobei es N Gleichungen für K Unbekannte gibt:

$$F(\boldsymbol{x}) \approx \sum_{i=1}^{K} = c_i \phi_i(\boldsymbol{x}) \tag{2.34}$$

Zu lösen sind nun die Gleichungen:

$$y_j = \sum_{i=1}^{K} c_i \phi_i(\boldsymbol{x_j}) \quad j = 1, \cdots, N \tag{2.35}$$

Daraus ergibt sich folgende Matrixschreibweise:

$$\boldsymbol{c} = H^{+}\boldsymbol{y} \tag{2.36}$$

H^{+} wird auch Moore-Penrose Pseudoinverse von H genannt und wie folgt berechnet:

$$H^{+} = (H^{\top}H)^{-1}H^{\top} \tag{2.37}$$

Die Berechnung der Koeffizienten über die Inverse ist sehr zeitintensiv, da die Komplexität polynominal mit der Anzahl der Datenmenge N zunimmt (ca N^3). Daher greift man oft zur Koeffizientenbestimmung auf Relaxationsverfahren wie z.B. Gradientenabstiegsverfahren oder konjugierte Gradientenabstiegsverfahren zurück.

Kapitel 3

Problemstellung

Ausgangspunkt für die Diplomarbeit ist die Fragestellung, ob eine Subsumptionsarchitektur zur Integration von lokalen Navigationsstrategien durch einen Lernansatz ersetzt werden kann. Der Grund für diese Fragestellung ist, dass die Subsumptionsarchitektur Probleme aufweist, die dem System innewohnen und nur schwer vermieden werden können. Die Ursachen der Probleme lassen sich auf die einzelnen verwendeten Navigationsstrategien zurückführen. In der Subsumptionsarchitektur ist es vorgesehen stets nur eine Navigationsstrategie als Bewegungsentscheidung zu wählen, daher können die Fehler der gewählten Strategie nicht durch die anderen Strategien ausgeglichen werden. Dieses Vorgehen führt zu unflexiblen Verhaltensweisen. Um dies zu vermeiden, soll die Subsumptionsarchitektur durch ein Lernverfahren ersetzt werden. Hierfür wurde ein Reinforcement Learning-Ansatz gewählt, da bei diesem Verfahren der Agent über Erfahrung lernen kann die Navigationsstrategie situationsabhängig zu gewichten.
In diesem Kapitel wird die Navigationsaufgabe vorgestellt und erläutert, wie die Subsumptionsarchitektur die Navigationsstrategien integriert. Danach werden die Fehlerquellen der einzelnen Navigationsstrategien kurz besprochen und ihre Auswirkungen auf die Navigationsleistung werden aufgezeigt. Am Ende des Kapitels wird die Wahl des Reinforcement Learning-Ansatzes als Alternative zur Subsumptionsarchitektur begründet.

3.1 Navigationsaufgabe

Die Basis für die Navigationsaufgabe stammt aus der Doktorarbeit von W. Hübner [10]. In dieser Doktorarbeit wurde ein System vorgestellt, in dem ein autonom navigierender Roboter ein Ortsgedächtnis aufbaut. Dieses Ortsgedächtnis wird als ein metrisch eingebetteter Ansichtsgraph repräsentiert. Der Aufbau des Ortsgedächtnisses wird durch die Kombination von drei lokalen Navigationsstrategien ermöglicht: dem visuellen Homing, der Wegintegration und der Hindernisvermeidung. Diese Strategien erlauben die Exploration und Kartierung von unbekannten Gebieten. Durch diesen Aufbau kann der Agent komplexere Navigationsleistungen wie zum Beispiel das Planen von Routen oder das Finden von Abkürzungen durch unbekanntes Gebiet zeigen [10].
Eine zentrale Rolle beim Aufbau des Graphmodells hat dabei ein robustes Heimfindungs-

verhalten. Das Heimfinden - auch Homing genannt - wird über die Integration der drei Navigationsstrategien ermöglicht. Durch die Kombination von Wegintegration und visuellem Homing kann der Wegintegrationsfehler mittels visueller Landmarkeninformation korrigiert werden. Die Integration dieser Navigationsstrategien geschieht über eine Subsumptionsarchitektur. Durch diese Subsumptionsarchitektur folgt der Roboter in einem Zeitschritt nur dem Bewegungsvorschlag einer Navigationsstrategie. Durch dieses Vorgehen entsteht allerdings das Problem, dass Fehlerquellen einer Strategie nicht durch andere Strategien ausgeglichen werden können.
Diese Diplomarbeit stellt nun einen Ansatz vor, bei dem der Agent die Integration der lokalen Navigationsstrategien selbstständig lernt. Das Ziel ist hierbei das Heimfindungsverhalten effizient zu gestalten. Im Unterschied zur Subsumptionsarchitektur sollen nun in jedem Zeitschritt die Bewegungsentscheidungen der einzelnen Navigationsstrategien zueinander gewichtet werden. Die Ermittlung der Gewichtung erfolgt hierbei über Reinforcement Learning. Die Gewichtung der Navigationsstrategie entspricht der Entscheidungsstrategie des Reinforcement Learning-Problems. Bevor nun der Lernansatz vorgestellt wird, werden die einzelnen Navigationsstrategien und die Subsumptionsarchitektur erläutert.

3.1.1 Lokale Navigationsstrategien

Für ein robustes Heimfindungsverhalten werden drei lokale Navigationsstrategien verwendet. Jede Navigationsstrategie liefert in jedem Zeitschritt einen egozentrischen Polarvektor $\boldsymbol{p}_t^x = (\rho_t, d_t)$ als Bewegungsentscheidung. Der Polarvektor kodiert die von der Strategie berechnete Richtung ρ_t und Distanz d_t zum Zielpunkt in egozentrischen Koordinaten.

3.1.1.1 Visuelles Homing

Das visuelle Homing ermöglicht dem Agenten durch den Vergleich zweier Bildaufnahmen zurück zu einem bekannten Ort zu gelangen. Hierzu wird die aktuelle Aufnahme der Umgebung ($\boldsymbol{I_t}$) mit dem gespeicherten Bild des Zielortes ($\boldsymbol{I^h}$) verglichen.
Um die räumliche Anordnung der Landmarken, die den Zielpunkt charakterisieren, zu erhalten, besitzt der Agent eine Panoramakamera (360° Sichtfeld). Durch die Panoramakamera ist die Ausrichtung des Agenten nicht entscheidend, da er stets alle sichtbaren Objekte der Umgebung wahrnehmen kann. Jeder Ort ist somit durch die Landmarkenkonfiguration definiert. Das Kamerabild wird in einem Verarbeitungsschritt auf 72 Pixel reduziert und stellt den Horizont des Panoramabildes dar. Um nun eine Bewegungsentscheidung aus den Bildaufnahmen zu extrahieren, wird der "Image Warp" Algorithmus von M. O. Franz [8] verwendet. Dabei wird, ausgehend vom aktuellen Bild, für eine bestimmte Bewegung ein erwartetes Bild ($\boldsymbol{I^c}$) kreiert. Dies erfolgt in diskreten Richtungen um den aktuellen Ort. Aus der entstandenen Menge an erwarteten Bildern wird nun jenes Bild ausgewählt, das zum gespeicherten Bild die höchste Ähnlichkeit aufweist, wie die folgende Formel beschreibt:

$$\boldsymbol{p}_t^h = (\rho_t^h, d_t^h)^\top = \arg\min_{\rho,d}(\boldsymbol{I}^h - \boldsymbol{I}^c(\rho, d))^2 \tag{3.1}$$

Die Bewegungsentscheidung wird über den Polarvektor $\boldsymbol{p}_t^h = (\rho_t^h, d_t^h)$ beschrieben, wobei ρ die Rotation und d die Distanz angibt. $\boldsymbol{I}^c(\rho, d)$ ist das Bild, das man nach der Bewegung (ρ, d) erwartet, hierbei wird angenommen, dass alle Landmarken gleich weit entfernt sind.

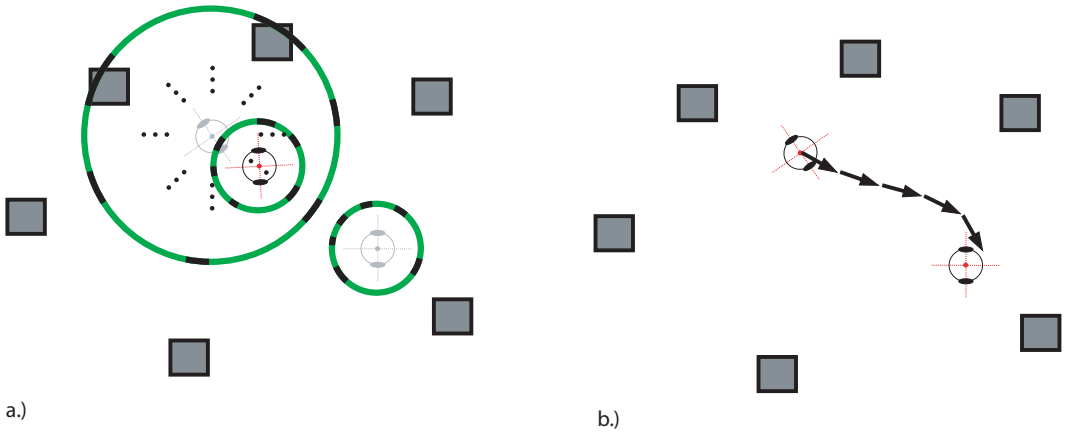

Abbildung 3.1: **Visuelles Homing:** Darstellung a.) zeigt, wie der Roboter ausgehend von seiner aktuellen Position (hell-grau dargestellter Roboter links oben) die erwarteten Bilder vorausberechnet. Die Punkte zeigen die Positionen an, für die die Bilder erzeugt werden. Eines dieser Bilder weist nun die höchste Ähnlichkeit zum Zielpunkt auf (hell-grauer Roboter unten rechts). Die Ringe um die aktuelle Position und um die Zielposition zeigen die schematische Abbildung des Panoramabildes der Umgebung. Im zweiten Bild b.) sieht man die gefahrene Strecke, die durch das Homing berechnet wurde.

In Abbildung 3.1 ist der Ablauf des visuellen Homings dargestellt. Darstellung a.) zeigt schematisch den "Image Warp" Algorithmus. In b.) sieht man die Trajektorie, die durch das visuelle Homing gefahren wurde.
Der gesamte Homing-Durchlauf setzt sich iterativ aus der Vorausberechnung der erwarteten Bilder und der folgenden Bewegungsentscheidung zusammen. Somit erhält man eine Sequenz aus Bewegungsentscheidungen, gespeichert als Polarvektoren $S^h = \{\boldsymbol{p}_0^h, \boldsymbol{p}_1^h \cdots \boldsymbol{p}_n^h\}$. Nach einem erfolgreichen Homing-Durchlauf wird angenommen, dass der Zielpunkt erreicht ist, wenn $\boldsymbol{p}_n^h \approx (\rho_n^h, 0)^\top$ gilt. Wenn allerdings der Agent nach einer festgelegten Anzahl von Iterationen das Ziel noch nicht erreicht hat, wird das Homing abgebrochen und als gescheitert angesehen.

3.1.1.2 Homing über Wegintegration

Die Wegintegration bietet dem Agenten die Möglichkeit über die Schätzung seiner Eigenbewegung zu einem Zielpunkt zurückzukehren. Der Wegintegrator ist hierbei definiert als $\boldsymbol{P}_t = (x_t, y_t, \phi_t)^\top$, wobei (x_t, y_t) die aktuell geschätzte Position, relativ zum Startpunkt, darstellt und ϕ_t die Orientierung des Agenten ist.
Bei der Wegintegration muss der Agent den Wegintegrator nach jedem Schritt aktualisieren. Hierzu berechnet man, wie der letzte Bewegungsschritt die jetzige Position im Verhältnis zum Startpunkt verändert hat. Aus dieser Schätzung kann der Heimvektor als

ein Polarvektor $\boldsymbol{p}_t^p = (\rho_t, d_t)^\top$ berechnet werden. Der Heimvektor gibt dabei an, in welcher Distanz und in welchem Winkel der Zielort bezogen auf die aktuelle Position liegt. Jede Bewegungsentscheidung wird in Teilschritte unterteilt, abhängig von der Auflösung des Schrittzählers des verwendeten Roboters. Wenn der Wegintegrator abgelaufen ist, wenn gilt $\boldsymbol{P}_0 = (0, 0, 0)$, ist der geschätzte Zielpunkt erreicht. Abbildung 3.2 zeigt schematisch die Heimfindung über Wegintegration. Die Position P_t stellt die aktuelle Position des Roboters dar, P_0 die Position des Zielortes. Der Heimvektor wird hier als Pfeil auf den Zielpunkt dargestellt. Die gestrichelte Linie ist die Trajektorie von P_t zu P_0.

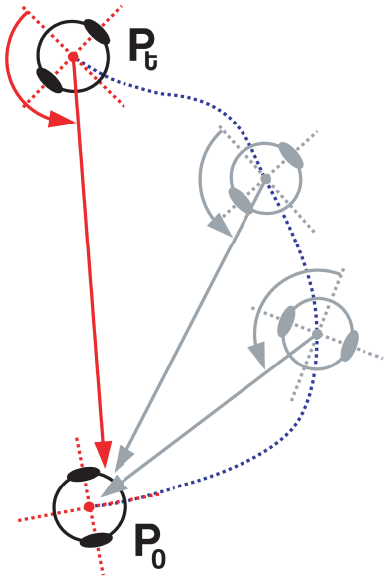

Abbildung 3.2: **Wegintegration** P_t stellt die aktuelle Position des Roboters dar und P_0 den Zielpunkt. Der Heimvektor ist als Pfeil auf den Zielpunkt angedeutet. Die gestrichelte Linie zeigt die Trajektorie zum Zielpunkt, in hell-grau sind dabei Zwischenschritte auf der Trajektorie eingezeichnet

3.1.1.3 Hindernisvermeidung

In dieser Arbeit wurde eine einfache Form der Hindernisvermeidung gewählt. Hierzu werden die Infrarotsensoren des simulierten Roboters ausgelesen. Je nach sensorischem Eingang werden vordefinierte Motorkommandos direkt an die Räder übertragen. Um Hindernisse vermeiden zu können, ist das Verhalten so aufgebaut, dass der Agent, sobald er ein Hindernis detektiert, sich von diesem Hindernis abwendet und eine kurze Strecke vom Hindernis wegfährt. Um „Dead Lock"-Situationen zu vermeiden, ist die Länge der Strecke zufällig auf einem Intervall von $(0, 5)$ gewählt. Auch dieser Mechanismus liefert als Bewegungsentscheidung einen Polarvektor $\boldsymbol{p}_t^o = (\rho_t, d_t)^\top$, wobei ρ_t die Rotation des Agenten vom Hindernis beschreibt und d_t die Länge angibt. Abbildung 3.3 zeigt den Ablauf der Hindernisvermeidung. Die Infrarotsensoren des Roboters werden durch offene Dreiecke am Roboter gekennzeichnet.

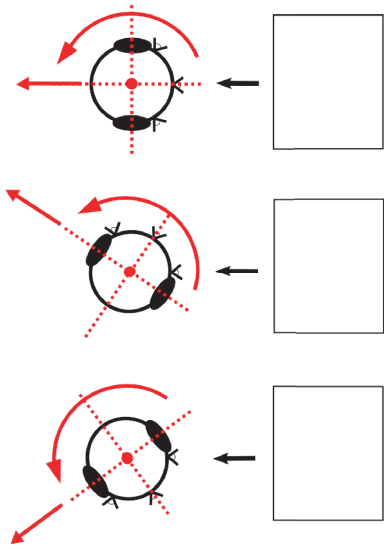

Abbildung 3.3: **Hindernisvermeidung:** Ein Hindernis wird als Rechteck dargestellt. Detektiert ein Sensor (offenes Dreieck am Roboter) ein Hindernis, dreht sich der Roboter vom Hindernis weg und fährt weg

3.1.2 Subsumptionsarchitektur

Der Ablauf des Gesamtsystems erfolgt in diskreten Zeitschritten. Jede der Navigationsstrategien liefert im Zeitschritt t einen Polarvektor $\boldsymbol{p}_t^x$. Dieser Polarvektor stellt die vorgeschlagene Bewegungsentscheidung der jeweiligen Navigationsstrategie dar. Allerdings widersprechen sich die einzelnen Strategien in ihrer Bewegungsentscheidung. Die Subsumptionsarchitektur [4] stellt eine häufig verwendete Entscheidungsstrategie zur Lösung des Integrationsproblems der Navigationsstrategien dar.
Eine mögliche Organisation der drei Navigationsstrategien sieht beispielsweise wie folgt aus: Zuerst wird überprüft, ob ein Hindernis detektiert wurde. In diesem Fall wird die Bewegungsentscheidung der Hindernisvermeidung $\boldsymbol{p}_t^o$ ausgeführt. Falls kein Hindernis den Weg zum Zielpunkt blockiert, folgt der Agent zunächst der Wegintegration mit der Bewegungsentscheidung $\boldsymbol{p}_t^p$. Wenn der Wegintegrator abgefahren ist ($\boldsymbol{p}_t^p = (0,0)$), wird mit Hilfe des visuellen Homings kontrolliert, ob das Ziel schon erreicht ist. Stimmt das Bild der jetzigen Position nicht mit dem gespeicherten Bild überein, wird die Bewegungsentscheidung des visuellen Homings $\boldsymbol{p}_t^h$ ausgeführt.
Abbildung 3.4 zeigt den beschriebenen Ablauf.

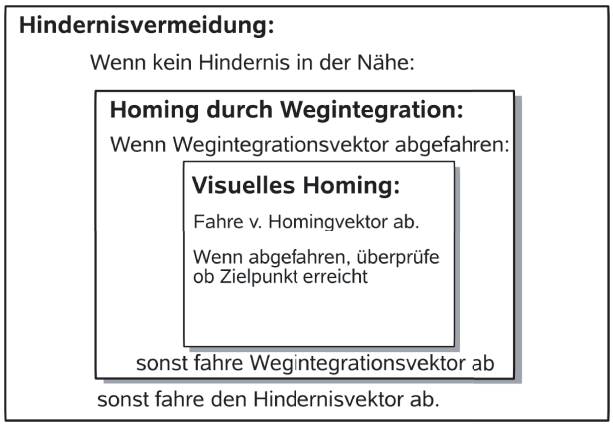

Abbildung 3.4: **Subsumptionsarchitektur** - schematisch Darstellung

3.2 Probleme der Subsumptionsarchitektur

Anhand von zwei Beispielen soll gezeigt werden, wie durch die unflexible Organisation der Subsumptionsarchitektur eine effiziente Navigation verhindert wird. Das erste Beispiel stellt die Kombination von Wegintegration und visuellem Homing dar. Wie schon im vorherigen Abschnitt ausgeführt, folgt der Roboter, solange kein Hindernis seinen Weg blockiert, seinem Wegintegrator und überprüft nach Abfahren des Heimvektors, ob das Ziel schon erreicht wurde. Wenn der Zielpunkt noch nicht erreicht ist, versucht der Roboter diesen über visuelles Homing zu erreichen.
Die Wegintegration ist jedoch sehr fehleranfällig. Ungenaue Schätzungen der Eigenbewegung addieren sich in jedem Zeitschritt auf und können dazu führen, dass der Heimvektor nicht mehr auf den Zielpunkt weist, sondern durch die akkumulierten Fehler eine falsche Position als Zielort angibt. Abbildung 3.5 zeigt schematisch den Ablauf der Kombination von Wegintegration und visuellem Homing. Hierbei stellt P_0 die vom Roboter geschätzte Position und Orientierung dar und die Position P_0^w seine wirkliche Position in Weltkoordinaten. Das Ziel des Roboters ist nun, den Zielpunkt P_n zu erreichen. Zuerst folgt der Roboter seinem Wegintegrator. Da schon der Startpunkt P_0^w fehlerhaft geschätzt war, ist auch der Heimvektor fehlerbehaftet. Aus diesem Grund befindet sich der Roboter nach Abfahrt seines Wegintegrators (blau gepunktete Spur) noch nicht an seinem eigentlichen Zielort. Von der Position P_n^w gelangt der Roboter nun über visuelles Homing in die Nähe ($P_n^{w'}$) seines eigentlichen Zielortes P_n. Die Ellipse Z kennzeichnet den Bereich, in dem die Bildähnlichkeit zwischen aktuellem Bild und dem Bild des Zielortes zu groß ist. In diesem Bereich Z kann über visuelles Homing nicht navigiert werden. Die zweite Ellipse C gibt den Fangbereich, auch Catchment Area genannt, des Zielortes P_n an. Nur in diesem Bereich ist die Navigation über visuelles Homing überhaupt möglich. Größe und Form des Fangbereichs C und der Ellipse Z hängen von der Bildqualität der Umgebung ab und sind schwer abzuschätzen [19].

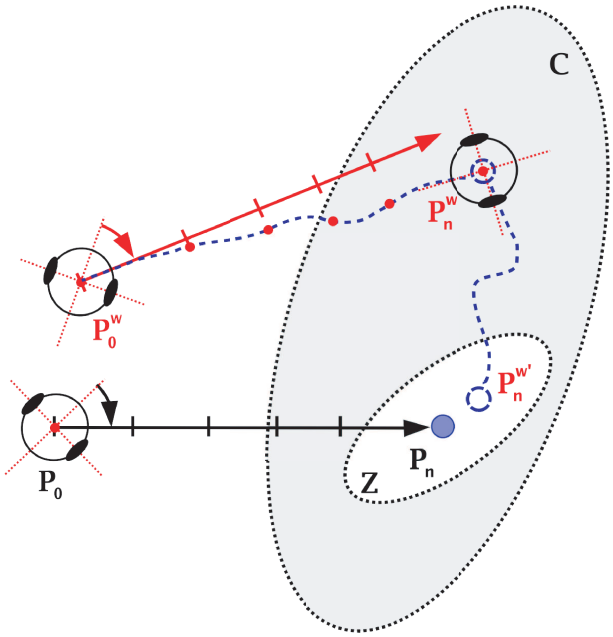

Abbildung 3.5: **Kombination von Wegintegration und visuellem Homing** [10] P_0^w ist der Ausgangspunkt des Agenten. Durch fehlerhafte Schätzung entspricht dieser Ort nicht der geschätzten Position P_0. Von Ort P_0^w gelangt der Agent über Homing durch Wegintegration zu Ort P_n^w. Die gestrichelte Linie zeigt die gefahrene Trajektorie. Ausgehend von diesem Punkt gelangt der Agent über visuelles Homing in die Nähe seines Zielortes P_n. Das visuelle Homing endet im Punkt $P_n^{w'}$. Näher kann der Agent nicht zum Zielpunkt kommen, da in diesem Bereich Z die Bildähnlichkeit schon zu hoch ist. Die Ellipse Z zeigt den Fangbereich des Zielpunktes an.

Ein erfolgreiches Heimfinden hängt vor allem davon ab, ob der Fehler, der durch die Wegintegration entstand, durch das visuelle Homing ausgeglichen werden kann. Dies ist nur dann möglich, wenn die durch die Wegintegration angefahrene Position noch im Fangbereich des visuellen Homings liegt. Ansonsten scheitert die Heimfindung.
Durch die starre Kopplung der Wegintegration und des visuellen Homings durch die Subsumptionsarchitektur kommt es zu einem "Zick-Zack"-Kurs. Abbildung 3.6 zeigt eine gefahrene Route, gestartet von Ort P_{start} zum Ort P_{ziel}. Man sieht an dieser Abbildung deutlich, dass der Wegintegrationsfehler mit der Länge des gefahrenen Weges stark zunimmt. Dies zeigt sich durch die starke Abweichung des Roboters auf dem Weg zum Zielort. Wenn der Wegintegrator abgefahren ist, erfolgt durch das visuelle Homing der Ausgleich der fehlerhaften Positionsschätzung. In Abbildung 3.6 zeigt sich dies durch einen abrupten Richtungswechsel ("Zick-Zack") des Roboters.

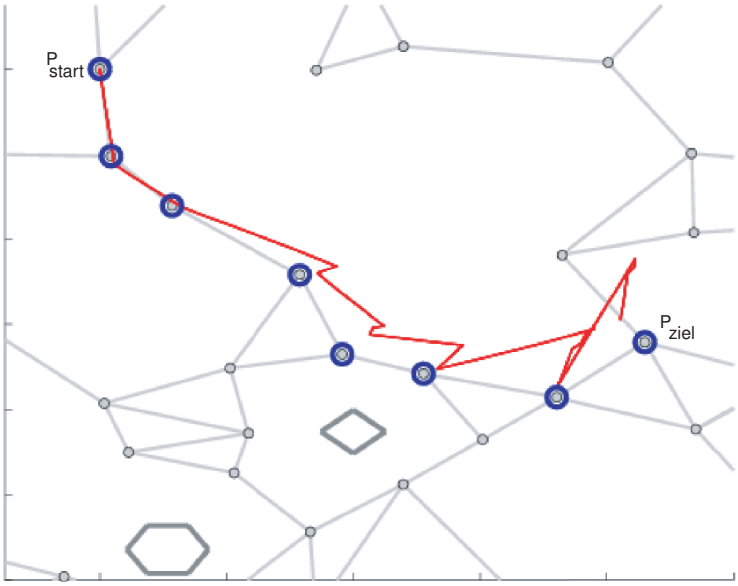

Abbildung 3.6: **Zick-Zack-Kurs aufgrund der starren Kopplung zwischen Wegintegration und vis. Homing** [10] Diese Abbildung zeigt eine gefahrene Route, gestartet von Ort P_{start} zu Ort P_{ziel}. Alle blauen Punkte zeigen Orte, die der Roboter auf seiner Route angefahren hat. Der in grau dargestellte Graph ist das bis zu diesem Zeitpunkt aufgebaute Ortsgedächtnis, wobei bekannte Orte als graue Punkte gekennzeichnet sind und über Kanten miteinander verbunden sind. Ein Ort ist nur dann mit einem anderen Ort über eine Kante verbunden, wenn von diesem Ort aus der andere Ort schon erreicht wurde. Hindernisse werden als dick gezeichnete graue Polygone abgebildet. Die rote Spur über den hervorgehobenen blauen Punkten zeigt den gefahrenen Weg des Roboters.

Sobald sich der Roboter im Fangbereich des aktuellen Zielpunktes befindet, liefern ihm sowohl die Wegintegration als auch das visuelle Homing Informationen darüber, wo sich der Zielpunkt befindet. Auf Grund dessen entstehen Abweichungen von den eigentlichen Zielpunkten.
Neu am Ansatz dieser Diplomarbeit ist, beide Navigationsstrategien gleichzeitig zu nutzen, das heißt, die beiden Bewegungsentscheidungen zueinander zu gewichten und zu einer einzigen Bewegungsentscheidung zusammenzufassen. Durch die Gewichtung der Navigationsstrategien soll der Roboter in der Lage sein, den "Zick-Zack"-Kurs zu vermeiden, nachdem er gelernt hat, wann er welche Navigationsstrategie wie gewichten muss. In diesem Beispiel wäre es von Vorteil schon früher die Information des visuellen Homings in die Bewegungsentscheidung mit aufzunehmen.
Das zweite Beispiel beschäftigt sich mit dem Fall der Hindernisvermeidung. Wenn der Roboter ein Hindernis in einer bestimmten Distanz detektiert, ist es in der Hindernisvermeidung so vorgesehen, dass er sich vom Hindernis abwendet und eine gewisse Strecke vom Hindernis wegfährt. Abbildung 3.7 zeigt die Auswirkung der Hindernisvermeidung auf den Ablauf der Navigationsaufgabe. Die Aufgabe des Roboters ist hierbei vom Start-

punkt P_{start} einer Route bis zum Zielpunkt P_{ziel} zu folgen. Auf seinem Weg stößt der Roboter auf ein Hindernis, und die Hindernisvermeidung wird aktiviert. Dies führt dazu, dass der Roboter von seiner eigentlichen Route abweicht. Nachdem der Roboter dem Hindernis ausgewichen ist, muss sich der Roboter neu lokalisieren und kann dann versuchen seine Route fortzusetzen.

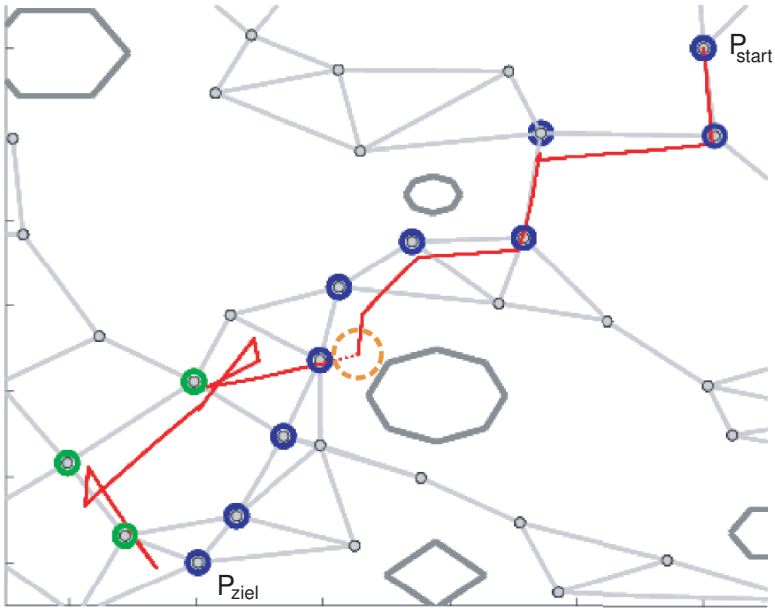

Abbildung 3.7: **Auswirkung der Hindernisvermeidung auf die Navigation** [10] Die Aufgabe ist hierbei, vom Startpunkt P_{start} einer Route bis zum Zielpunkt P_{ziel} zu folgen. Die auf dem Weg liegenden Punkte sind blau markiert. Auf seinem Weg stößt der Roboter auf ein Hindernis, in Abbildung 3.7 durch einen gestrichelten orangefarbenen Kreis gekennzeichnet, und die Hindernisvermeidung wird aktiviert. Dies führt dazu, dass der Roboter von seiner eigentlichen Route abweicht. Nachdem er dem Hindernis ausgewichen ist und eine gewisse Strecke gefahren ist, muss sich der Roboter neu lokalisieren und kann dann versuchen seine Route fortzusetzen. Die Orte in der Abbildung, die grün dargestellt sind, zeigen die Orte, die durch die Hindernisvermeidung angefahren wurden.

Durch diese unflexible Hindernisvermeidung kommt der Roboter zu weit von der eigentlichen Route ab und muss sich neu lokalisieren. Um dies zu verhindern, könnte man auch bereits während der Hindernisvermeidung in die Bewegungsentscheidung Informationen von der Wegintegration und dem visuellen Homing einfließen lassen.
Durch Gewichtung der Navigationsstrategien in jedem Zeitpunkt soll die Navigationsleistung verbessert werden und Fehler oder gefahrene Umwege durch die anderen Navigationsstrategien ausgeglichen werden.

3.3 Integration über eine gewichtete Summe

In dieser Diplomarbeit wird ein neuer Ansatz zur Integration lokaler Navigationsstrategien vorgestellt. Das Ziel ist es dabei, die einzelnen Bewegungsentscheidungen $\boldsymbol{p}_t$ der Navigationsstrategien in jedem Zeitpunkt situationsabhängig zueinander zu gewichten um so eine Bewegungsentscheidung $\boldsymbol{p}_t^m$ zu formulieren. Gleichung 3.2 beschreibt die gewichtete Summe, über die die einzelnen Bewegungsentscheidungen zu einer Bewegungsentscheidung zusammengefasst werden:

$$\boldsymbol{p}_t^m = \alpha \boldsymbol{p}_t^h + \beta \boldsymbol{p}_t^p + (1 - \alpha - \beta)\boldsymbol{p}_t^o \qquad 0 \leq \alpha - \beta \leq 1 \tag{3.2}$$

Die Linearkombination in der Gleichung 3.2 setzt sich aus den Polarvektoren der einzelnen Navigationsstrategien zusammen. Jeder Vektor stellt eine mögliche Bewegungsentscheidung dar: $\boldsymbol{p}_t^h$ = visuelles Homing, $\boldsymbol{p}_t^p$ = Wegintegration und $\boldsymbol{p}_t^p$ = Hindernisvermeidung. Je nach Gewichtung der einzelnen Vektoren ergibt sich die Bewegungsentscheidung $\boldsymbol{p}_t^m$. Durch die Beschreibung $0 \leq \alpha + \beta \leq 1$ entsteht der Raum der möglichen Entscheidungen. Es kann somit nur einer Strategie komplett gefolgt werden. Zudem ist die Hindernisvermeidung nur dann aktiv, wenn die Summe von α und β nicht 1 ist. Abbildung 3.8 zeigt schematisch den Raum der möglichen Gewichtungen.

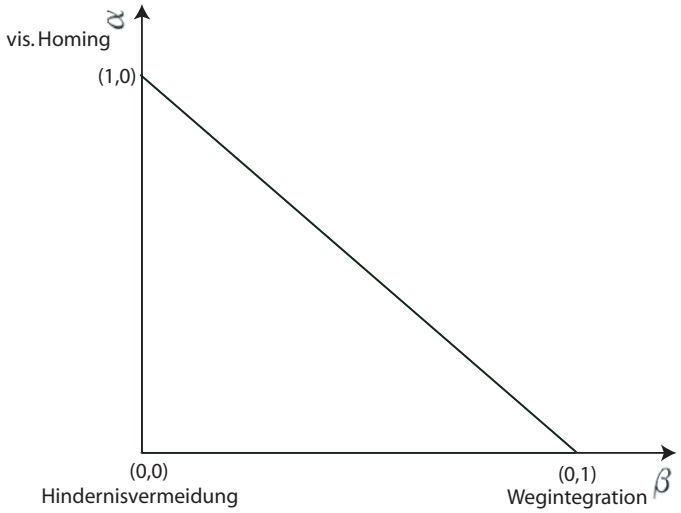

Abbildung 3.8: Raum der gewichteten Navigationsstrategien

In der Subsumptionsarchitektur sind die möglichen Entscheidungen auf den Ecken des Dreiecks repräsentiert, da hierbei immer nur der Bewegungsentscheidung einer Strategie gefolgt wird. Die Wahl der Gewichte muss situations- und zeitbedingt erfolgen, da beispielsweise der Wegintegrator mit zunehmender Strecke immer ungenauer wird. Ebenso hat die Hindernisvermeidung höchste Priorität, allerdings kann die Dringlichkeit variieren, je nach Abstand des Roboters zum Hindernis. Aus diesem Grund wurde zur Ermittlung einer Entscheidungsstrategie ein Reinforcement Learning-Ansatz gewählt, der im folgenden Kapitel vorgestellt wird. Der Vorteil bei der Anwendung von Reinforcement Learning ist, dass der Roboter durch Erfahrungen selbst die Gewichtung der Navigationsstrategien lernt.

Kapitel 4

Reinforcement Learning Problem

Die Gewichtung der Navigationsstrategien zueinander soll situations- und zeitabhängig so gesetzt werden, dass der Roboter in der Lage ist seinen Zielort sicher wiederzufinden. Die Ermittlung der Entscheidungsstrategie, d.h. die Auswahl der α und β Werte, erfolgt durch Reinforcement Learning. Der Agent soll durch seine eigenen Erfahrungen lernen, zu welchem Zeitpunkt und in welcher Situation er welche Strategie wie gewichtet. Hierzu muss die Lernaufgabe in ein Reinforcement Learning-Problem umformuliert werden. Entscheidend ist dabei (siehe 2.2.2.1) die Wahl der einnehmbaren Zustände und Aktionen. Als Zustände werden die von den lokalen Navigationsstrategien gelieferten Bewegungsentscheidungen gewählt. Die Aktionen sind durch die Problemstellung vorgegeben und werden somit durch α und β Werte dargestellt.
Die Lernaufgabe des Roboters ist definiert als Heimfindung zum Zielort. Erreicht der Agent seinen Zielort in einer vorgegebenen Zeitspanne, erhält er eine Belohnung. Schafft er es nicht in der vorgegebenen Zeit bekommt er keine Belohnung. Die Wahl der Aktionen in der Lernphase erfolgt randomisiert. Durch das gezeigte Verhalten in der Lernphase kann der Roboter Erfahrungen sammeln. Erfahrungen bedeutet hierbei den Zusammenhang zwischen Belohnung und einer Sequenz von Zustands-Aktionspaaren. Diese Erfahrungen werden durch die verwendete Q-Funktion widergespiegelt. Diese Funktion berechnet für jedes Zustands-Aktionspaar den erwarteten Reward. Der Reward ist eine reelle Zahl, die dem Roboter vorgibt, wie wahrscheinlich es ist, mit diesem Paar eine Belohnung zu erhalten (siehe 2.2.2.1). Nachdem die Lernphase abgeschlossen ist, kann der Agent durch die Q-Funktion bestimmen, in welchem Zustand er welche Aktion ausführen muss um seinen Zielpunkt wiederzufinden. Die Wahl der Aktion in einem Zustand wird über die Maxima der Q-Funktion bestimmt. Hierfür wird das lokale Maximum des Zustands unter den möglichen Aktionen gesucht. Diese Aktion stellt die erfolgversprechendste Aktion dar.
Im folgenden Kapitel soll nun das Reinforcement Learning-Problem genauer besprochen werden.

4.1 Lernansatz

Zur Aufstellung des Reinforcement Learning-Problems müssen die grundlegenden Elemente definiert werden. Die grundlegenden Elemente sind der Zustandsraum, der Aktionsraum, die Belohnung, die Rewardfunktion und die Valuefunktion.

4.1.1 Zustands-Aktionsraum

Zustände: Die Zustände werden über die Bewegungsentscheidung der lokalen Navigationsstrategien definiert. Jede Bewegungsentscheidung ist als Polarvektor $\boldsymbol{p}_t = (\rho_t, d_t)$ kodiert und beinhaltet einen Richtungsvorschlag ρ_t und eine Distanzschätzung d_t. Der Zustandsraum setzt sich somit aus der Bewegungsentscheidung des visuellen Homings ($\boldsymbol{p}_t^h$), der Bewegungsentscheidung der Wegintegration ($\boldsymbol{p}_t^p$) und der Bewegungsentscheidung der Hindernisvermeidung ($\boldsymbol{p}_t^o$) zusammen. Durch die Wahl des Zustandsraums ist es dem Roboter möglich, die „Dringlichkeit“ der einzelnen Bewegungsentscheidungen in seiner Entscheidung zu berücksichtigen. Die Stärke des Bewegungsvorschlags einer Navigationsstrategie spiegelt sich wider in der Richtung ρ_t und der Distanz d_t. Wenn $\rho_t = 0$ ist, liefert die jeweilige Strategie keine Rotationsinformation, da sie mit der eingenommenen Ausrichtung des Roboters übereinstimmt. Ähnlich verhält es sich auch mit der Distanz d_t: wenn beispielsweise der Wegintegrator die geschätzte Ausgangsposition annimmt, liefert die Bewegungsentscheidung eine Distanz $d_t = 0$. Somit ist es dem Roboter möglich zu lernen, wie er die einzelnen Navigationsstrategien situationsabhängig zu gewichten hat.
Durch die Verwendung der Monte Carlo-Methode wird zusätzlich ein diskreter Zeitschritt t als Dimension in den Zustandsraum aufgenommen, wodurch jeder Zustand in der Sequenz einzigartig wird. Dadurch wird auch das Problem umgangen, gleiche Zustände einer Sequenz entweder zu mitteln oder nur den ersten Zustand in die Berechnung des Reward einfließen zu lassen (siehe 2.2.2.1). Entscheidender dabei ist allerdings, dass durch dieses Vorgehen jede Bewegungsentscheidung einen Zeitstempel erhält, so dass der Roboter darüber hinaus lernen kann, die Gewichtung der Strategien zeitabhängig zu wählen. Somit setzt sich der Zustandsraum wie folgt zusammen:

- ρ_t^h = vorgeschlagene Richtungsentscheidung des visuellen Homings
- d_t^h = vorgeschlagene Distanz des visuellen Homings
- ρ_t^p = vorgeschlagene Richtungsentscheidung der Wegintegration
- d_t^p = vorgeschlagene Distanz der Wegintegration
- ρ_t^o = vorgeschlagene Richtungsentscheidung der Hindernisvermeidung
- d_t^o = vorgeschlagene Distanz der Hindernisvermeidung
- t = Zeitschritt

Aktionen: Die Wahl der Aktionen wird durch die Problemstellung vorgegeben. Da die α und β Werte die Aktionen darstellen, kann der Agent lernen, welche Aktion in welchem Zustand zum Erfolg führt. In der Lernphase kann der Agent in jedem Zustand den α und β Wert frei wählen, wobei gilt, dass die Summe von α und β kleiner gleich 1 sein muss. In der Verhaltensphase werden die Aktionen über die Q-Funktion bestimmt. In dieser Phase wird für den aktuellen Zustand diejenige Aktion gewählt, welche das Maximum der Q-Funktion in diesem Zustand darstellt.

4.1.2 Belohnung und Rewardfunktion

Der Roboter erhält eine Belohnung, wenn er in einer vorgegebenen Zeit seinen Zielpunkt wieder erreicht. Die vorgegebene Zeitspanne für die Heimfindungsaufgabe ist dabei der Entfernung des Agenten zu seinem Zielort angepasst. Die genauen Parameter werden in Kapitel 6 ausführlich besprochen.
Bei der Belohnung handelt es sich um eine reelle Zahl. Zur Berechnung der Rewardfunktion erhält der Agent in jedem Zeitschritt einen Returnwert r_t, der die aktuelle Belohnung der jeweiligen Aktion darstellt. Diese beträgt im Falle des Erreichens des Zielpunktes $1,0$. Falls der Zielpunkt nicht erreicht ist, aber die Zeitgrenze noch nicht überschritten wurde, wird der Agent mit $-0,015$ bestraft.

$$r_t = \begin{cases} 1,0 & \text{,wenn Ziel erreicht} \\ -0,015 & \text{,sonst} \end{cases} \tag{4.1}$$

Durch diese leichte Bestrafung wird ein Zeitdruck erzeugt. Zudem schafft dieser Bestrafungsterm eine bessere Bedingung für die Approximation der Q-Funktion durch das RBF. Auf die Bestrafung und ihre Konsequenz bezogen auf das Approximationsverhalten wird in Kapitel 6 noch ausführlich eingegangen.
Mit Hilfe der Returnwerte r_t wird am Ende eines Durchlaufs der Reward $R(\boldsymbol{s}, \boldsymbol{a})$ für jedes Zustands-Aktionspaar der gegebenen Sequenz gebildet. Die Rewardfunktion setzt sich wie folgt zusammen:

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{4.2}$$

Die berechneten Rewardwerte der Zustands-Aktionspaare werden später für die Approximation der Q-Funktion verwendet.

4.1.3 Q-Funktion

Als Valuefunktion wurde in diesem Lernansatz die Q-Funktion gewählt. Die Gleichung 4.3 zeigt die mathematische Formulierung der Q-Funktion:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \tag{4.3}$$

Die Q-Funktion stellt ein Zustands-Aktionspaar in Verbindung mit dem erwarteten Reward dar (siehe 2.2.2.1). Die Entscheidungsstrategie π, unter der die Q-Funktion $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ betrachtet wird, ist in der Lernphase eine Zufallsstrategie. In der Verhaltensphase wird dann π die Q-Funktion wie folgt bestimmt:

$$\pi(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_{a} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{4.4}$$

4.2 Anwendung der Monte Carlo Methode

Zur Lösung des Reinforcement Learning-Problems wird die Monte Carlo-Methode verwendet. Der Vorteil dieser Methode ist, dass für ihre Anwendung nicht verlangt wird, dass die Dynamik des Lernproblems zwangsläufig einem Markov-Entscheidungsprozess entspricht. Dieser Faktor spricht für die Verwendung der Methode, da die Dynamik hinter dem Lernproblem nicht analysiert wurde.
Für die Monte Carlo Methode benötigt man Sequenzen von erhaltenen Zustands-Aktionspaaren mit ihren Rewards aus der Lernphase. Aus diesen Erfahrungen/Informationen kann in der Verhaltensphase mit Hilfe der Q-Funktion diejenige Strategie berechnet werden, welche den höchsten Erfolg unter den gegebenen Daten verspricht (siehe 2.2.2.1).
Bei dieser Anwendung der Monte Carlo Methode werden die Lernphase und die Verhaltensphase, das Zeigen des gelernten Verhaltens, aufgespalten, worauf im nächsten Abschnitt eingegangenen wird.

4.2.1 Lernphase

Die Lernphase setzt sich aus einer vordefinierten Anzahl von Durchläufen der Lernaufgabe zusammen. In einem Durchlauf hat der Roboter die Aufgabe zu seinem Zielpunkt zurückzukehren. Am Anfang der Lernphase befindet sich der Roboter an einem zufällig ausgewählten Punkt in der Umgebung. Jeder Durchgang startet mit der Initialisierung. Als erstes speichert der Agent ein Bild seiner aktuellen Umgebung und setzt seinen Wegintegrator zurück auf Null. Der Ort stellt nun den Zielort P_0 dar. Als nächstes wird der Agent auf eine zufällig gewählte Position, die sich in einem bestimmtem Abstand und einer bestimmten Richtung vom Startpunkt befindet, gefahren. Ausgehend von der neuen Position P_n hat der Agent nun die Aufgabe in einer vorgegebenen Zeit den Zielort P_0 wieder zu finden.
Während der Heimfindungsphase sammelt der Roboter Erfahrungen, welche Wahl der Gewichte zum Erfolg führt. Hierzu werden alle Zustands-Aktionspaare und ihr zugehöriger Return gespeichert. Die Aktionen werden zufällig gewählt, wobei die Bedingung gilt, dass $0 \leq \alpha + \beta \leq 1$. Die Heimfindungsphase ist dann beendet, wenn der Agent den Zielpunkt wiedergefunden hat oder die Zeit abgelaufen ist. Nach der Heimfindungsphase folgt die Auswertung der Sequenz von Zustands-Aktionspaaren indem die Rewards der einzelnen Paare nach der Gleichung 4.2 berechnet werden.

Approximation der Q-Funktion Die Approximation der Q-Funktion ist notwendig, da sowohl der Aktionsraum als auch der Zustandsraum kontinuierlich sind. Für die Approximation wird ein Radiale-Basisfunktionen-Netzwerk (RBF-Netzwerk) verwendet. Im nachfolgenden Kapitel werden der Aufbau und die Implementierung des RBF-Netzwerks ausführlich besprochen und die verwendeten Algorithmen vorgestellt. Durch die Approximation der Q-Funktion ist es dem Agenten in der Verhaltensphase möglich, auch in noch nicht besuchten Zuständen den erwarteten Reward durch die Q-Funktion abzuschätzen. Dazu wird das RBF-Netz auf dem Zustands-Aktionsraum definiert und die Basisfunktionen so im Raum verteilt, dass diese äquidistant den Raum abtasten. Die genaue Beschreibung der Parameterwahl und Definition des RBF-Netzes erfolgt in Kapitel 6.

4.2.2 Verhaltensphase

In der Verhaltensphase wendet der Agent zur Lösung der Navigationsaufgabe sein erlangtes Wissen an. Um das gelernte Verhalten zu testen, wird dem Agenten die gleiche Aufgabe gestellt wie in der Lernphase. Hierzu erfolgt die gleiche Initialisierung wie in der Lernphase: der Agent wird an einen zufälligen Ort gesetzt, macht ein Bild und setzt den Wegintegrator zurück. Von diesem Ort wird er auf ein andere Position gefahren, von der aus die Heimfindungsaufgabe startet.
In der Heimfindungsphase wird in jedem Zustand die Aktion ausgewählt, die einem Maximum der Q-Funktion entspricht. Das Maximum der Q-Funktion stellt die Aktion dar, die den höchsten erwarteten Reward verspricht. Somit lässt sich die Strategie π der Verhaltensphase wie folgt beschreiben:

$$\pi(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_{a} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{4.5}$$

Die Zustände können lediglich beobachtet werden, da sie von der Umgebung und der Auswertung der einzelnen Navigationsstrategien festgelegt werden. Die Aktionen hingegen sind in jedem Zustand wählbar.
Die Maximumssuche wird in Kapitel 5 vorgestellt. Bei der Maximumssuche wird ausgenutzt, dass der Zustand nur beobachtet werden kann und damit eine Maximumssuche nur bezogen auf die α und β erfolgen muß.

Kapitel 5

Radiale-Basisfunktionen-Netz (RBF-Netz)

Mit Hilfe eines Radialen-Basisfunktionen-Netzwerkes soll die Q-Funktion approximiert werden. Wie in den Grundlagen (siehe 2.2.2.2) schon ausgeführt, handelt es sich dabei um eine Linearkombination aus radial-symmetrischen Basisfunktionen. Diese Form von Funktionsapproximator eignet sich besonders gut für die Approximation höher dimensionaler Funktionen [16]. Im folgenden Abschnitt wird der allgemeine Grundaufbau und die Implementierung des RBF-Netzes ausgeführt. Die genaue Anpassung und die Parameterwahl für die Approximation der Q-Funktion wird in Kapitel 6 beschrieben. Am Ende dieses Kapitels werden mit Hilfe ein- und zweidimensionaler Beispiele die für das Reinforcement Learning maßgeblichen Eigenschaften des RBF-Netzes beschrieben.

5.1 Grundaufbau

RBF-Netzwerke werden dazu verwendet das allgemeine Interpolationsproblem zu lösen. In schwächerer Form können RBF-Netzwerke auch zur Funktionsapproximation genutzt werden. Diese geschieht über eine Linearkombination von radialen Basisfunktionen, die mit Hilfe von Koeffizienten an die Funktion angepasst werden. Die folgende Gleichung beschreibt diesen Grundaufbau:

$$f(\boldsymbol{x}) = \sum_{i=0}^{N} c_i \phi_i(\boldsymbol{x}) \tag{5.1}$$

Als Basisfunktionen werden hier normierte Gauß-Kernels verwendet:

$$\phi(\boldsymbol{x}) = \frac{1}{\sigma\sqrt{2\,\pi}}\, e^{\frac{||\boldsymbol{\mu}_i - \boldsymbol{x}||}{2\sigma^2}} \tag{5.2}$$

Die Zentren der Basisfunktionen sind dabei so ausgewählt, dass sie den Eingangsraum in äquidistanten Abständen abtasten. Der Abstand zwischen zwei Zentren ist durch die Wahl der Standardabweichung (σ) der Gaußkurve vorgegeben. Der Abstand ist hier auf

2σ festgelegt. Nach T. Poggio & F. Girosi [16] lässt sich die Approximationsgüte verbessern, wenn sich die Basisfunktionen überlappen. Die Wahl der Standardabweichung σ bestimmt somit nicht nur die Breite der einzelnen Gaußkurven, sondern auch die Anzahl der Basisfunktionen, die den Raum abtasten.

5.2 Bestimmung der Koeffizienten

Die Koeffizientenbestimmung stellt den Lernschritt bei diesem Verfahren dar. Man könnte hierbei nach Aufstellung des Gleichungssystems (siehe 2.2.2.2) die Koeffizienten über Matrixinvertierung berechnen. Allerdings ist die Matrixinvertierung eine sehr rechenintensive Operation (N^3) und nur bei bis zu ≈ 3000 Basisfunktionen realisierbar [16]. Da im vorliegenden Fall eine siebendimensionale Funktion approximiert werden soll und die Anzahl der Basisfunktionen exponentiell mit der Anzahl der Dimensionen ansteigt, ist die Anwendung einer Matrixinvertierung nicht möglich. Deshalb wird hier zur Bestimmung der Koeffizienten ein Relaxationsverfahren angewendet. Hierbei handelt es sich um ein Verfahren iterativ die Koeffizienten zu bestimmen. Die folgende Formel beschreibt das Optimierungsproblem als die Minimierung der Fehlerquadrate:

$$g(\boldsymbol{x}) = \sum_{j}^{n} \left[y_j - \sum_{i}^{N} c_i \phi_i(x_j) \right]^2 \tag{5.3}$$

Es stehen zur Lösung n Datenpunkte zur Verfügung um die N Koeffizienten der N Basisfunktionen zu bestimmen. Für die Bestimmung der Koeffizienten wird nun von dieser Gleichung die erste Ableitung gebildet:

$$\frac{\partial g(\boldsymbol{x})}{\partial c_k} = -2 \sum_{j}^{n} \left[y_j - \sum_{i}^{N} c_i \phi_i(x_j) \right] \phi_k(x_j) \tag{5.4}$$

Diese Gleichung wird gleich Null gesetzt, der Faktor -2 ist dabei zu vernachlässigen

$$0 = \sum_{j}^{n} y_j \phi_k(x_j) - \sum_{j}^{n} \sum_{i}^{N} c_i \phi_i(x_j) \phi_k(x_j) \tag{5.5}$$

Nach der Auflösung nach einem Koeffizienten c_k, kann durch eine Aufspaltung der einzelnen Summen die Lösung so formuliert werden, dass sie iterativ berechnet werden kann.

$$c_k = \frac{\overbrace{\sum_{j}^{n} y_j \phi_k(x_j)}^{A} - \sum_{i \neq k}^{N} c_i \overbrace{\sum_{j}^{n} \phi_i(x_j) \phi_k(x_j)}^{B}}{\underbrace{\sum_{j}^{n} \phi_k^2(x_j)}_{C}} \tag{5.6}$$

Die Summen A, B und C sind dabei nicht abhängig von den Koeffizienten, sondern lediglich vom aktuellen Messpunkt x_j. Aus diesem Grunde können die einzelnen Summen wie folgt iterativ berechnet werden, wobei $(n+1)$ das Hinzufügen einer neuen Messung bedeutet:

$$A(n+1) = \left[\sum_{j}^{n} y_i \phi_k(x_j)\right] + y_{n+1}\phi_k(x_{n+1}) \tag{5.7}$$

$$B(n+1) = \left[\sum_{j}^{n}\sum_{i\neq k}^{N} \phi_i(x_j)\phi_k(x_j)\right] + \phi_i(x_{n+1})\phi_k(x_{n+1}) \tag{5.8}$$

$$C(n+1) = \left[\sum_{j}^{n} \phi_k^2(x_j)\right] + \phi_k^2(x_{n+1}) \tag{5.9}$$

Relaxation zur Bestimmung der Koeffizienten Die Relaxation bestimmt nach jeder Eingabe eines neuen Datenpunktes die Koeffizienten des RBF-Netzes neu. Hierbei wird ausgenutzt, dass durch die Verwendung von „lokalisierten“ Basisfunktionen auch die Neubestimmung der Koeffizienten lokal möglich ist. Das bedeutet, dass ein neuer Datenpunkt sich am stärksten auf die Veränderung des Koeffizienten der nächstliegenden Basisfunktion auswirkt. Die Nachbarbasisfunktionen der nächstliegenden Basisfunktion sind von der Veränderung des Koeffizienten ebenfalls betroffen, da die Gaußkurven überlappen. Weit entfernte Basisfunktionen, d.h. weiter als 4σ, werden durch diese Veränderung nur wenig betroffen, da die Basisfunktionen außerhalb von $4\sigma \approx 0$ sind.
Ein Relaxationszyklus besteht aus folgenden Schritten: Zunächst wird die nächstliegende Basisfunktion zum neuen Datenpunkt bestimmt. Für diese Basisfunktion wird der neue Koeffizient nach Gleichung 5.6 bestimmt. Wenn die Änderung zwischen dem vorherigen und dem neu berechneten Koeffizienten kleiner als eine vordefinierte Grenze ist, ist die Relaxation beendet. Falls die Änderung größer als diese Grenze ist, werden alle Basisfunktionen gesucht, die im Bereich von 4σ positioniert sind. Diese Basisfunktionen werden in eine Liste geschrieben, welche auch die gerade geänderte Basisfunktion mit einschließt. Die Relaxation läuft nun so lange, bis sich keine Basisfunktion mehr in dieser Liste befindet. In jedem Relaxationsschritt wird der Koeffizient der ersten Basisfunktion in der Liste neu berechnet und überprüft, ob sich dieser maßgeblich verändert hat. Wenn das nicht der Fall ist, wird diese Basisfunktion aus der Liste gelöscht. Falls die Veränderung größer ist als ein vordefiniertes Maß, werden auch ihre Nachbarbasisfunktionen (4σ-Bereich) in die Liste aufgenommen.

Der folgende Pseudocode zeigt die Relaxation schematisch, wobei „Basisfunktion“ mit Bfct abgekürzt wird.

Abb. 5.3: Pseudocode der Relaxation zur Bestimmung der Koeffizienten

```
Init:
        Berechnet für nächstliegende Bfct Koeffizienten
        if(|| alt_Koeffizient - neu_Koeffizient || < Vordef. Grenze)
        {
                Relaxation ist beendet
        }
        else
        {
                Suche alle Bfct im Bereich 4 sigma
                Speichere Bfct_relax in Liste Relax_Liste
        }

Relaxation:
        Init

        while(Relax_Liste nicht leer)
        {
                Bestimme Koeffizient von Relax_Liste[0]
                if(|| alt_Koeffizient - neu_Koeffizient || < Vordef. Grenze)
                {
                        lösche Element Relax_Liste[0] aus der Relax_Liste
                }
                else
                {
                        Suche alle Bfct im Bereich 4 sigma von Relax_Liste[0]
                        Speichere gefundene Bfct_relax in Liste Relax_Liste
                }
        }
```

5.3 Bestimmung der Maxima

Die Maximumssuche ist essentiell, wenn das RBF-Netz für Reinforcement Learning genutzt wird, da die Maxima der Q-Funktion das gewünschte Verhalten definieren. Aus diesem Grund wurde bei der Implementierung der RBF besonderes Gewicht auf die Maximumssuche gelegt. Eine Maximumssuche bei einer mehrdimensionalen Funktion stellt eine nicht triviale Aufgabe dar. Zur Bestimmung der Maxima wird auch hier ein iteratives Verfahren verwendet, das parallel zur Approximation die Maxima ermittelt. Hierzu muss vom RBF-Netz:

$$f(\boldsymbol{x}) = \sum_{i}^{N} c_i \phi_i(\boldsymbol{x}) \tag{5.10}$$

die erste Ableitung nach $\boldsymbol{x}$ gebildet werden:

$$\frac{\partial f(\boldsymbol{x})}{\partial x_j} = -\sum_{i}^{N} c_i \phi_i(||\boldsymbol{\mu_i} - \boldsymbol{x}||) \cdot \frac{(x_j - \mu_{ij})}{\sigma^2} \tag{5.11}$$

und zur Bestimmung der Maxima gleich Null gesetzt werden.

Algorithmus zur Maximumssuche Die Maximumssuche erfolgt nach Eingabe eines neuen Datenpunktes und der Relaxation der Koeffizienten. Ausgangspunkt für die Suche der neu entstandenen Maxima ist jene Basisfunktion, welche am nächsten zum neu hinzugefügten Datenpunkt liegt. Auch bei diesem Verfahren wird die Lokalität der Gaußkurven ausgenutzt. Hierbei sollte man bedenken, dass ein neues Maximum nur dann entstehen kann, wenn der Koeffizient maßgeblich verändert wurde. Diese Eigenschaft wird bei der Suche der Maxima genutzt um den Raum der Suche einzuschränken.
Zur Initialisierung wird dem Suchalgorithmus jene Basisfunktion übergeben, welche die nächstliegende Basisfunktion zum neu eingegebenen Datenpunkt darstellt. Ausgehend von dieser Basisfunktion werden nun Startrichtungen für die Maximumssuche bestimmt. Die Startrichtungen zeigen in die Richtung der benachbarten Basisfunktionen. Die eigentliche Maximumssuche erfolgt über ein konjugiertes Gradientenverfahren (aus der C++ Bibliothek der Numerical Recipes), dem diese Startrichtungen übergeben werden. Das konjugierte Gradientenverfahren benötigt hierzu die Richtungsableitung (siehe **??**) und den Funktionswert an einer bestimmten Stelle. Anschließend wird überprüft, ob das berechnete Maximum bereits gefunden wurde. Dies ist notwendig, da durch Hinzunahme eines neuen Datenpunktes sowohl neue Maxima auftreten als auch alte Maxima verschwinden können. Die Kontrolle der Maxima erfolgt in zwei Schritten. Zuerst werden die bereits gefundenen Maxima daraufhin überprüft, ob in diesem berechneten Bereich - ungefähr 2σ um die aktuelle Basisfunktion - ein Maximum gefunden wurde. Wenn das der Fall ist, muss überprüft werden, ob dieses alte Maximum immer noch vorhanden ist. Nach dieser Kontrolle werden alle neu gefundenen Maxima auf Ähnlichkeit zu den alten Maxima untersucht. Wenn ein neu gefundenes Maximum in einem vordefinierten Abstand zu einem bereits gefundenen Maximum liegt, wird das alte Maximum gelöscht.
Nachdem alle Datenpunkte eingegeben, relaxiert und die Maxima bestimmt wurden, liegen alle Informationen über die approximierte Q-Funktion vor, die man für das Reinforcement Learning benötigt.

Der folgende Pseudocode zeigt nochmals die Bestimmung der Maxima. „kGradient“ steht im Pseudocode für konjugiertes Gradientenverfahren.

Abb. 5.3: Pseudocode der Maximumssuche

```
Berechnet für nächstliegende Bfct Startposition für kGradient

Stratposition
{
        Berechne Startposition für jede benachbarte Basisfunktion
}

Starte mit diesem Positionen das k. Gradient

überprüfen Alt_Maxima:
{
        if(Alt_Maxima im akt. Bereich liegt)
        {
                überprüfe ob es noch vorhanden ist
                if(nicht)
                {
                        lösche Atl_Maximum
                }
        }
}

überprüfe Neu_Maxima
{
        if(Neu_Maximum ähnlich zu Alt_maximum)
        {
                lösche Alt_Maximum
        }
}
```

5.4 Beispielapproximationen

Zum Testen der Koeffizientenbestimmung und der Maximumssuche wurden eine eindimensionale Sinusfunktion und eine zweidimensionale Sinus-Kosinusfunktion verwendet. Für die Approximation des eindimensionalen Sinus wurde ein Datensatz mit 100 Datenpunkten erzeugt. Der vorgegebene Bereich zur Approximation liegt zwischen $[-12,5 : 12,5]$. Die eingegebenen Datenpunkte wurden aus einem Bereich zwischen $[13,0 : 13,0]$ genommen und ohne Rauschen der Approximation zur Verfügung gestellt. Die Standardabweichung σ der Gaußverteilung wurde auf $0,5$ gesetzt.

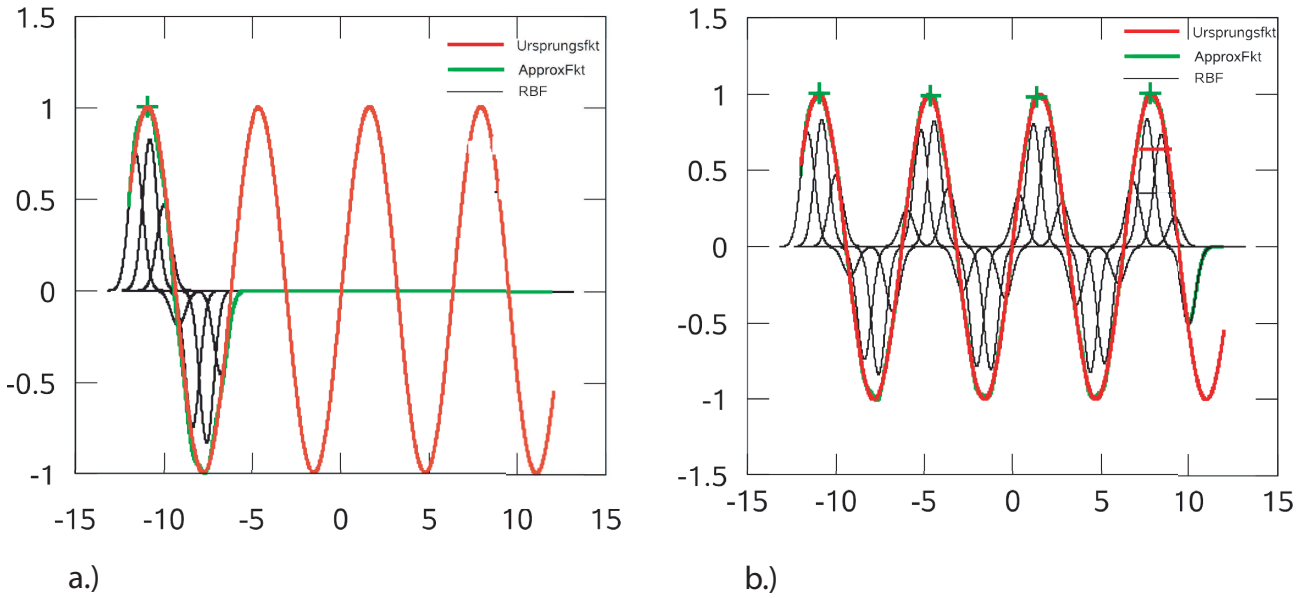

Abbildung 5.1: **Approximation einer Sinusfunktion** In a.) wird die Approximation der Sinusfunktion nach 20 Datenpunkten gezeigt. In b.) sieht man die abgeschlossene Approximation nach 100 Datenpunkten.

Abbildung 5.1 zeigt die Approximation nach der Eingabe von 20 Datenpunkten(a.) und nach der Eingabe von allen 100 Datenpunkten (b.). Die grünen Kreuze zeigen die gefundenen Maxima. An diesem Beispiel kann gezeigt werden, dass das RBF-Netz in der Lage ist einen eindimensionalen Sinus ohne Fehler zu approximieren und alle Maxima zu ermitteln.
Zum Test im mehrdimensionalen Raum wurde die Approximation einer zweidimensionalen Sinus-Kosinusfunktion vorgenommen:

$$f(\boldsymbol{x}) = (\cos(0,5 \cdot x_1))(\sin(0,5 \cdot x_2)) \tag{5.12}$$

Die zweidimensionale Funktion wird auf einem Bereich von $[-5,0 : 5,0]$ approximiert. In diesem Bereich wurden 100 Datenpunkte erzeugt. Das RBF wurde auf einem Bereich von $[-5,0 : 5,0]$ definiert und die Standardabweichung σ auf $0,5$ gesetzt.

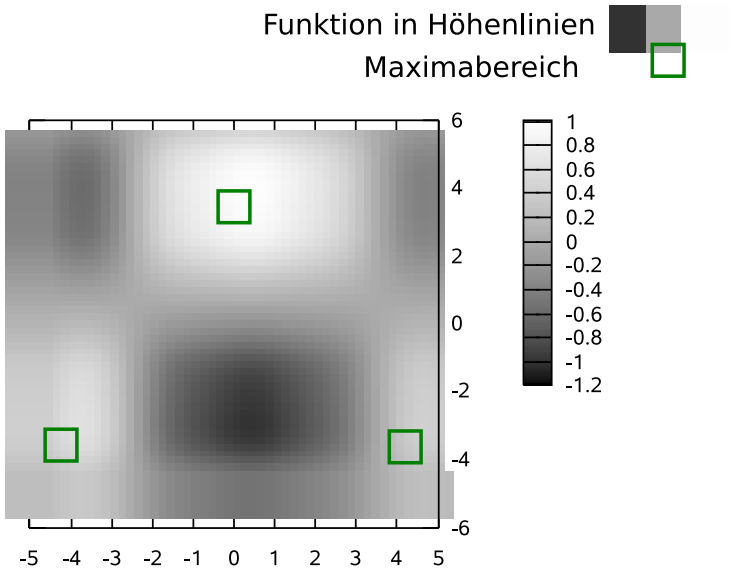

Abbildung 5.2: **Approximation einer zweidimensionalen Sinus-Kosinusfunktion** Hochdarstellung der Sinus-Kosinusfunktion. Die grünen Quadrate zeigen die berechneten Maxima der Funktion.

Abbildung 5.2 zeigt die Höhendarstellung der approximierten Funktion. Die Bereiche, die mit einem grünen Rahmen gekennzeichnet sind, stellen die gefundenen Maxima dar.
Wie diese Beispielapproximationen zeigen, kann das aufgestellte System diese Funktion approximieren und alle Maxima bestimmen.

5.5 Auswahl der Messpunkte

Für die Approximationsgüte ist die Auswahl der Messpunkte von großer Bedeutung. Aus diesem Grund wurde eine Analyse der Auswahl der Messpunkte durchgeführt. Nach T. Poggio & F. Girosi sollten die Messpunkte auf den Zentren der Basisfunktionen liegen [16].
Um diesen Sachverhalt experimentell zu belegen, wurde die eindimensionale Funktion $f(x)_t$ approximiert. Diese Funktion wurde von M.J.L. Orr [15] vorgeschlagen, da sie unterschiedliche Qualitäten aufweist, wie z.B. einen konstanten Bereich, ein globales Maximum und einen Sattelpunkt. Es handelt sich dabei um ein "Hermite"-Polynom:

$$\begin{aligned} f(x)_t &= 1 + \gamma \cdot (1 - x + 2x^2)e^{-\nu \cdot x^2} \\ \gamma &= 2,5 \\ \nu &= 0,5 \end{aligned} \tag{5.13}$$

Dieses Polynom wurde in einem Bereich von $[-8,0 : 8,0]$ mit Hilfe von 100 Datenpunkten approximiert. Hierzu wurde auf die Datenpunkte ein normalverteiltes Rauschen mit einer Standardabweichung von 0,25 addiert. Drei unterschiedliche Fälle wurden untersucht. Bei der ersten Approximation wurden die Messpunkte zufällig im vorgesehenen Bereich

ausgewählt und dem RBF-Netz zur Verfügung gestellt. Im nächsten Fall wurden nur die Datenpunkte, die genau zwischen den Zentren des RBF-Netzes liegen, gesammelt, bei der letzten Untersuchung lagen die Datenpunkte immer genau auf den Zentren der Basisfunktionen. Abbildung 5.3 zeigt die Approximation mit zufällig gewählten Messpunkten. In den folgenden Abbildungen 5.3, 5.4, 5.5 werden die einzelnen Basisfunktionen (dünnere schwarze Gaußkurven), die durch das RBF approximierte Funktion (grüne Funktion), die Ursprungsfunktion $f(x)_x$ (rote Funktion) und die Datenpunkte (blau) dargestellt.

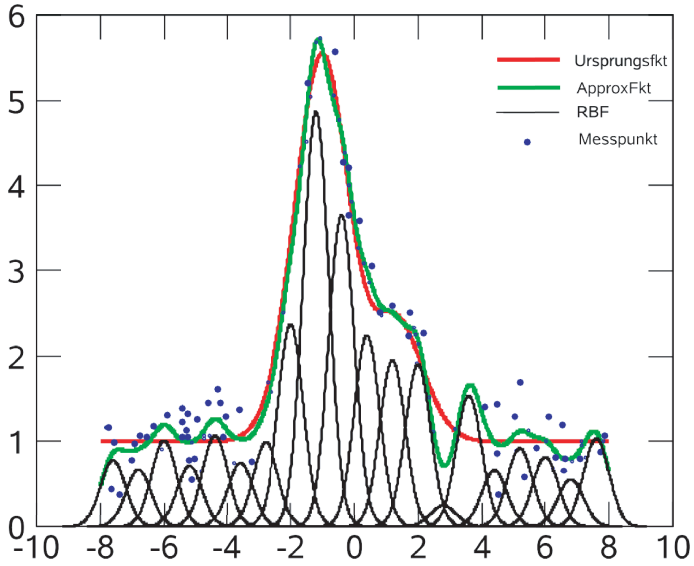

Abbildung 5.3: **Approximation mit zufällig gewählten Messpunkten**

Man sieht in Abbildung 5.3, dass die Approximationsgüte vom Rauschen der Daten und der Verteilung der Messpunkte abhängt. In den Bereichen, wo viele Messpunkte auftreten, ist das RBF in der Lage, die ursprüngliche Funktion zu approximieren. Wenn allerdings nur wenige verrauschte Datenpunkte in einem Bereich liegen, besitzt das RBF-Netz zu wenig Information über die Ursprungsfunktion.

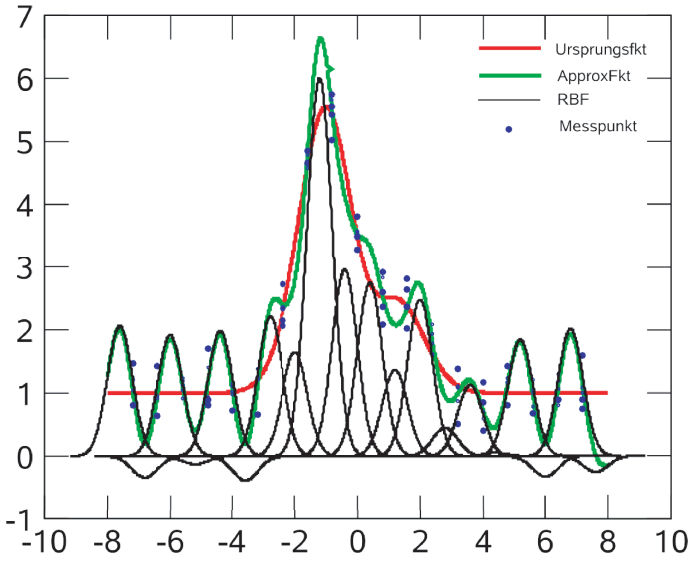

Abbildung 5.4: **Approximation mit Messpunkten, die zwischen den Zentren liegen**

Abbildung 5.4 zeigt die Approximation der Funktion mit Messpunkten, die genau zwischen den Zentren der Basisfunktionen liegen. Es fällt hierbei auf, dass die Approximation mit der gleichen Anzahl an Messpunkten wesentlich schlechter ist, als bei der Approximation mit den zufällig gewählten Messpunkten. Ein Grund für diese schlechte Approximationsgüte ist, dass eine Annäherung der zu approximierenden Funktion nur über die Wahl der Koeffizienten und somit die Höhe der einzelnen Gaußkurven geschehen kann. Die Wahl eines Koeffizienten einer Basisfunktion ist abhängig von den Messpunkten, die in einem Bereich von 4σ um das Zentrum dieser Basisfunktion liegen. Wenn nun mehrere Datenpunkte mit unterschiedlichen Werten im Abstand von σ, bezogen auf das Zentrum der Basisfunktion, zur Bestimmung des Koeffizienten genutzt werden, geht dies als Rauschen in die Koeffizientenbestimmung ein. Dies führt dazu, dass die Koeffizienten sich schlecht an die gegebenen Datenpunkte anpassen können.
Den letzten untersuchten Fall stellt Abbildung 5.5 dar. Für diese Approximation wurden ausschließlich Punkte verwendet, die genau auf den Zentren der Basisfunktionen lagen. Bei diesem Vorgehen ist die Approximationsgüte deutlich besser als bei den beiden vorherigen Approximationen. Im Gegensatz zum vorhergehenden Fall liefert die Lage der Datenpunkte den höchsten Informationsgehalt zur Bestimmung der Koeffizienten, da die Messpunkte die Wahl der Koeffizienten am genauesten beschreiben.

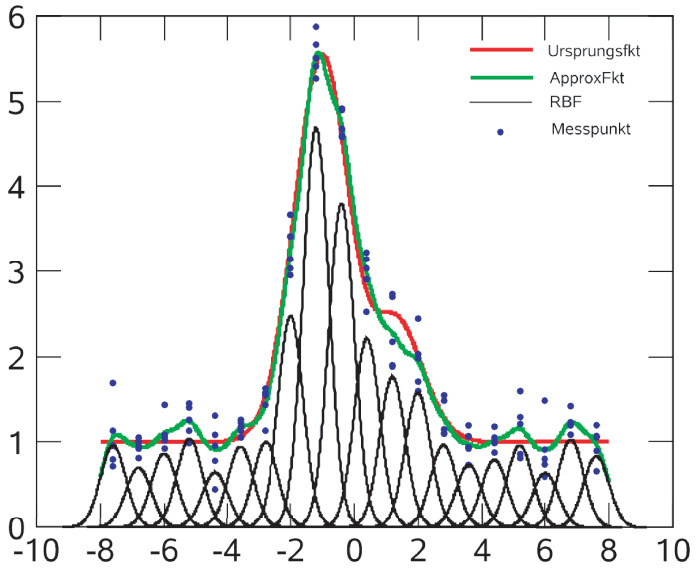

Abbildung 5.5: **Approximation mit Messpunkten, die auf den Zentren liegen**

Aus diesen Ergebnissen kann man ableiten, dass die Messpunkte für eine Approximation durch ein RBF-Netz im besten Fall auf den Zentren der Basisfunktionen liegen sollten, um zum einen eine gute Approximation zu erhalten und zum anderen die besten Voraussetzungen für die Relaxation der Koeffizientenbestimmung zu schaffen. Bei der Verwendung des RBF-Netzes beim Reinforcement Learning Problem ist eine effiziente Koeffizientenbestimmung notwendig. Allerdings sind die gegebenen Messpunkte nicht frei wählbar, da sich ein Messpunkt aus den Zuständen und den Aktionen des Agenten zusammensetzt. Die Zustände $(\boldsymbol{p}_t^h, \boldsymbol{p}_t^p, \boldsymbol{p}_t^o)$ sind durch die Bewegungsentscheidungen der einzelnen Navigationsstrategien vorgegeben. Die Aktionen sind in der Lernphase, d.h. in der Phase, in der die Q-Funktion approximiert wird, allerdings frei wählbar. Um die Approximationsgüte zu verbessern, wurde festgelegt, dass nur Aktionen gewählt werden dürfen, die auf den Zentren der Basisfunktion liegen.
In einer weiteren Experimentreihe wurde analysiert, wie sich Rauschen in Y-Richtung und X-Richtung der Datenpunkte, die auf den Zentren von Basisfunktionen liegen, auf die Approximationsgüte und die Relaxationsgeschwindigkeit auswirkt. Dazu wurde eine Funktion approximiert, die als Datenpunkte stets die Werte 1 oder 0 erhalten hat. Durch diese Wahl der Messpunkte soll ermittelt werden, ob das RBF-Netz in der Lage ist über den gegebenen Datenpunkten zu mitteln. Die Messpunkte werden hierbei zufällig auf 0 oder 1 gesetzt. Die Fähigkeit der Mittelung ist entscheidend, da beim Reinforcement Learning-Ansatz die Monte Carlo-Methode verwendet wird. Bei dieser Methode wird der erwartete Reward eines Zustands über Mittelung der erhaltenen Returnwerte dieses Zustands berechnet. Die Funktion wird auf einem Intervall von [0:50] approximiert. Die Datenpunkte liegen immer auf den Zentren der 25 Basisfunktionen. Jede Basisfunktion hat eine Standardabweichung von $\sigma = 1,0$ und erhält zur Approximation 500 Datenpunkte, somit ergibt sich eine Gesamtzahl von 12500 Datenpunkten. Es werden im Folgenden nun drei Fälle betrachtet: zuerst der Fall, dass kein Rauschen auf ein Datenpunkt addiert wird. Abbildung 5.6 zeigt in a.) die Approximation nach 1000 Datenpunkten und in b.)

das RBF-Netz nach 12500 Datenpunkten. Man sieht, dass das RBF-Netz in der Lage ist die Datenpunkte zu mitteln. Nach 12500 Datenpunkten liegt der Funktionswert des RBF-Netzwerkes bei ungefähr 0,5.

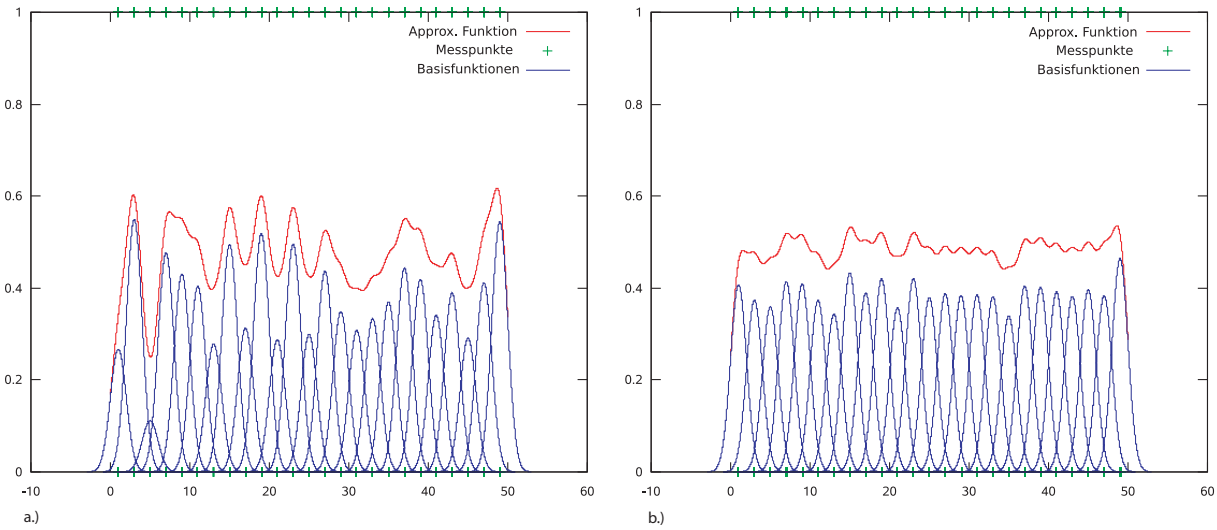

Abbildung 5.6: **Mittelung ohne Rauschen**

Im nächsten Fall wird auf die Datenpunkte ein normalverteiltes Rauschen in Y-Achse addiert. Hierbei werden nun Datenpunkte, die auf 1 gesetzt werden sollten, aus eine Intervall zwischen [0,8:1,0] ausgewählt und Datenpunkte, die den Wert Null erhalten sollten, aus einem Intervall von [0:0,3] gewählt. Abbildung 5.7 zeigt die Approximation in a.) nach 2000 Datenpunkten und in b.) nach 12500 Datenpunkten. Auch hier zeigt sich, dass das RBF-Netz in der Lage ist durch die Datenpunkte zu mitteln.

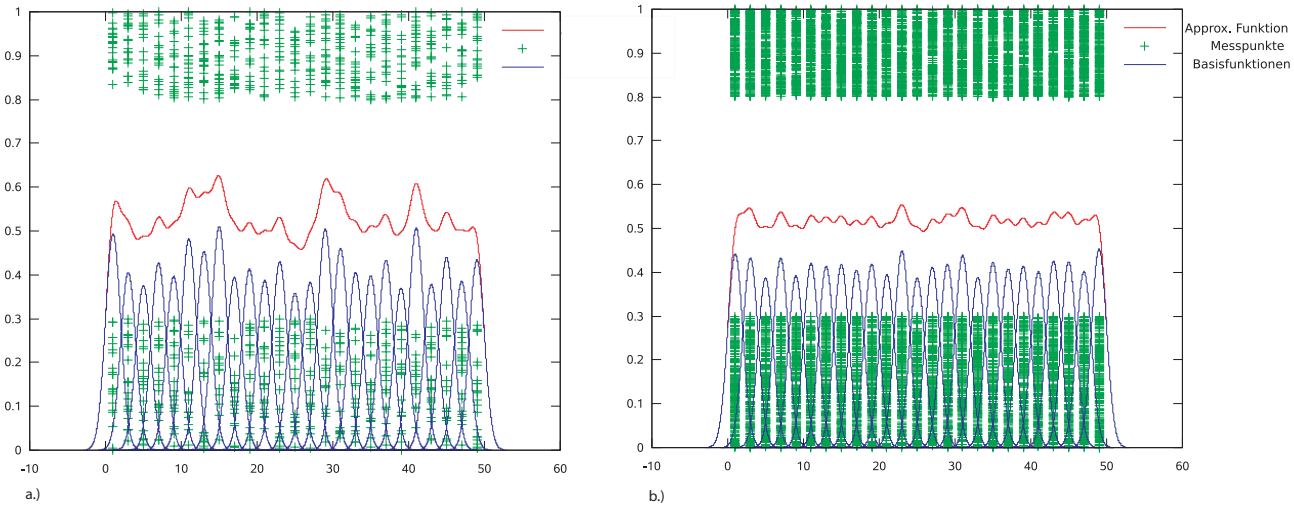

Abbildung 5.7: **Mittelung mit Rauschen in Y-Achse**

Zudem lässt sich feststellen, dass die Approximation der verrauschten Daten genauso lange benötigt wie ohne Rauschen.
Beim letzten Fall wird auf die Datenpunkte sowohl ein Rauschen auf der Y-Achse als auch ein Rauschen auf der X-Achse addiert. Dies entspricht dem Fall, dass die Datenpunkte aus dem Zustands-Aktionsraum zwar bei der Wahl der Aktion auf den Zentren verschoben wurden, aber der eingenommene Zustand, der durch die Bewegungsentscheidung der Navigationsstrategie vorgegeben ist, beibehalten wurde. Somit liegen die Messpunkte nicht auf den Zentren der Basisfunktionen. Abbildung 5.8 zeigt deutlich, dass das RBF-Netzwerk nach 12500 Datenpunkten noch nicht konvergiert ist und somit noch keine Mittelung der Datenpunkte erreicht hat. Zudem dauert die Relaxation 0,7 mal länger als bei den bisher vorgestellten Beispielen.

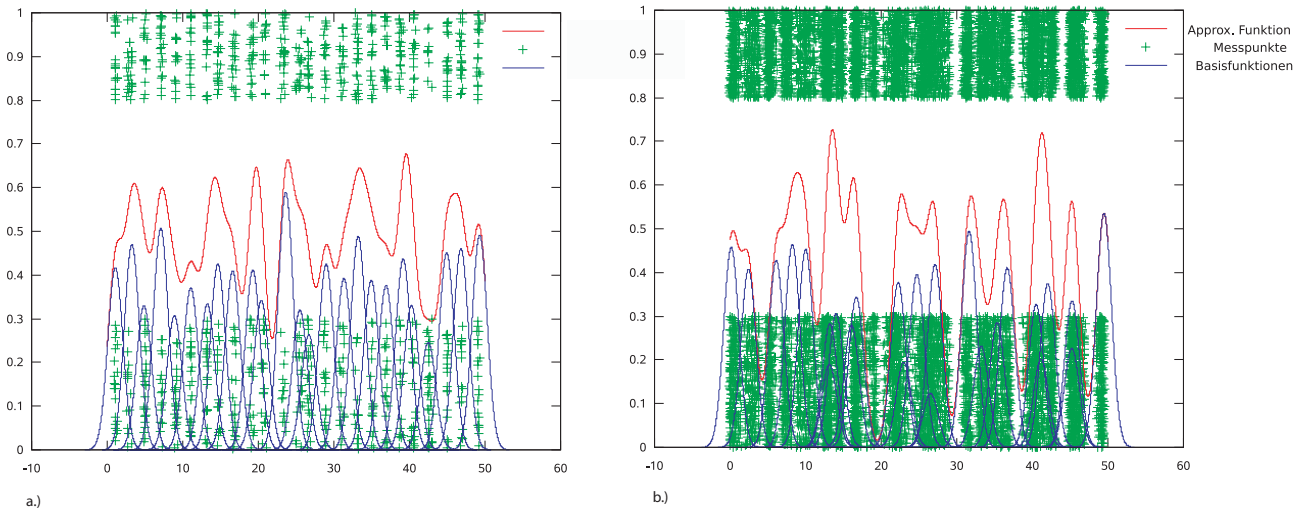

Abbildung 5.8: **Mittelung mit Rauschen auf X- und Y-Achse**

Dieses letzte Beispiel zeigt deutlich, wie entscheidend die Lage der Messpunkte für die Approximationsgüte und Relaxationsgeschwindigkeit ist. Aus diesem Grund werden bei der Approximation der Q-Funktion die Messpunkte quantisiert, was bedeutet, dass der aktuelle Zustand auf die nächstliegende Basisfunktion abgebildet wird und so in die Relaxation eingeht. Durch diese Quantisierung erhält man zwar einen Approximationsfehler, allerdings ist die Approximationsgüte verbessert und die benötigte Rechenzeit der Relaxation geringer.

Kapitel 6

Lernexperiment und Ergebnisse

In diesem Kapitel wird das Lernexperiment vorgestellt, mit dem der Ansatz zur Integration lokaler Navigationsstrategien getestet wird. Wie in Kapitel 4 ausgeführt, wird das Lernexperiment in eine Lern- und eine Verhaltensphase eingeteilt. In der Lernphase erhält der Agent die Möglichkeit Erfahrungen über seine Umgebung zu sammeln. Diese Erfahrungen werden dann in Form der Q-Funktion gespeichert. Hierzu wird die Q-Funktion durch ein RBF approximiert, da sowohl der Zustandsraum als auch der Aktionsraum kontinuierlich sind. Die Einstellung des RBF erfordert bei dieser hochdimensionalen Funktionsapproximation eine sorgfältige und effiziente Implementierung. Aus diesem Grunde werden Veränderungen des RBF-Netzes ausgeführt, die vor allem die Auswahl der Messpunkte und die Maximumssuche betreffen.
In der Verhaltensphase wird nun das erworbene Wissen genutzt um das Verhalten des Agenten zu bestimmen. Die verwendetet Strategie π entspricht dabei einer so genannten „*greedy policy*“, d.h. es werden nur Aktionen ausgewählt, die das Maxima der Q-Funktion in einem Zustand darstellen und somit die höchste Erfolgswahrscheinlichkeit besitzen.
Zum Abschluss dieses Kapitels werden die Ergebnisse besprochen. Hierbei wird anhand von drei Trajektorien gezeigt, welche Entscheidungsstrategie der Roboter in der Lernphase gelernt hat.

6.1 Simulationsumgebung und Roboter

Simulationsumgebung: Bei der Simulationsumgebung handelt es sich um eine 9 qm große Arena, in der sich der Roboter aufhält. Die Form der Arena ist quadratisch. In der Arena befinden sich Hindernisse, die als konvexe Polygone dargestellt sind. Die Hindernisse haben einen zufällig gewählten Grauwert, die Arenawände sind weiß. Die Farbwerte werden zur Simulation des aktuellen Panoramabildes verwendet. Weiterhin werden vier IR-Sensoren simuliert, so dass Hindernisse und die Arenawände der Umgebung detektiert werden können [10]. Abbildung 6.1 zeigt die Benutzeroberfläche der Simulationsumgebung. Diese ist aufgeteilt in „*Environment Map*“, die Darstellung der aktuellen Umgebung, in der der Roboter fährt. Der Roboter wird hier als ein blauer Kreis dargestellt. Die rote Linie markiert seine Ausrichtung (Fahrtrichtung/Heading). Der Bereich „*Views*“

zeigt das aktuelle Kamerabild der Panoramakamera. Über den „*Control*“-Bereich kann die Simulation mit dem start/stop Schalter aktiviert oder beendet werden. Die anderen Schalter werden in dieser Simulation nicht verwendet. Über „*Scale*“ kann man die Größe der abgebildeten Simulationsumgebung verändern (Zoom). Durch die Schalter im Bereich „*Robot*“ ist der Benutzer in der Lage den Roboter manuell zu bewegen.

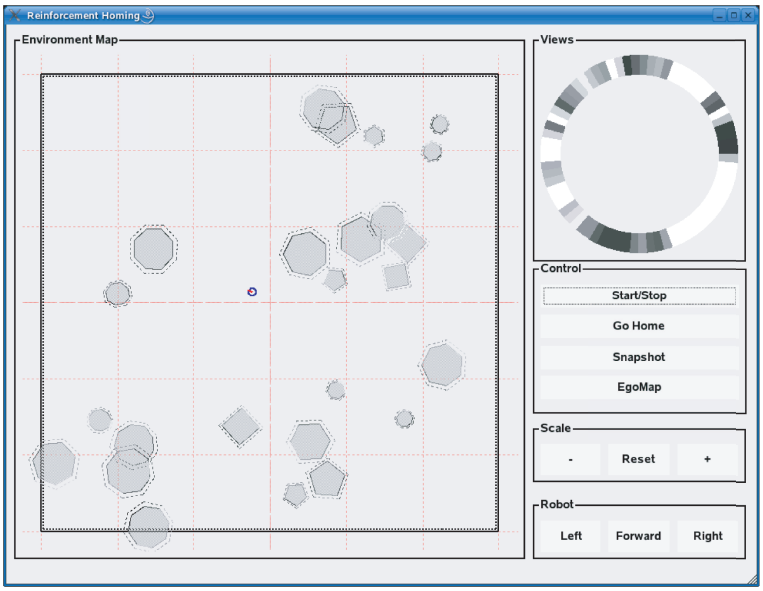

Abbildung 6.1: Simulationsumgebung

Roboter: In der Simulation wird ein virtueller Roboter verwendet. Das reale Vorbild ist der Khepera Miniatur Roboter (K-Team). Die Basisausstattung des realen Khepera umfasst acht IR-Sensoren. Der Roboter wird über zwei Räder mit Schrittmotoren angetrieben. Zusätzlich wurde eine Panoramakamera, entwickelt von Matthias Franz [8], simuliert. Der simulierte Roboter wurde so gestaltet, dass die Ansteuerung vom realen Roboter übernommen werden konnte [10]. Hierzu muss der Antrieb über zwei Räder mit Schrittmotor simuliert werden. Jedes Rad wird dabei um einen diskreten Wert bewegt. Durch das Zählen der Radstände kann der Roboter seine Eigenbewegung berechnen, was die Basis der Odometrie darstellt.
Der simulierte Roboter verfügt über vier Infrarotsensoren. Dadurch ist der Roboter in der Lage mögliche Hindernisse zu detektieren und mit Hilfe der Hindernisvermeidung zu umfahren. Sie sind am Roboter vorne und hinten und an beiden Seiten angebracht.
Die virtuelle Kamera wurde nach dem Vorbild der Panoramakamera von Matthias Franz [8] aufgebaut. Mit Hilfe eines Tracing-Algorithmus wird die virtuelle Aufnahme des Horizontbildes modelliert. Hierzu werden 360 radiale Strahlen mit unendlicher Länge erzeugt. Jedem Strahl wird der Farbwert zugeordnet, den das nächststehende Objekt aufweist, das von dem Strahl geschnitten wird. Die Objekte der Simulationsumgebung haben einen Farbwert zwischen weiß (Arenawände) und Dunkelgrau (Hindernis). Durch diesen Algorithmus wird ein ein 72 Pixel-Panoramabild erzeugt. Mit Hilfe dieses Panoramabildes ist es möglich das visuelle Homing für die Navigation zu nutzen.

6.2 Reinforcement Learning-Problem

Das Reinforcement Learning-Problem wird analog zu Kapitel 4 formuliert. In diesem Abschnitt soll die Anpassung und Parameterwahl auf das Lernexperiment ausgeführt werden.

Belohnung & Rewardfunktion Erreicht der Agent seinen Zielort in einer vorgegebenen Zeit von 15 Zeitschritten, erhält er eine Belohnung ($r_t = 1,0$), ansonsten erhält er eine Bestrafung von $r_t = -0,015$.

$$r_t = \begin{cases} 1,0 & \text{,wenn Ziel erreicht} \\ -0,015 & \text{,sonst} \end{cases} \tag{6.1}$$

Durch die Bestrafung wird die Rewardfunktion für eine Sequenz hyperbelförmig. Würde diese Bestrafung nicht erfolgen, hätte die Rewardfunktion für einen erfolgreichen Durchlauf die Form einer Boxfunktion. Dies ist für die Approximation der Q-Funktion hinderlich, da das RBF-Netz nicht ohne weiteres in der Lage ist konstante Bereiche mit niedriger Varianz zu approximieren. Der Grund hierfür ist, dass das RBF-Netz durch eine Linearkombination aus Gaußkurven dargestellt ist, die keine konstanten Bereiche darstellen können.
Durch die gegebenen Returns kann nun der Reward eines Zustands-Aktionspaares einer Sequenz berechnet werden. Bei der durchgeführten Lernaufgabe wurde die „Discounting Rate“ $\gamma = 0,9$ gesetzt.

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{6.2}$$

Zustands-Aktionsraum Um die Komplexität des aufgestellten Systems so minimal wie möglich zu gestalten, wurden zwei Zustandsdimensionen, die Distanz des visuellen Homings d_t^h und die Distanz der Hindernisvermeidung d_t^o, nicht in das System übernommen. Die Wahl fiel auf diese beiden Dimensionen, da zum einen die Distanz der Hindernisvermeidung randomisiert gewählt wird und somit keine Information für das System liefert. Zum anderen ist die Distanz des visuellen Homings sehr fehleranfällig. Der Zustandsraum ist wie folgt definiert:

- ρ_t^h = vorgeschlagene Richtungsentscheidung des visuellen Homings
- ρ_t^p = vorgeschlagene Richtungsentscheidung der Wegintegration
- d_t^p = vorgeschlagene Distanz der Wegintegration
- ρ_t^o = vorgeschlagene Richtungsentscheidung der Hindernisvermeidung
- t = Zeitschritt

Der Aktionsraum wird über die α und β Werte gebildet. Somit ergibt sich ein siebendimensionaler Zustands-Aktionsraum.

Gewichtung der Navigationsstrategien Über die Gewichtung der einzelnen Navigationsstrategien wird die Bewegungsentscheidung des Roboters für jeden Zeitschritt neu berechnet. In diesem Aufbau wird die Distanz d_t^m des integrierten Bewegungsvektors $\boldsymbol{p}_t^m = (\rho_t^m, d_t^m)$ vorgegeben, die Länge eines einzelnen Schrittes beträgt 2 cm. Die Rotation ρ_t^m des Bewegungsvektors $\boldsymbol{p}_t^m$ wird über die Gewichtung der Strategien festgelegt:

$$\rho_t^m = \alpha\rho_t^h + \beta\rho_t^p + (1 - \alpha - \beta)\rho_t^o \tag{6.3}$$

Je nach Setzung von α und β gehen die Richtungsentscheidungen ρ_t^x unterschiedlich stark in die Bewegungsentscheidung des Roboters ein. Die α und β Werte sind die Aktionen, die der Agent in jedem Zustand wählen kann. In der Lernphase werden α und β zufällig gewählt. In der Verhaltensphase werden die α und β Werte über die Q-Funktion bestimmt. Sie stellen das Maxima der Q-Funktion des aktuellen Zustands dar.

Lernaufgabe In diesem Lernexperiment hat der Roboter die Aufgabe zu einem Zielpunkt zurückzukehren. Hierzu wird der Roboter an einen Ort in der Umgebung gesetzt, macht dort mit der Panoramakamera eine Aufnahme der Umgebung und aktualisiert seinen Wegintegrator. Danach wird er 28 cm in eine zufällig ausgewählte Richtung weggefahren. Abbildung 6.2 zeigt diese Aufgabenstellung. Das rote Kreuz markiert den Startpunkt, der gleichzeitig den Zielort der Heimfindung darstellt. Der rote Kreis hat ungefähr einen Radius von 28 cm und zeigt die Distanz, über die der Roboter wieder heimfinden muss.

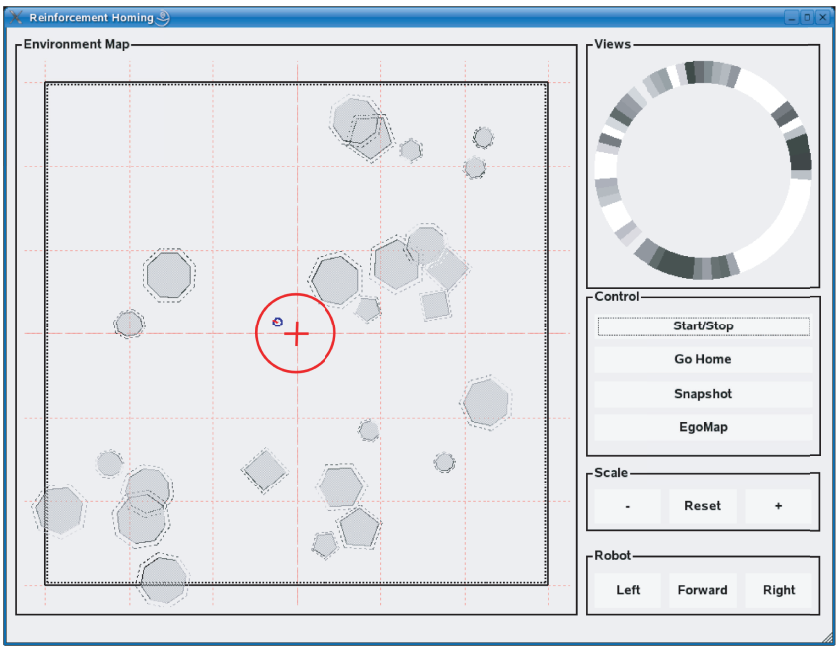

Abbildung 6.2: Simulationsumgebung

6.3 Einstellung des RBF

Die Approximation der Q-Funktion erfolgt durch ein RBF-Netz. Als erstes muss der zu approximierende Bereich definiert werden, des weiteren muss die Standardabweichung σ der einzelnen Gaußkurven festgelegt werden. Aus Effizienzgründen wurde zudem eine spezielle Messpunktauswahl implementiert und die Maximumssuche speziell für diese Problemstellung verbessert.

6.3.1 Skalierung des Zustands-Aktionsraumes

Der Bereich, in dem das RBF definiert werden soll, ist durch den Zustands-Aktionsraum festgelegt. Die einzelnen Dimensionen des Zustands-Aktionsraumes stellen gleichzeitig die Dimensionen des RBF dar. Allerdings sind die einzelnen Dimensionen in unterschiedlich großen Bereichen definiert. Die Rotationen ρ_t sind als Winkel in Bogenmaß definiert. Sie sind im Bereich von $-\pi$ bis π angegeben. Die Distanz d_t^p der Wegintegration ist im Bereich von 0 bis 5 cm definiert. Die Zeit t hängt von der benötigen Zeit des Agenten für die gefahrene Sequenz ab und ist durch die Zeitgrenze zwischen 0 und 15 Zeitschritten eingegrenzt. Die α und β Werte liegen zwischen $[0:1]$. Durch diese unterschiedlich großen Bereiche der Dimensionen ist es notwendig eine Skalierung der Dimensionswerte vorzunehmen, so dass die einzelnen Dimensionen auf gleichgroße Bereiche abgebildet werden können. Hierzu werden die einzelnen Dimensionen so transformiert, dass alle Dimensionen in ein Intervall von $[0:10]$ fallen.

6.3.2 Parametereinstellung

Die Größe der einzelnen Gaußkurven, die die Basisfunktionen des verwendeten RBF darstellen, ist abhängig von der Einstellung der Standardabweichung σ. Bei dieser Anwendung des RBF ist σ auf 1,0 gesetzt worden. Der Überlapp der einzelnen Basisfunktionen wurde auf σ festgelegt. Durch die Wahl des σ-Wertes und die Definition des Überlapps ist zugleich die Anzahl der Basisfunktionen festgelegt. Das verwendete RBF wird mit einer Anzahl von 78125 Basisfunktionen initialisiert, was bedeutet, dass jede Dimension durch fünf Basisfunktionen abgetastet wird.

6.3.3 Messpunktauswahl

Die Wahl der Messpunktorte ist entscheidend für die Qualität der Approximation (siehe 5.5). Zudem ist die Lage der Messpunkte auch ein maßgeblicher Faktor, wie viel Zeit benötigt wird, die Koeffizienten durch das Relaxationsverfahren zu bestimmen. Bei den ersten Versuchen wurde festgestellt, dass das Relaxationsverfahren, wenn die Messpunkte nicht auf den Zentren der einzelnen Basisfunktionen liegen, im Vergleich zu dem Fall, dass die Messpunkte auf den Zentren liegen, 0,7 mal mehr Rechenzeit benötigt. Dies wurde bei der Wahl der Lage der Messpunkte berücksichtigt.
Ein Messpunkt wird in diesem Ansatz durch die gegebenen Zustandsparameter und die Wahl der Aktion beschrieben. Die gegebenen Zustandsvariablen sind durch die lokalen

Navigationsstrategien bestimmt. Je nach Situation schlägt jede Strategie einen Bewegungsvektor vor. Die Zustände setzen sich aus den Rotationsentscheidungen $(\rho_t^h, \rho_t^p, \rho_t^o)$ und der Distanzentscheidung (d_t^p) zusammen. Zudem wird der jeweilige Zeitschritt in die Zustandsmenge aufgenommen. Die Aktionen können in der Lernphase, in der die Approximation der Q-Funktion erfolgt, frei gewählt werden. Da die Messpunkte auf den Zentren der Basisfunktionen liegen sollten, werden die Aktionen zuerst zufällig bestimmt und anschließend überprüft, welche Basisfunktion zu diesem Messpunkt (Zustand + Aktion) am nächsten liegt. Das Zentrum der nächstliegende Basisfunktion bestimmt so die Wahl der Aktion (α und β). Die Lage der Messpunkte wird quantisiert, d.h. sie werden auf die nächstliegende Basisfunktion geschoben. Dadurch wird erreicht, dass jeder Datenpunkt, der in die Relaxation einfließt, auf dem Zentrum einer Basisfunktion liegt. Zwar kommt es durch dieses Vorgehen zu Messungenauigkeiten, aber die Ungenauigkeit ist akzeptabel, da diese Anwendung keine sehr gute Approximation der Q-Funktion benötigt. Entscheidender ist, dass zum einem die Relaxation in annehmbarer Zeit terminiert und zum anderen eine gute Mittelung der Datenpunkte erfolgen muss. Dies ist durch die Quantisierung der Messpunkte möglich (siehe 5.5).

6.3.4 Einschränkung der Maximumssuche

Die Suche der Maxima stellt in einem siebendimensionalen Raum eine zeitaufwendige Prozedur dar. Die Maxima der approximierten Q-Funktion repräsentieren die Zustands-Aktionspaare, die in der Lernphase die höchste Wahrscheinlichkeit zeigten, dass die Heimfindungsaugabe erfolgreich gelöst wurde. In der Verhaltensphase wird nun das gewonnene Wissen genutzt um in der jeweiligen Situation die Aktion zu wählen, die am wahrscheinlichsten zum Erfolg führt. Hierfür muss für einen Zustand $(\rho_t^h, \rho_t^p, d_t^p, \rho_t^o, t)$ die Aktion (α, β) bestimmt werden, die das Maximum für diesen Zustand darstellt. Um die Maxima zu bestimmen wird die vorgestellte Maximumssuche (siehe 5.3) verwendet. Anders als in 5.3 vorgestellt, wird nun die Maximumssuche von der Relaxation abgetrennt, da nicht alle Maxima der approximierten Q-Funktion benötigt werden und es somit effizienter ist nur die notwendigen Maxima zu bestimmen. Da allerdings der Zustand durch die Navigationsstrategien schon vorgegeben ist, werden diese Parameter in der Maximumssuche nicht mit optimiert und die Maximumssuche erfolgt somit nur in einem Unterraum des aufgespannten Zustands-Aktionsraumes. Dieser Unterraum ist durch den gegebenen Zustand definiert und besitzt nur α und β als freie Parameter. Aus diesem Grund wird aus der Maximumssuche im siebendimensionalen Raum eine Suche im zweidimensionalen Unterraum. Durch dieses Vorgehen kann die Maximumssuche wesentlich effizienter erfolgen. Das gefundene Maximum stellt nun die α und β Werte dar, welche die höchste Wahrscheinlichkeit besitzen zu einer erfolgreichen Heimfindung zu führen. Falls für den gegebenen Zustand $(\rho_t^h, \rho_t^p, d_t^p, \rho_t^o, t)$ kein Maximum gefunden wird, da zum Beispiel in der Lernphase keine Information über diesen Zustand gesammelt wurde, werden die α und β Werte zufällig gewählt.

6.4 Lernphase

In der Lernphase soll der Agent lernen, wie er sich in einer gegebenen Situation $(\rho_t^h, \rho_t^p, d_t^p, \rho_t^o, t)$ für welche Aktion (α und β) entscheiden soll. Das Lernen erfolgt über die Erfahrungen des Agenten, die er in der Lernphase sammelt. Die Lernphase ist dabei in 10 Episoden aufgeteilt. In jeder Episode macht der Roboter eine Aufnahme seines Startpunktes und aktualisiert seinen Wegintegrator. Danach wird der Roboter 28 cm in eine zufällige Richtung weggefahren. In der Heimfindungsphase versucht der Roboter seinen Zielpunkt wieder zu erreichen. Während der Lernphase werden in jedem Zustand die Aktionen zufällig gewählt (6.3.3). Für jedes Zustands-Aktionspaar erhält der Agent einen Returnwert r_t. Nach jedem Durchlauf wird die approximierte Q-Funktion aktualisiert. Ist die Relaxation abgeschlossen, kann der nächste Durchgang gestartet werden. Abbildung 6.3 zeigt schematisch den Ablauf der Lernphase.

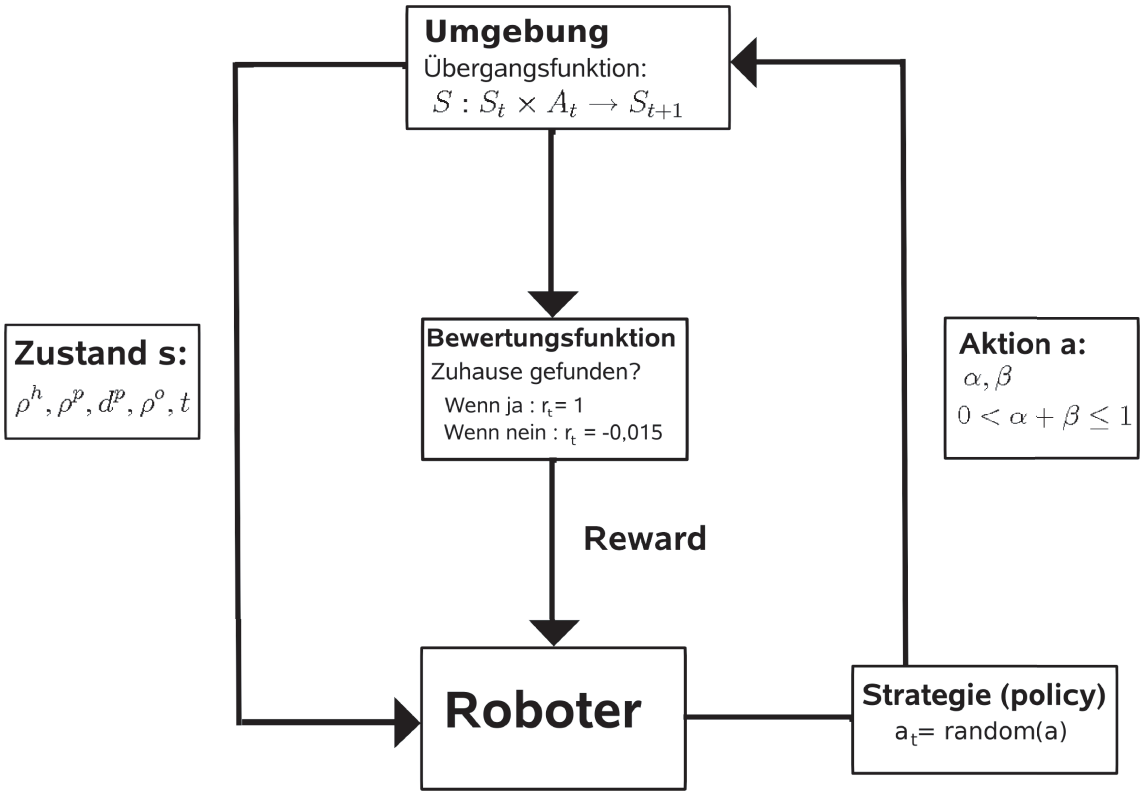

Abbildung 6.3: **Schematische Darstellung der Lernphase**

6.5 Verhaltensphase

In der Verhaltensphase (siehe 4.2.2) wendet der Agent sein gewonnenes Wissen aus der Lernphase an. Abgesehen vom Strategiewechsel („*policy*“) ist die Verhaltensphase mit der Lernphase identisch. Bei der Heimfindung nutzt der Agent nun sein in der Lernphase erlangtes Wissen. Deshalb wird in jedem Zustand das Maximum der Q-Funktion für diesen Zustand bestimmt (siehe 6.3.4). Die verwendete Strategie in der Verhaltensphase entspricht der „*greedy policy*“:

$$\pi(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_{a} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{6.4}$$

Abbildung 6.4 zeigt den schematischen Ablauf der Verhaltensphase.

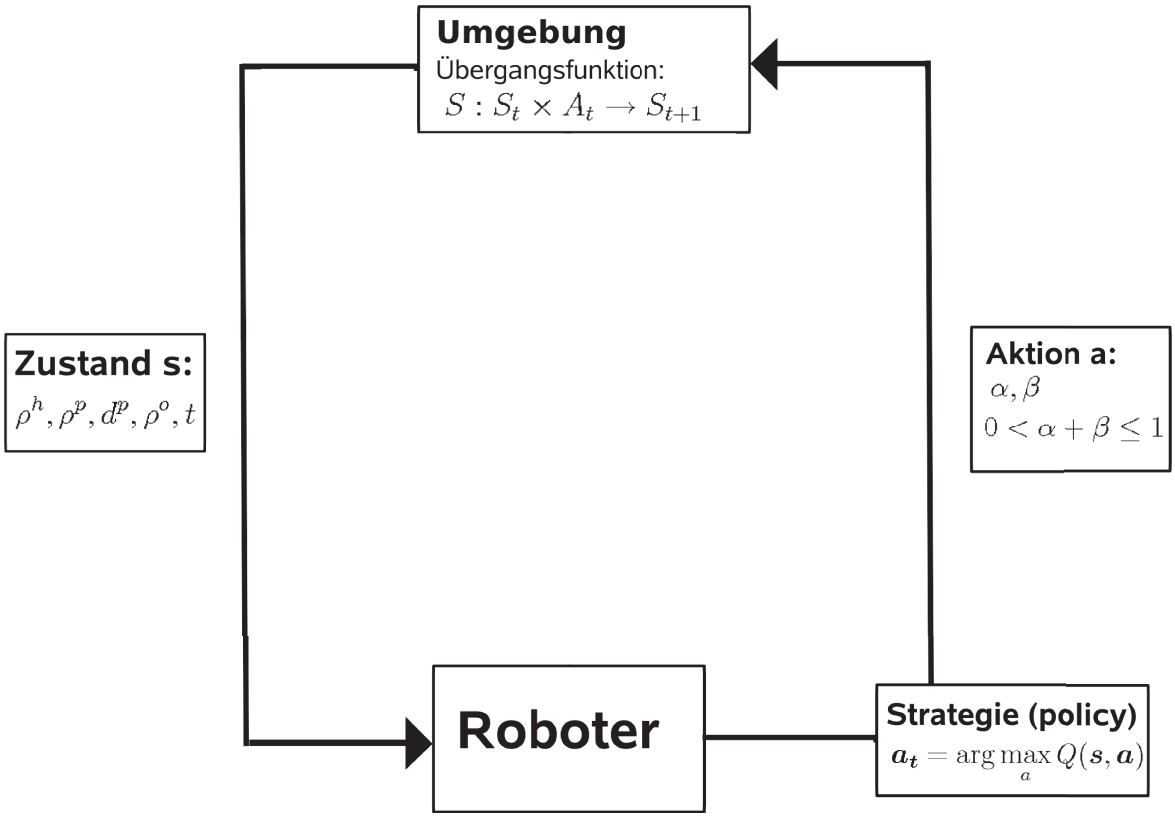

Abbildung 6.4: **Schematische Darstellung der Verhaltensphase**

6.6 Ergebnisse

Nach der Lernphase, in der die Entscheidung (Wahl von α und β) zufällig getroffen wurde, wird nun in der Verhaltensphase überprüft, welche Entscheidungsstrategie der Agent gelernt hat. Sowohl in der Lern- als auch in der Verhaltensphase werden 10 Durchläufe ausgeführt. Der Ablauf in Lern- und Verhaltensphase ist identisch. In der Heimfindungsphase der Verhaltensphase bestimmt der Agent nun die α und β Werte (die Aktionen) über die in der Lernphase approximierte Q-Funktion. Das Maximum der Q-Funktion zum aktuellen Zustand ($\boldsymbol{s} = \rho_t^h, \rho_t^p, d_t^p, \rho_t^o, t$) geben die Aktionen α und β an.

$$\pi(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_a Q(\boldsymbol{s}, \boldsymbol{a}) \tag{6.5}$$

Anhand von drei beispielhaften Trajektorien soll nun dargestellt werden, welche Strategie $\pi(\boldsymbol{s}, \boldsymbol{a})$ der Agent gelernt hat. Eine Trajektorie zeigt den Weg, den der Agent in der Heimfindungsphase zurückgelegt hat. Jeder Zeitpunkt, an dem der Agent α und β neu bestimmen muss, wird nun im folgenden Entscheidungspunkt genannt. Nach der Berechnung der α und β Werte folgt der Roboter seinem berechneten Bewegungsvektor $\boldsymbol{p}_t^m = (\rho_t^m, d_t^m)$, wobei $d_t^m = 2cm$ ist. Die Rotation ρ_t^m ergibt sich aus der gewichteten Summe der Rotationen der Navigationsstrategien:

$$\rho_t^m = \alpha\rho_t^h + \beta\rho_t^p + (1 - \alpha - \beta)\rho_t^o \tag{6.6}$$

Alle Rotationen werden egozentrisch, d.h. bezogen auf das aktuelle Heading des Roboters, angegeben. In den Abbildungen 6.5, 6.7, 6.9 werden durch Pfeile die Rotationsanteile der einzelnen Navigationsstrategien, die Bewegungsentscheidung und die Ausrichtung (Heading) des Roboters dargestellt:

- schwarz gestrichelter Pfeil = aktuelles Heading des Roboters
- roter Pfeil = Rotation ρ_t^h des visuellen Homings
- grüner Pfeil = Rotation ρ_t^p der Wegintegration
- blauer Pfeil = Rotation ρ_t^o der Hindernisvermeidung
- oranger Pfeil = Rotation ρ_t^m der Bewegungsentscheidung

Grüne Kreuze kennzeichnen die Entscheidungspunkte, an welchen der Roboter α und β neu bestimmt hat. Die Trajektorie selbst ist rot dargestellt.
Abbildung 6.5 zeigt die Trajektorie der ersten Episode. Der Roboter beginnt seine Heimfindungsphase am Punkt (29,5;-5,5). Im Punkt (7,5;1,2) endet die Trajektorie, da der Roboter seinen Zielpunkt erreicht hat. Exemplarisch wird an vier Entscheidungspunkten gezeigt, wie sich die berechnete Bewegungsentscheidung zusammensetzt. Im ersten Entscheidungspunkt auf der Trajektorie wurden $\alpha = 0,3$ und $\beta = 0,1$ gesetzt. Die Ausrichtung des Roboters ist bezüglich des Startpunktes genau um 180° gedreht, da er von diesem Ort am Anfang des Durchlaufs weggefahren wurde und sich währenddessen nicht

gedreht hat. Wie man sieht, zeigt die Rotation der Wegintegration, in grün dargestellt, in Richtung (ρ_t^p = -180°) des Zielpunktes. Alle anderen Rotationen liegen bei Null und zeigen somit in die Richtung des aktuellen Headings. Durch die vorgegebene Gewichtung ergibt sich daher die Rotation $\rho_1^m = -18°$. Durch diese niedrige Gewichtung der Wegintegration vollzieht der Roboter keine komplette Drehung auf seinen aktuellen Zielpunkt hin. Erst im nächsten Entscheidungspunkt dreht sich der Roboter so, dass seine Ausrichtung zum Zielpunkt zeigt. In allen folgenden Entscheidungspunkten stimmt die Ausrichtung des Roboters mit der Richtung des Zielpunktes überein. Aus diesem Grund liefern alle Navigationsstrategien den Rotationswert Null. Der Rotationsanteil der Hindernisvermeidung liefert, wie auch in den folgenden Trajektorien, immer den Wert Null, da bei allen Durchläufen kein Hindernis detektiert wurde. Selbst wenn der Roboter ein Hindernis detektieren würde, könnte er nicht darauf reagieren, da während der Lernphase nie ein Hindernis aufgetaucht ist.

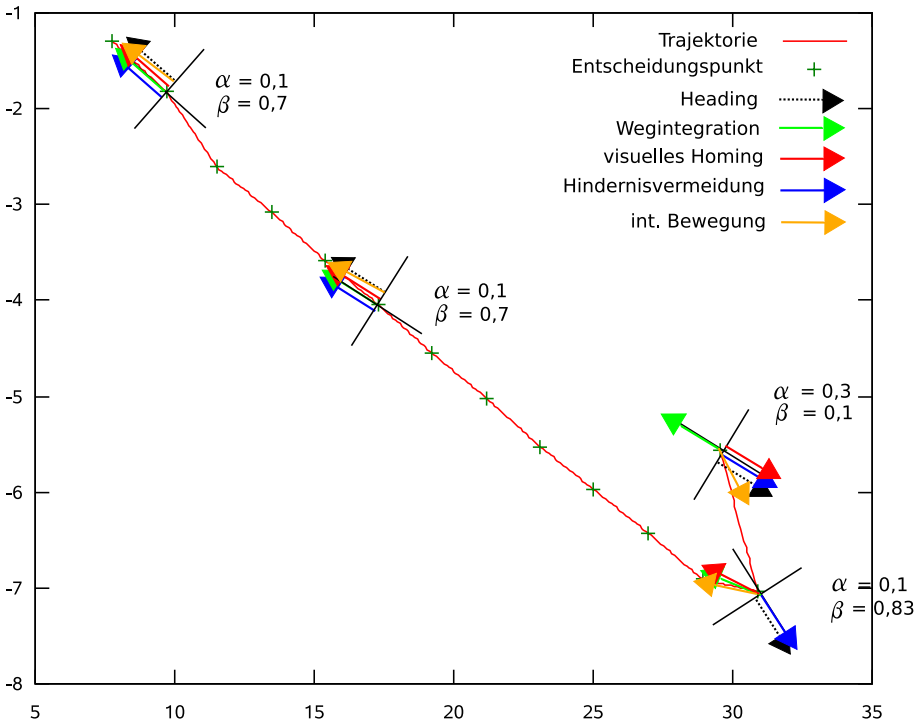

Abbildung 6.5: **Beispieltrajektorie 1:**

Die Abbildungen 6.6, 6.8, 6.10 zeigen den zeitlichen Verlauf von α und β für die einzelnen Durchläufe. Auf der X-Achse ist die Zeit angegeben, auf der Y-Achse sind die α und β Werte dargestellt. Die Hindernisvermeidung ergibt sich hierbei aus $\gamma = 1 - \alpha - \beta$.
Abbildung 6.6 zeigt die Gewichtung der einzelnen Navigationsstrategien für den ersten Durchlauf.

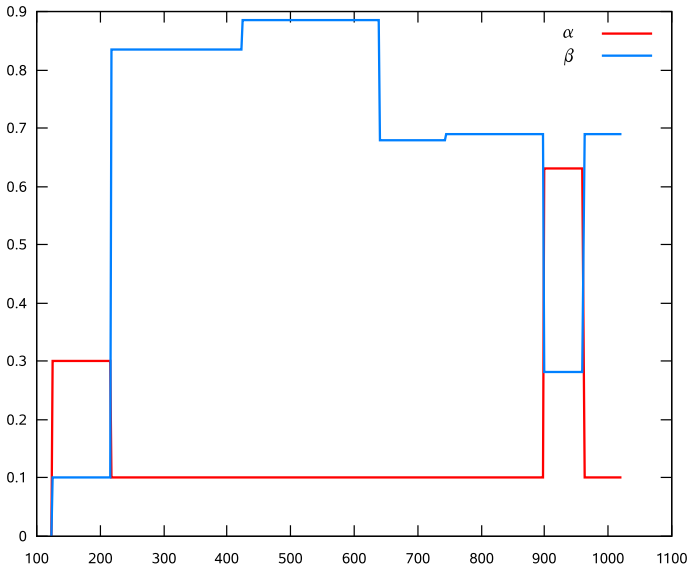

Abbildung 6.6: **Gewichtung der Navigationsstrategien der Trajektorie 1**

Man sieht in Abbildung 6.6 deutlich, dass β meist höher ist als α, was zur Folge hat, dass der Roboter die Wegintegration gegenüber dem visuellen Homing wesentlich stärker gewichtet. Nur in zwei von 15 Entscheidungspunkten (im ersten und im vierzehnten) war die Gewichtung des visuellen Homings höher.
In der zweiten Episode lief der Roboter in der Heimfindungsphase die Trajektorie aus Abbildung 6.7. Auffällig bei dieser Trajektorie ist, dass sie sehr geradlinig zum Zielpunkt zurückführt. Dies trifft auch auf die nicht gezeigten Trajektorien zu. Der Anfangspunkt liegt bei diesem Durchlauf bei (29,5;19,5), der Zielpunkt bei (0;7,9). Im ersten Entscheidungspunkt wird die Wegintegration mit $\beta = 0,9$ gewichtet. Die Richtungsentscheidung der Wegintegration beträgt $\rho_1^p = -180°$. Das visuelle Homing liefert einen Wert von $\rho_1^h = 150°$. Durch die Gewichtung ergibt sich eine Rotation von $\rho_1^m = -147°$. In den folgenden Entscheidungspunkten wird die Wegintegration immer höher gewichtet als das visuelle Homing $\beta > \alpha$. Erst im 13. Entscheidungspunkt erfolgt ein Wechsel in der Gewichtung. In diesem Entscheidungspunkt wird das visuelle Homing mit $\alpha = 0,7$ gewichtet und die Wegintegration mit $\beta = 0,3$. Das visuelle Homing zeigt dabei eine vorgeschlagene Rotation von $\rho_{13}^h = -30°$ und die Wegintegration von $\rho_{13}^p = -6°$. Daraus ergibt sich die Rotation von $\rho_{13}^m = -22,8°$. Im 13. und im letzten Entscheidungspunkt wird die Wegintegration wieder höher gewichtet als das visuelle Homing.

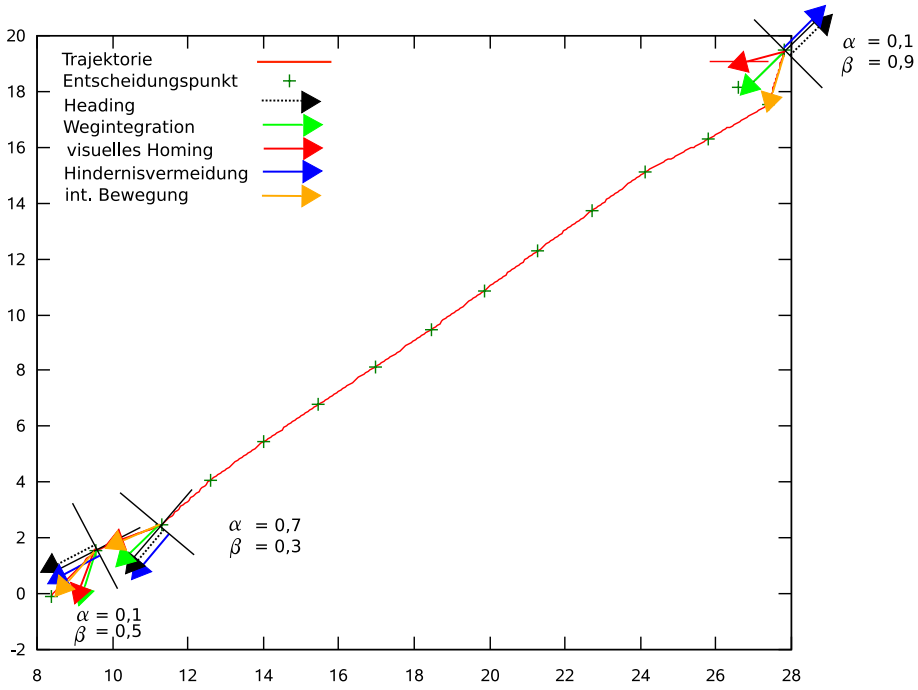

Abbildung 6.7: **Beispieltrajektorie 2**

Die Hindernisvermeidung liefert während der gesamten Trajektorie den Rotationswert $\rho_t^o = 0$, da kein Hindernis auf dem Weg detektiert wurde.
Abbildung 6.8 stellt den zeitlichen Verlauf der Gewichtung der Navigationsstrategien für den zweiten Durchlauf dar. Auch bei dieser Trajektorie zeigt sich, dass die Wegintegration deutlich höher gewichtet wurde als das visuelle Homing ($\beta > \alpha$). Nur im Entscheidungspunkt 13 wurde das visuelle Homing höher gewichtet als die Wegintegration.

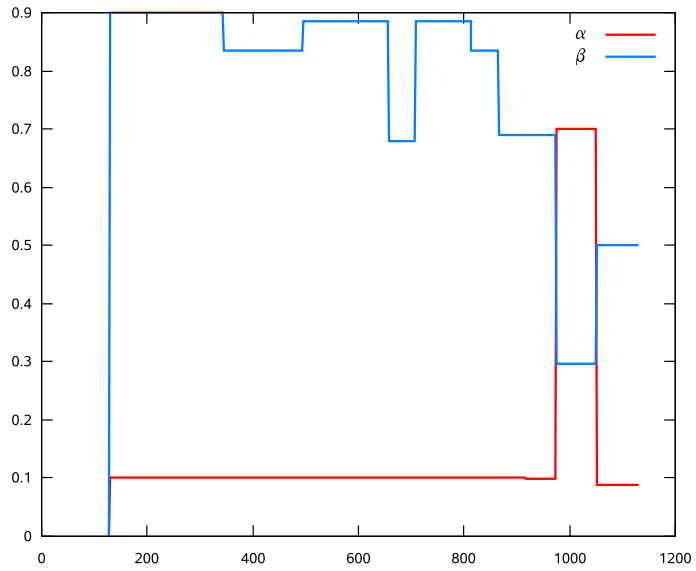

Abbildung 6.8: **Gewichtung der Navigationsstrategien der Trajektorie 2**

Bei der letzten Trajektorie der Verhaltensphase lag der Startpunkt der Heimfindungsphase bei Punkt (19,5;12). Der Zielort befand sich an Punkt (7,5;13). Exemplarisch sind in Abbildung 6.9 nun vier Entscheidungspunkte dieser Trajektorie schematisch dargestellt. Auch in dieser Trajektorie wurde auf der Fahrt kein Hindernis detektiert, somit liefert die Hindernisvermeidung während des Durchlaufs immer den Wert $\rho_t^o = 0°$. Im ersten Entscheidungspunkt liegt die Gewichtung der Naviationsstrategien bei $\alpha = 0,1$ und $\beta = 0,7$. Sowohl die Richtungsentscheidung der Wegintegration als auch die des visuellen Homings zeigen in Richtung des Zielpunktes. In diesem Entscheidungspunkt liefert die Wegintegration den Wert $\rho_1^p = 180°$ und das visuelle Homing $\rho_1^h = 164°$. Über die gesetzte Gewichtung ergibt sich eine Richtungsentscheidung von $\rho_1^m = 142°$. Im zweiten Entscheidungspunkt werden sowohl die Wegintegration als auch das visuelle Homing mit $\alpha = \beta = 0,3$ gewichtet. Der Rotationsanteil der Wegintegration beträgt $\rho_2^p = 40°$. Das visuelle Homing schlägt eine Rotation von $\rho_2^h = 30°$ vor. In der durchgeführten Bewegung lag der Rotationsanteil bei $\rho_2^m = 21°$. Nach diesem Entscheidungspunkt wurde die Wegintegration bis zum Entscheidungspunkt 12 höher gewichtet als das visuelle Homing. Im Entscheidungpunkt 12 wurde das visuelle Homing mit $\alpha = 0,63$ und die Wegintegration mit $\beta = 0,3$ gewichtet. Das visuelle Homing gibt an diesem Ort eine Richtungsentscheidung von $\rho_{12}^h = -15°$. Der Rotationsanteil der Wegintegration beträgt $\rho_{12}^p = 5°$. Somit ergibt sich für den Entscheidungspunkt 12 eine Rotation von $\rho_{12}^m = 7,95°$. Nach diesem Entscheidungspunkt erfolgt wieder eine höhere Gewichtung der Wegintegration. Im 13. Entscheidungspunkt liegt die Gewichtung der Wegintegration bei $\beta = 0,9$, das visuelle Homing wird mit $\alpha = 0,1$ gewichtet. Diese Gewichtung wird bis zum Ende des Durchlaufs beibehalten.

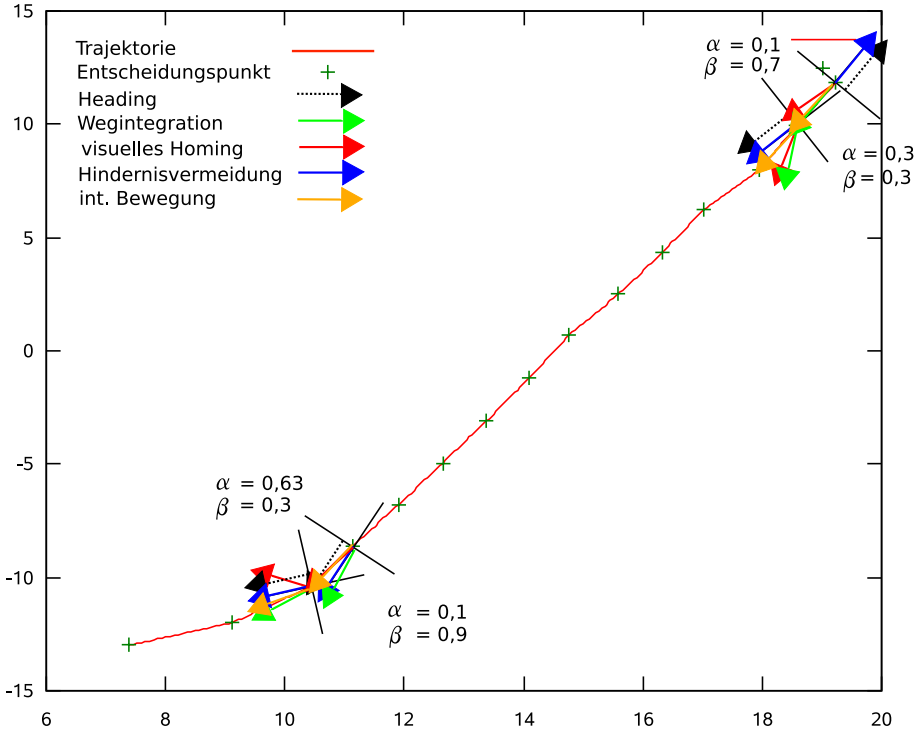

Abbildung 6.9: **Beispieltrajektorie 10**

Abbildung 6.10 zeigt den zeitlichen Verlauf von α und β für die 10. Episode.

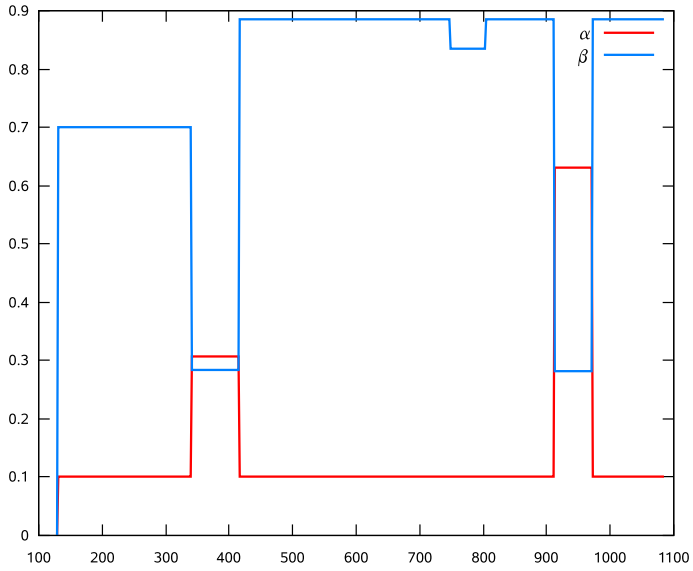

Abbildung 6.10: **Gewichtung der Navigationsstrategien der Trajektorie 10**

Auch in diesem Durchlauf (6.10) zeigt sich deutlich, dass die Wegintegration höher gewichtet wurde als das visuelle Homing. Nur zwei Mal wurde das visuelle Homing höher gewichtet als die Wegintegration. Diese geschah im 2. und im 12. Entscheidungspunkt. Nach der Untersuchung der gefahrenen Trajektorien kann man sagen, dass der Roboter in der Lernphase eine Gewichtung der Navigationsstrategien gelernt hat, da in jedem Entscheidungsschritt ein Maximum der Q-Funktion für den aktuellen Zustand gefunden wurde. Auch wurde bei den Untersuchungen deutlich, dass die eingenommenen Zustände in der Heimfindungphase sehr ähnlich waren, da oft bereits gefundene Maxima aus den vorherigen Durchläufen in den folgenden Trajektorien verwendet wurden. Zudem zeigte sich in der Verhaltensphase, dass die Wegintegration in der Regel höher gewichtet wurde als das visuelle Homing. Über die Hindernisvermeidung kann mit diesen Ergebnissen keine Aussage gemacht werden, da in keinem Durchlauf der Roboter ein Hindernis detektiert hat und deshalb die Richtungsentscheidung dieser Navigationsstrategie immer $\rho_t^o = 0°$ war.

Kapitel 7

Diskussion und Ausblick

7.1 Diskussion

Durch die vorgestellten Ergebnisse konnte gezeigt werden, dass der Roboter in der Lernphase die Gewichtung der Navigationsstrategien gelernt hat. Allerdings ist hinter der Gewichtung der Navigationsstrategien noch keine klare Strategie erkennbar. Was man sagen kann, ist dass die Wegintegration fast immer höher gewichtet wurde ($\beta > \alpha$). Der Grund für diese hohe Gewichtung der Wegintegration könnte sein, dass in der bisherigen Simulationsumgebung kein Rauschen auf die Sensoren gelegt wurde. Somit zeigt der durch die Wegintegration berechnete Heimvektor immer präzise zum Zielpunkt zurück.
Zudem ist anscheinend die Distanz von 28 cm zwischen Start- und Zielpunkt zu gering gewählt, so dass der Roboter sich sehr häufig noch im Fangbereich C (siehe 3.1.1.1) des Zielpunktes befand. Dies zeigt sich dadurch, dass das visuelle Homing schon am Anfang der Trajektorie in Richtung des Zielpunktes zeigt. Dies kann nur der Fall sein, wenn der Roboter sich schon im Fangbereich des Zielpunktes befindet. Wenn allerdings sowohl das visuelle Homing als auch die Wegintegration in die Richtung des Zielpunktes weisen, ist die Gewichtung zwischen der Wegintegration und dem visuellen Homing nicht entscheidend.
Die Hindernisvermeidung war in allen Durchläufen nicht aktiv, somit kann mit den vorliegenden Ergebnissen keine Aussage über die Gewichtung der Hindernisvermeidung gemacht werden. Allerdings stellt sich die Frage, aus welchem Grund die Gewichtung der Wegintegration gegenüber dem visuellen Homing sich nicht immer auf 1 summierte ($\alpha+\beta < 1$). Das sollte nur der Fall sein, wenn die Hindernisvermeidung in diesem Augenblick aktiv wäre. Ein Grund dafür könnte sein, dass in der Lernphase über das betroffene Zustands-Aktionspaar noch nicht ausreichend Information gesammelt wurde, und so ein Generalisierungsfehler für dieses Zustands-Aktionspaar in der Q-Funktion vorliegt. Ein anderer Grund für die Unterschätzung der Gewichtung von Wegintegration und visuellem Homing könnte sich auf Approximationsfehler an den Randbereichen zurückführen lassen. Es ist bekannt, dass die Approximation von Randpunkten für ein RBF-Netz nicht sehr präzise ist . Durch die Skalierung der einzelnen Zustandsdimensionen wurden die Rotationswinkel $-180°$ und $180°$ auf die Ränder des RBFs Bereiches geschoben. Genau

für diese Zustände trate oft eine geringere Gewichtung der Wegintegration und des visuellen Homings ein. Eine einfache Lösung dieses Problems wäre, das Koordinatensystem für diese Dimension periodisch zu wählen. Eine weitere Möglichkeit diese Fehlerquelle auszuschließen wäre die Verwendung von normalisierten RBF-Netzen zur Approximation der Q-Funktion . Diese spezielle Art von RBF-Netzwerken hat den Vorteil, dass sie eine wesentlich glattere Q-Funktion approximieren könnte und zudem kaum Randprobleme aufweist. . Ein normalisiertes RBF-Netz (NRBF) ist wie folgt definiert:

$$NRBF_j(\boldsymbol{x}) = c_j \frac{\phi_j(\boldsymbol{x})}{\sum_{i=0}^{N} \phi_i(\boldsymbol{x})} \tag{7.1}$$

Abschließend kann man sagen, dass durch die Ergebnisse gezeigt werden konnte, dass der Agent durch den vorgestellten Ansatz in der Lage ist die Gewichtung der Navigationsstrategie situations- und zeitabhängig zu wählen. Allerdings müssen für die genauere Analyse des Lernverfahrens noch weitere Experimente durchgeführt werden, bei denen die Entfernung zwischen Start- und Zielpunkt größer gewählt und ein Rauschen auf die Sensoren gelegt wird.

Approximation im Reinforcement Learning Die Approximation der Q-Funktion wird im Reinforcement Learning immer unabdingbarer, da die wenigsten Kontrollaufgaben für einen realen Roboter sich in einem diskreten Raum beschreiben lassen. Eine Möglichkeit wäre den kontinuierlichen Zustands-Aktionsraum zu diskretisieren. Allerdings besteht durch eine schlechte Diskretisierung die Gefahr wichtige Zustände zu verstecken. Ein Zustand ist versteckt, wenn er nach der Diskretisierung gemeinsam mit anderen Zuständen repräsentiert wird, obwohl diese Zustände nicht äquivalent sind. Folglich kann durch eine Diskretisierung wichtige Information über das System verloren gehen und somit die optimale Q-Funktion nicht bestimmt werden [17]. Wenn die Diskretisierung den Raum zu genau abtastet, verliert das System die Fähigkeit zu Generalisieren [17]. Der Hauptgrund, der gegen eine Diskretisierung spricht ist, dass gerade in multidimensionalen Räumen die Anzahl der durch die Diskretisierung entstandenen Zustände exponentiell mit der Anzahl der Dimensionen ansteigt. Somit ist ab einer gewissen Größe des Lernproblems eine Diskretisierung nicht zu realisieren.
Geschichtlich bedingt sind alle Ansätze des Reinforcement Learnings auf einen diskreten Zustands-Aktionsraum ausgelegt, da bei der Entwicklung des Reinforcement Learnings über das Dynamische Programmieren die optimale Q-Funktion berechnet wurde (siehe 2.2.2.1). Selbst die berühmten Algorithmen wie das *Q-Learning* oder auch die *Temporal Difference* Methode (TD(λ)) arbeiten mit dem Verfahren der *Value Iteration*, welches das Dynamische Programmieren verwendet. Die Einführung eines kontinuierlichen Raumes ist somit eine maßgebliche Veränderung. Eine einfache Ersetzung der „Look-up Table“ durch einen Funktionsapproximator, in Kombination mit den bereits erwähnten Verfahren reicht dabei jedoch nicht aus. Es konnte gezeigt werden, dass mit diesem Ansatz selbst einfachste Aufgaben nicht gelöst werden konnten [2]. Zur Approximation der Q-Funktionen wurden unterschiedlichste Verfahren wie beispielsweise polynominale Regression [2], Neuronale

Netze mit Backpropagation-Verfahren [2], CMAC (Cerebellar model articulation controller) [11] oder Radiale Basisfunktionsnetze, wie in dieser Diplomarbeit,[11] und *local weighted* Regression [2], [17] verwendet. Zum Teil konnte bei diesen Verfahren gezeigt werden, dass die Value/Q-Funktion erfolgreich approximiert werden konnte. Allerdings ist die Konvergenz der Funktionsapproximation nicht garantiert und erfordert eine sehr genaue Einstellung der Parameter [2]. Die Ursache für diese Probleme lässt sich zurückführen auf die angewendete Methode. In den aufgeführten Anwendungen handelte es sich stets entweder um Q-Learning- oder TD(λ)-Methoden. Bei der Verwendung dieser Methoden lässt sich das schlechte Konvergenzverhalten auf die systematische Überschätzung der Rewardwerte zurückführen [22]. Nach den Ausführungen von S. Thrun & A. Schwartz [22] liegt der Grund für die systematische Überschätzung darin, dass in jedem Zeitschritt das Maximum der Q-Funktion berechnet wird. Da sowohl beim TD(λ) als auch bei Q-Learning die Schätzung des aktuellen Rewards von der Schätzung des letzten Rewards abhängt, werden durch die Verwendung des Maximums der Q-Funktion immer Rewardwerte bevorzugt, die einen größeren Funktionswert aufweisen. Aus diesem Grund akkumulieren sich über die Zeit Fehler, die durch die systematische Begünstigung höherer Rewardwerte entstehen. Dies hat zur Konsequenz, dass je nachdem, wie groß das Rauschen auf den vorliegenden Zustands-Aktionspaaren ist, die systematische Überschätzung dazu führen kann, dass das Lernverfahren nicht konvergiert.
In dem Lernansatz, der in dieser Diplomarbeit vorgestellt wurde, ist eine systematische Überschätzung der Rewardwerte ausgeschlossen, da in diesem Reinforcement Learning-Ansatz die Monte Carlo-Methode verwendet wird. Der Vorteil dieser Methode besteht darin, dass die Schätzung des Rewards eines Zustands-Aktionspaares nicht von der Schätzung des vorherigen Paares abhängt. Dadurch kann sich eine Fehlschätzung nicht auf die Schätzung des nächsten Paares auswirken und somit können auch keine systematischen Fehler entstehen. Zur genauen Untersuchung des Konvergenzverhaltens und auch der Fehleranalyse muss allerdings erst eine systematische Auswertung des aufgestellten Systems erfolgen.

7.2 Ausblick

Der in dieser Diplomarbeit vorgestellte Ansatz beschreibt eine Möglichkeit die Gewichtung lokaler Navigationsstrategien über Reinforcement Learning situations- und zeitabhängig zu lernen. Allerdings handelt es sich dabei um einen Grundentwurf dieses Ansatzes. Aus Zeitgründen konnten viele Verbesserungvorschläge und weitere interessante Fragestellungen nicht weiterverfolgt werden. Diese sollen abschließend kurz vorgestellt werden. Zur Approximation der Q-Funktion wird in dieser Arbeit ein RBF-Netzwerk verwendet. Während der Integration des RBFs in das Reinforcement Learning-Problem wurde schnell festgestellt, dass die Form der Rewardfunktion starke Auswirkungen auf das Laufzeitverhalten der Relaxation der Koeffizienten und die Maximumssuche hat. Aus diesem Grund wurde die geringe Bestrafung in jedem Zeitschritt, der noch nicht zum Ziel führte, eingeführt. Allerdings ist auch diese Hyperbelform der Rewardfunktion nicht optimal gewählt, da sie Sprungstellen in dem zu approximierenden Raum hervorruft. Eine andere

Möglichkeit bestünde darin keinen Bestrafungsterm zu verwenden und somit für jedes Zustands-Aktionspaar einer Sequenz den gleichen Reward zu vergeben. Hierzu müsste allerdings das bestehende RBF-Netzwerk durch die Einarbeitung eines Polynoms so verbessert werden, dass dieses System auch konstante Bereiche approximieren kann. Dies ist mit dem bestehenden System nicht möglich, da eine Linearkombination auf Gaußkurven keine konstanten Bereiche approximieren kann. Zudem müsste man auch die Maximumssuche erweitern, so dass konstante Abschnitte erkannt und nicht in die Maximumssuche aufgenommen würden. Allerdings wäre durch die veränderte Rewardfunktion nicht zwangsläufig ein besseres Konvergenzverhalten der Koeffizientensuche gewährleistet, da auch diese Boxform der Rewardfunkton Sprungstellen im Zustands-Aktionsraum verursachen würde. Aus diesem Grund sollten unterschiedliche Formen der Rewardfunktion analysiert werden, um eine effizientere Koeffizientenbestimmung zu gewährleisten.
Des weiteren konnten einige interessante Fragestellungen bezüglich des Lernverhaltens nicht bearbeitet werden. Eine besonders beim Reinforcement Learning sehr interessante Fragestellung wäre, ob das aufgestellte System in der Lage ist sich an eine veränderte Umgebung zu adaptieren. Die Frage hierbei ist, ob es länger dauert das System komplett neu lernen zu lassen oder das bestehende System an die veränderte Umgebung anzupassen. Zudem wäre es interessant, ob der Agent auch mit der ihm zur Verfügung gestellten Information in der Lage ist, zu lernen, wann es sinnvoll wäre einen Durchlauf abzubrechen. In dem jetzigen System ist dieses Abbruchkriterium durch das Zeitlimit fest vorgegeben.

Literaturverzeichnis

[1] Alpaydin, E.: *An Introduction to Machine Learning.* MIT Press, 2004

[2] Boyan, J. A. ; Moore, A. W.: Generalization in Reinforcement Learning: Safely Approximatiing the Value Function. In: *Advances in Neural Information Processing Systems: Proceedings of 1994 Conference , Cambridge MA: MIT Press* (1995), S. 369–376

[3] Braitenberg, V.: *Vehicles. Experiments in Synthetic Psychologie.* MIT Press, Cambrige, MA, 1984

[4] Brooks, R.: A robust layered control system for a mobile robot. In: *IEEE Journal of Robotics and Automation* 2(1) (1986)

[5] Cartwright, B.A. ; Collett, T.S: Landmark learning in bees. In: *Journel of Computational Physiology* A 151 (1983), S. 521–543

[6] Dudel ; Menzel ; Schmidt: *Neurowissenschaften - Vom Molekül zur Kognition.* Springer, 2001

[7] Franz, M. O. ; Mallot, H. A.: Biometric robot navigation. In: *Robotics and autonomous Systems* 30 (2000), S. 133–153

[8] Franz, M.O. ; Schölkopf, B. ; Mallot, H.A ; Bülthoff, H.H: Where did I take that snapshot? Scene-based homing by image matching. In: *Biological Cybernectics* 79 (1998), S. 191–202

[9] Gallistel, C.R.: *The organization of learning.* MIT Press, Cambrige, 1990

[10] Hübner, Wolfgang: *From Homing Behavior to Cognitive Mapping Integration of Egocentric Pose Relations and Allocentric Landmark Information in a Graph Model*, University of Tübingen Department of Cognitive Neuroscience Auf der Morgenstelle 28 72076 Tübingen, Germany, Diss., 2004

[11] Kretchmar, R. M. ; Anderson, C. W.: Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning. In: *Proceedings of the IEEE Int. Conf. on Neural Networks* (1997), S. 834–837

[12] Lane, T. ; Wilson, A.: Toward a Topological Theory of Relational Reinforcement Learning for Navigation Tasks. In: *Proceedings of the 18. Int. Florida Artifical Intelligence Research Society Conferenc (FLAIRS-2005)* (2005)

[13] Martìnez-Marín, T. ; Duckett, T.: Fast Reinforcement Learning for Vision-guided Mobile Robots. In: *Proceedings of IEEE Int. Conf. on Robotics and Automation* (2005)

[14] Millán, R. ; Torras, C.: Efficient Reinforcement Learning of Navigation Strategies in an Autonomous Robot. In: *Proceedings of IEEE Int. Conf. of Intelligent Robots and System* (1990), S. 15–22

[15] Orr, M.J.L.: Recent Advances in Radial Basis Function Networks. In: *Technical Report, Institute for Adaptive and Neural Computation, Division of Informatics, Edinburgh University, www.anc.ed.ac.uk/ mjo/paper/intro.ps* (1999)

[16] Poggio, T. ; Girosi, F.: A Theory of Networks for Approximation and Learning. In: *C.B.I.P. Paper* 31 (1989)

[17] Smart, W. D. ; Kaelbling, L. P.: Practical Reinforcement Learning in Continuous Spaces. In: *Proceedings of the 17. Int. Conf. on Machine Learing (ICML-2000)* (2000), S. 903–910

[18] Smart, W. D. ; Kaelbling, L. P.: Effective Reinforcement Learning for Mobille Robots. In: *Proceedingf of IEEE Int. Conf. on Robotics and Automation (ICRA)* Vol. 4 (2002), S. 3404–3410

[19] Stürzel, Wolfgang: *Sensorik und Bildverarbeitung für Landmarken-basierte Navigation*, Department of Cognitive Neuroscience Auf der Morgenstelle 28 72076 Tübingen, Germany, Diss., 2003

[20] Sutton, R. S. ; Barto, A. G.: *Reinforcement Learning*. Cambridge, Massachusetts, London England : MIT Press, 1998

[21] Terry, W. S.: *Learing and Memory*. Pearson Education, Inc, 2006

[22] Thrun, S. ; Schwartz, A.: Issues in Using Function Approximation for Reinforcement Learning. In: *Proceedings of the Fourth Connectionist Models Summer School* (1993)

[23] Wehner, R. ; Michel, B. ; Antonsen, P.: Visual navigation in insects: Coupling of egocentric and geocentric information. In: *Journal of Experimental Biology* 199 (1996), S. 129–140