Universität Bremen

Verbundvorhaben	VIRGO^4 **Vorhersagesysteme in reaktiven Gruppen autonomer Roboter**
Titel	**Abschlussbericht der Universität Bremen**
Förderkennzeichen	50RA1114
Zuwendungsempfänger	Universität Bremen, Fachbereich 03, AG Robotik Robert-Hooke-Straße 1, 28359 Bremen
Ausführende Stelle	Universität Bremen, Fachbereich 03, AG Robotik
Projektleiter	Prof. Dr. Frank Kirchner
Bewilligungszeitraum	01.04.2011 - 30.06.2014
Autoren	Elmar Berghöfer, Raúl Domínguez, Alexander Duda, David Feess, Malgorzata Goldhoorn, Dr. Sylvain Joyeux, Michaela Kording, Steffen Planthaber, Christian Rauch, Felix Rehrmann, Martin Schröer, Dr. Tim Tiedemann
Erstellungsdatum:	17. Dezember 2014

Gefördert durch:

Bundesministerium
für Wirtschaft
und Energie

aufgrund eines Beschlusses
des Deutschen Bundestages

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Wirtschaft und Energie unter dem Förderkennzeichen 50RA1114 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.
Gefördert von der Raumfahrt-Agentur des Deutschen Zentrums für Luft- und Raumfahrt e.V. mit Mitteln des Bundesministeriums für Wirtschaft und Energie aufgrund eines Beschlusses des Deutschen Bundestages unter dem Förderkennzeichen 50RA1114.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Das vorliegende Dokument stellt den Abschlussbericht des Vorhabens VirGo^4 dar. VirGo^4 wurde durch das Bundesministerium für Wirtschaft und Energie (BMWi, Förderkennzeichen 50RA1114) gefördert.

Dieser Abschlussbericht folgt dabei dem in der Anlage 2 der BNBest-BMBF 98 gegebenen Muster. In Kapitel 1 wird eine kurze Darstellung des Projektes gemäß Punkt I gegeben, Kapitel 2 stellt die inhaltlichen Entwicklungen in einer eingehenden Darstellung gemäß Punkt II des Musters dar. Die Punkte III (Erfolgskontrollbericht) und IV (Berichtsblatt bzw. Document Control Sheet) werden durch gesondert abgegebene Dokumente erfüllt.

Die dargestellten Arbeiten umfassen alle von der Universität Bremen geleisteten Arbeiten im Vorhaben VirGo^4. Abbildung 1.1 auf Seite 14 gibt den Überblick über die Aufteilung der Arbeitspakete. In diesem Bericht werden die Arbeitspakete der AG Robotik der Universität Bremen vorgestellt.

1 Kurzdarstellung nach BNBest-BMBF 98 3.2 I

In diesem Kapitel wird zunächst die Aufgabenstellung des Projektes beschrieben (Abschnitt 1.1). Anschließend werden die Voraussetzungen des Vorhabens (Abschnitt 1.2) sowie Planung und Ablauf des Vorhabens (Abschnitt 1.3) behandelt. Es folgen Erläuterungen zum wissenschaftlichen Stand (Abschnitt 1.4) und zur Zusammenarbeit mit anderen Stellen (Abschnitt 1.5).

1.1 Aufgabenstellung

Um die Zuverlässigkeit von Software für Robotersysteme zu erhöhen verfolgte VirGo[4] unter anderem folgendes Ziel:

Eine Architektur zur Verhaltenssteuerung autonomer Roboter wurde entwickelt, die die Steuerung von Robotern in zukünftigen Weltraummissionen erleichtern und robuster gestalten kann. Ein Teil beschäftigt sich mit der gelernten Vorhersage der Auswirkungen von Aktionen eines Roboters auf seine Umwelt und den Zustand des Systems selbst. Ein Roboter kann diese Information verwenden, um die Güte seiner Aktionen zu bewerten und sein Verhalten, falls notwendig, zu adaptieren. Es lassen sich hiermit auch Abschätzungen über Auswirkungen einer Entscheidung treffen, die zur frühzeitigen Erkennung von Fehlverhalten bei autonomen Entscheidungen dienen könnten.

1.1.1 Wissenschaftliche und technische Arbeitsziele

Autonome mobile Roboter sind ein aktives Forschungsgebiet in der Robotik. Für den Einsatz in sicherheitskritischen Anwendungen waren die bei Beginn des Vorhabens existierenden Lösungen jedoch meist nicht geeignet. Der Grund dafür war das Fehlen einer vorhersehbaren,

robusten Strategie zur autonomen Steuerung von Robotersystemen, um den Ablauf einer Mission möglichst zuverlässig vorausplanen zu können und das System in die Lage zu versetzen, auf unerwarteter Störungen reagieren zu können. Beispielhaft soll hier die Interaktion mit Gebieten angeführt werden, die für Menschen schwer zugänglich oder gefährlich sind. Da autonomes Handeln die Effektivität eines Robotersystems verbessern kann, entwickelte VirGo^4 Ansätze für verlässliches, autonomes Verhalten und Mechanismen zur Überwachung, Vorhersage und Bewertung der Entscheidungen autonomer mobiler Roboter, die einzeln und im Team agieren. Dies trägt dazu bei, einige der Unwägbarkeiten autonomer Entscheidungsfindung zu kompensieren.

Im Folgenden werden die für die AG Robotik der Universität Bremen relevanten einzelnen Arbeitsziele definiert. Zu der im Laufe des Vorhabens erfolgten Bearbeitung dieser Ziele siehe das zusammenfassende Kapitel 3.

Verhaltenssteuerung

Ein wesentliches Ziel von VirGo^4 war es, einen konkreten Architekturentwurf auf Basis des LoB-Modells zu entwickeln, der Komponenten zur internen Selbsteinschätzung eines Systems und zur Vorhersage des System- und Umweltzustands enthält. Das im Rahmen des Vorhabens zu entwickelnde Vorhersagesystem sollte es ermöglichen, Auswirkungen autonom geplanter Aktionen abschätzen zu können und sollte zur Überwachung und Bewertung des Systemzustands herangezogen werden. Technisch stand die Realisierung des Architekturentwurfs im Vordergrund. Daraus abgeleitet stellte sich die Frage, inwiefern eine Prädiktion zur Zustandsschätzung sinnvoll ist und ob sich dies mit der Erkenntnis bei biologischen Systemen vergleichen lässt. Insgesamt bot sich die Verfolgung eines biologisch- bzw. kognitionspsychologisch-inspirierten Verfahrens zur Verhaltensauswahl und Verhaltensevaluation an.

Vorhersagesystem

Zentraler Bestandteil der VirGo^4-Architektur ist eine Komponente zur Vorhersage von Umweltparametern bzw. der Auswirkungen von Aktionen. Die Anforderungen wurden beim Entwurf des Konzepts zur Verhaltenssteuerung formuliert. Auswirkungen von Aktionen könnten durch Simulation auf Basis eines internen Modells prädiziert werden. Da einem Robotersystem meist jedoch nur beschränkte Ressourcen zur Verfügung stehen, versuchte VirGo^4 einen

geeigneten Kompromiss zwischen Effizienz und Genauigkeit zu finden, wobei der Echtzeitfähigkeit eine höhere Priorität zugeordnet wurde. Entscheidend ist hier auch der Rückgriff auf biologisch-motivierte Verfahren, die eine besonders effiziente Problemlösung vermuten ließen.

1.2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Die Forschung der Arbeitsgruppe Robotik der Universilät Bremen beschäftigt sich mit der Entwicklung von intelligenten Agenten, die durch Interaktion mit der Umwelt lernen und in der Lage sind, Erkenntnisse über ihre Umgebung zu sammeln sowie Handlungsoptionen selbständig zu identifizieren. Ein Kernaspekt für die Durchführung extra-terrestrischer Missionen sind Robotersysteme, die autonom auf unvorhergesehene Ereignisse reagieren können. Der Fokus des Verbundvorhabens VirGo^4 liegt hier auf der Umsetzung eines Frameworks zur verhaltensbasierten autonomen Steuerung von Robotersystemen.

Zu den bisherigen Arbeiten und Kernkompetenzen der AG Robotik zählen die folgenden, im Rahmen von VirGo^4 eingesetzten, Punkte:

- Entwicklung einer adaptionsfähigen und robusten Lernarchitektur, die aus einer reaktiven und einer höheren deiberativen Schicht besteht.
- Entwicklung von Lernverfahren im Bereich Bestärkendes Lernen und Test derselben auf robotischen Kontrollproblemen.
- Forschung im Bereich evolutionärer Algorithmen, insbesondere im Bereich Neuroevolution.
- Entwicklung von modularen Software-Frameworks zur Implementierung und zur systematischen Untersuchung von Lernverfahren.

Diese Erfahrungen und Kompetenzen der AG Robotik waren insbesondere die Voraussetzungen für das in VirGo^4 implementierte Levels-of-Behaviour-Modell und die untersuchten Vorhersageverfahren im Rahmen der entwickelten Selbstevaluation.

1.3 Planung und Ablauf des Vorhabens

Die Arbeitsplanung des Verbundvorhabens VirGo[4] gliederte sich in fünf Arbeitspaketgruppen (siehe Abbildung 1.1). Der Schwerpunkt der AG Robotik lag auf der AP-Gruppe 4x00. In dem AP 4100 bzw. AP 4300 wurden das Levels-of-Behaviour-Modell und die Vorhersage und Selbstbewertung spezifiziert. Die Arbeitspakete 4200 und 4400 deckten die Implementierungen dieser Komponenten ab.

Die Arbeitspakete 2200 bis 2500 und 3x00 wurden von dem Partner DFKI bearbeitet (siehe hierzu dessen Abschlussbericht). In der AP-Gruppe 5x00 realisierten beide Partner gemeinsam die Demonstration der Vorhabensergebnisse. AP 2100 beinhaltet die Darstellung des Stands der Technik und wurde von beiden Projektpartnern erarbeitet.

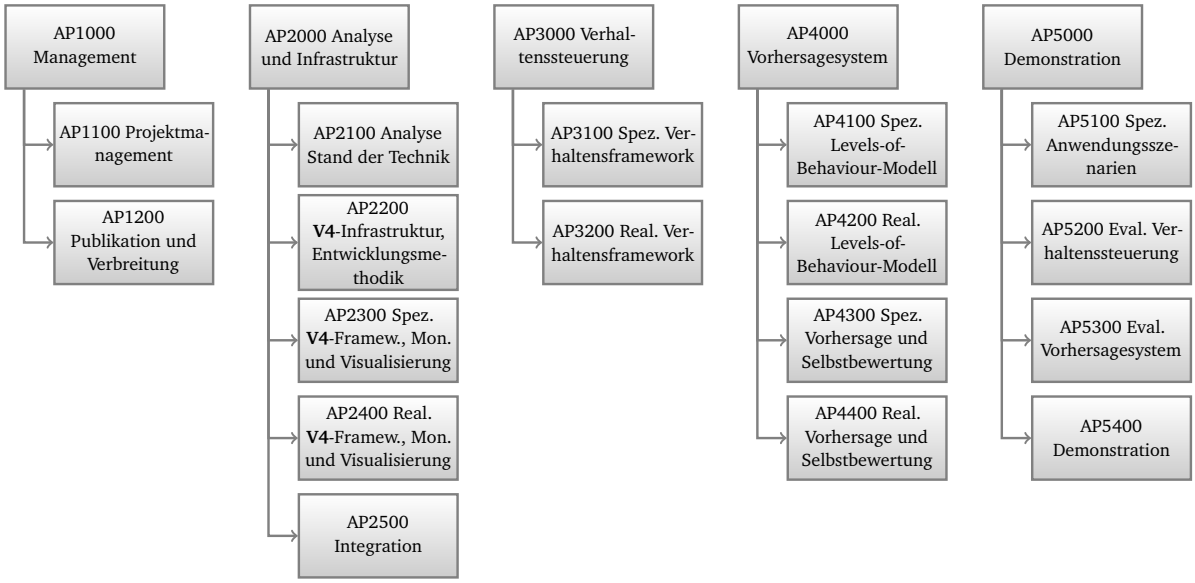


Abbildung 1.1 Diagramm zur Work-Breakdown-Structure (WBS) des Verbundvorhabens VirGo[4]

Zur Kompensation von unvorhergesehenem technischen Mehraufwand, der unter anderem zu einem erhöhtem Zeitaufwand bei der Integration führte, wurde eine kostenneutrale Verlängerung des Verbundvorhabens bis zum 30. Juni 2014 beantragt. Diese Änderung wurde vom Zuwendungsgeber genehmigt. Der angepasste Projektplan ist in Abbildung 1.2 dargestellt.

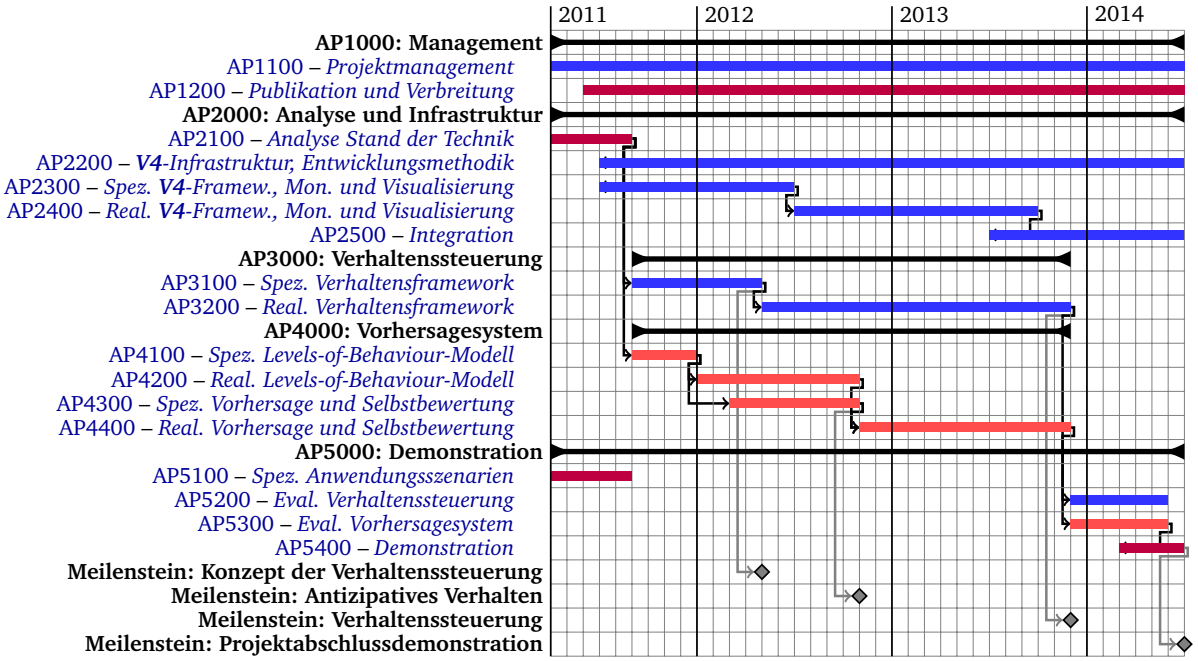


Abbildung 1.2 Gantt-Chart des Verbundvorhabens VirGo[4] in der finalen Version

1.4 Wissenschaftlicher und technischer Stand zu Beginn des Vorhabens

Das zu realisierende Levels-of-Behaviour-Modell zählt zu der Klasse der Schichten-Architekturen. Schichten-Architekturen sind weit verbreitet, meistens mit drei, manchmal auch mit zwei Schichten. Der zweite Schwerpunkt der Arbeiten der AG Robotik bildet die Vorhersage und Selbstevaluation eines robotischen Systems. Im Folgenden wird ein exemplarischer Überblick gegeben über den bei Vorhabensbeginn aktuellen Stand der Technik bei robotischen Selbstevaluations- sowie Vorhersageverfahren und bei mehrschichtigen Verhaltenssteuerungen für Roboter. Details zum Stand der Technik sind auch der Beschreibung der Ergebnisse von AP 2100 zu entnehmen.

Vorhersage und Selbstbewertung Bei der Kombination von Vorhersage und Selbstbewertung eines Systems kann die Vorhersage die Basis für die Selbstbewertung bilden: Existiert eine Vorhersage, so kann im Vergleich mit aktuellen Messdaten der eigene Zustand von dem System bewertet werden. Die Vorhersage eines Systems ist wiederum

mit Methoden des maschinellen Lernens gekoppelt, da erst durch die Erfahrung (Lernen) die Abschätzung von Konsequenzen ermöglicht wird. Verfahren zu einer einfachen Vorhersage von Positionen durch die Abschätzung von Richtungsvektoren wurden bereits vorgestellt [16]. Fortführende Arbeiten nutzen biologische Daten oder Verhaltensexperimente mit Menschen [26], um beispielsweise Bewegungen auf ein Ziel vorauszusehen und Hindernissen auszuweichen. Und aus der Kognitionswissenschaft stammen Modelle zur Wahrnehmung von Bewegung, wie in [82] vorgestellt. Die robotische Integration und Demonstration ist dabei allerdings in der Regel sehr beschränkt und z.T. ausschließlich auf Simulationen gestützt. In [67] wird die Wahrnehmung des Raums zum Greifen von Objekten zum Gegenstand von Vorhersage und Evaluation genommen. Die Abschätzung der Konsequenzen einer Aktion findet dort über die parallele Simulation von Greifmustern und die anschließende Evaluierung der zu erwartenden Resultate statt.

Strukturmodelle Das Functional Reference Model der ESA unterteilt die Software-Architektur eines Robotersystems in drei Schichten. Die unterste Schicht (Mission Layer) interagiert direkt mit den Hardwarekomponenten. Die mittlere Schicht (Task Layer) stellt grundlegende Verhaltensweisen bereit, um es der obersten Schicht (Action Layer) zu ermöglichen, die Planung und Definition von abstrakten Zielen vorzunehmen. Jede Schicht plant eigenständig und führt die benötigten Funktionen aus.

Weitergehende Ansätze *ESA-Projekte* Die ESA hat verschiedene Projekte initiiert, die jeweils einzelne Teilgebiete der autonomen Entscheidungsfindung untersuchen. Diese sind teilweise ebenfalls relevant für die Arbeiten der AG Robotik. VIMANCO (Vision Manipulation of Non-Cooperative Objects) ist ein lernfähiges System zur Erkennung und Verfolgung von dreidimensionalen Objekt in Bildern. Bei HiPeRCAR (High Performance Resilient Computer for Autonomous Robotics) handelt es sich um ein redundant ausgelegtes Computersystem für die Datenverarbeitung, das automatisch auf Fehlersituationen reagieren kann. Als Betriebssystem steht mit xLuna ein echtzeitfähiger, Posix-kompatibler Systemkern zur Verfügung, der auch benutzerfreundlichere Systeme wie RTAI Linux verwalten kann. *Curiosity Cloning* Ein weiteres Forschungsprojekt der ESA ist Curiosity Cloning – Neural Modelling for Image Analysis –, das untersucht, ob fallspezifische Neugier eines Menschen kopiert und auf einen Roboter übertragen werden kann.

Zu Details siehe auch die eingehende Darstellung, insbesondere zu Arbeitspaket 2100.

1.4.1 Eigene Vorarbeiten

Die Universität Bremen ist mit 290 Professuren und 19.000 Studierenden eine Universität mittlerer Größe mit breitem Fächerspektrum. Sie bietet mehr als 100 Studiengänge in rund 30 wissenschaftlichen Disziplinen an und wurde im Sommer 2012 zur Exzellenz-Universität gekürt. Die Arbeitsgruppe Robotik (AGR) der Universität Bremen untersucht und entwickelt Methoden der künstlichen Intelligenz und ist besonders in den Bereichen maschinelles Lernen und Optimierungsverfahren aktiv. Die Arbeitsgruppe beschäftigt sich dabei insbesondere mit der Entwicklung von intelligenten Agenten, die durch Interaktion mit der Umwelt lernen und in der Lage sind, Erkenntnisse über ihre Umgebung zu sammeln sowie Handlungsoptionen selbständig zu identifizieren. Die am Institut vorhandenen Kompetenzen waren eine ideale Basis für die Entwicklung des Vorhersagesystems als Teil des LoB-Modells.

IMMI Ziel des IMMI Projektes ist es, ein Brain Reading System zu schaffen, das mit Hilfe von Signalverarbeitungsmethoden und maschinellen Lernverfahren eine adaptive passive Beaufsichtigung eines Operators realisiert. Hierbei werden spezifische Änderungen von Elektroenzephalographie(EEG)-Signalen detektiert und mit Hilfe maschineller Lernverfahren analysiert. Diese Analyse kann auf verschiedene Arten genutzt werden. Beispielsweise wird die Intention eines Operators detektiert, eine Bewegung auszuführen, was direkt für eine Bewegungsvorhersage genutzt werden kann. Weiterhin wurde in dem Projekt eine Bewegungsvorhersage basierend auf gemessener Muskelaktivität realisiert und mit der EEG-Bewegungsvorhersage kombiniert und Konzepte zur multimodalen Unterstützung vorgestellt.

1.4.2 Fachliteratur und verwendete Dokumentationsdienste

Die verwendete Fachliteratur ist im Literaturverzeichnis ab Seite 84 angegeben.

1.5 Zusammenarbeit mit anderen Stellen

Das Verbundvorhaben VirGo[4] wurde gemeinsam von dem Robotics Innovation Center des DFKI und der AG Robotik der Universität Bremen durchgeführt. Darüber hinaus gab es im Rahmen des Vorhabens keine weiteren Kooperationen.

2 Eingehende Darstellung nach BNBest-BMBF 98 3.2 II

Die folgenden Abschnitte beschreiben eingehend die vorgenommenen Arbeiten und Untersuchungen im Vorhaben VirGo[4] sowie deren Ergebnisse.

2.1 Verwendung der Zuwendung

2.1.1 AP 2100: Analyse Stand der Technik

An den Arbeiten in diesem AP waren beide Partner AG Robotik und das DFKI RIC gemeinsam beteiligt. In diesem Bericht werden nur die Arbeiten der AG Robotik beschrieben. Zu den Arbeiten des Partners DFKI siehe dessen eigenen Abschlussbericht.

Verhaltenssteuerung

Ein Hauptarbeitspunkt zur Realisierung der Autonomie im Projekt VirGo[4] ist die Implementierung von biologisch motivierten Verhaltensmodellen. Dabei wird auf die bereits im Proposal erwähnte 3-Schichten-Architektur „Levels-of-Behavior“ (**Abb. 2.1**) zurückgegriffen.

Das allgemeine Modell einer verhaltensbasierten Architektur ist in **Abb. 2.2** dargestellt. Zu sehen ist die Parallelisierung verschiedener Verhalten (*Behavior 1 ... n*), wodurch eine sehr flexible Steuerung von Robotersystemen ermöglicht wird. Es hat sich gezeigt, dass sich diese verhaltensbasierenden Systeme selbst an die sich schnell ändernden Bedingungen in der realen Welt anpassen können. Michaud beschreibt in [45] die EMIB-Architektur (*Emotion and*

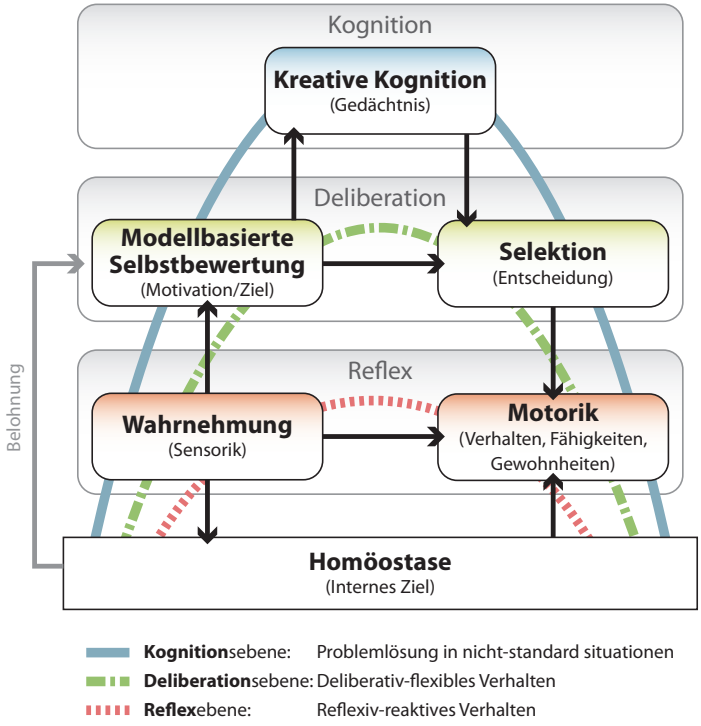


Abbildung 2.1 Levels-of-Behavior Modell

Motivation for Intentional selection and configuration of Behaviour-producing modules) und gibt eine kurze Übersicht der drei Modelle *Hierarchical decomposition, Functional decomposition* und *Behaviour-based decomposition,* welche jeweils die Modularisierung von Funktionen beschreiben, wie sie in **Abb. 2.2** für *Behavior-based* Modelle zu sehen ist. Zusätzlich wird ansatzweise auf ein 3-Schichtenmodell (*LoB, ESA-Referenzmodell*) eingegangen. Um diese extreme Parallelisierung zu ermöglichen, unterliegen diese Module den Bedingungen: (1) Ein Modul sollte einen sehr begrenzten Funktionsumfang haben, Module sollten demnach so fein granuliert werden wie möglich, (2) Module dürfen keine festen Abhängigkeiten zueinander haben oder exklusive Ressourcen nutzen. Erst dadurch wird die parallele Ausführung und eine höhere Abstraktion möglich, mit der auch high-level Aktionen durchgeführt werden können. Dabei übernehmen Module Aufgaben wie Sensordatenverarbeitung oder Planung, haben aber gleichzeitig Zugriff auf Sensoren und Aktoren eines Systems (siehe *Stimuli, Action* in **Abb. 2.2**).

Dabei spielen die Aspekte *Adaption* der Behavior und *Auswahl* aus dem Behavior-Pool eine entscheidende Rolle. Verfahren, welche Behavior anhand der Intention eines Systems anpassen, sind in [45, 46] beschrieben. Die Intention des Systems ergibt sich dabei über

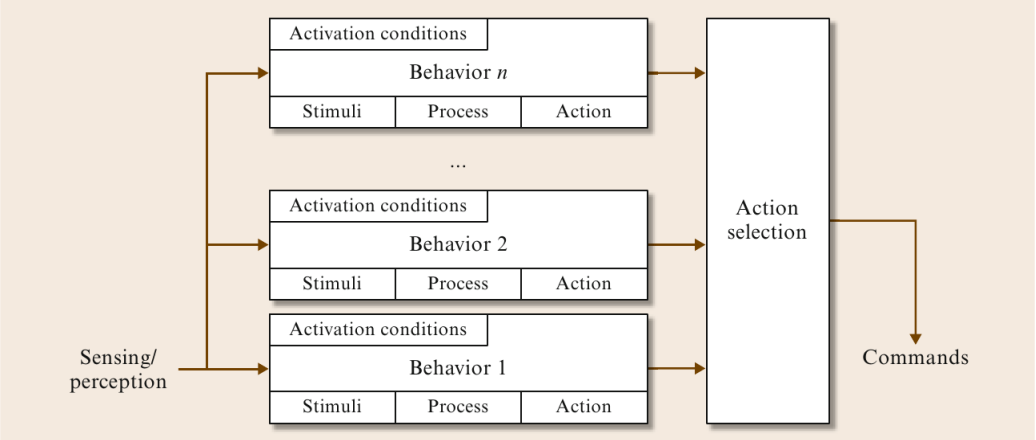


Abbildung 2.2 Allgemeines Modell eines Behavior-Based System ([72], Seite 895, Fig. 38.1)

Schlussfolgerungen, dem Wissen sowie dem Weltbild allgemein.

Das Modell von Norman & Shallice [50] aus dem Jahre 1986 beschreibt aus psychologischer Sicht, wie Aktionen ausgewählt und ausgeführt werden. Der Grundgedanke ist dabei die Unterscheidung von reaktiven und gewollten Handlungen. Diese Unterscheidung ist notwendig, da der Mensch nicht alle Aktionen mit dem gleichen kognitiven Aufwand ausführen kann. Reaktive ungewollte Aktionen (Reflexe, Gewohnheiten) werden wesentlich schneller ausgeführt und benötigen eine geringere kognitive Leistung. Aktive und gewollte Handlungen hingegen benötigen eine gewisse kognitive Verarbeitung, um entsprechend auf neue Situationen und Gegebenheiten reagieren zu können. Dabei können gewollt hervorgerufene Handlungen in automatischen Reaktionen übergehen, wenn diese ausreichend oft hervorgerufen werden. Ein weiterer Punkt des Modells von Norman & Shallice ist das sogenannte *Supervisory Attentional System* (SAS), welches die Aufmerksamkeit des Handelnden auf eine Aktion lenkt, um diese mit entsprechender Sorgfalt ausführen zu können. Dies ist vor allem dann nötig, wenn viele Störeinflüsse wahrgenommen werden und keine ausreichende kognitive Kapazität vorhanden ist, um alle Handlungen parallel auszuführen. Analog zu einem Prozess-Scheduler ist das *SAS* dafür zuständig, die Ausführung von Handlungen zu koordinieren und priorisierten Aktionen entsprechende Ressourcen (Sensorik, Motorik, kognitive Leistung) bereitzustellen. Auf diesem Modell aufbauende Arbeiten gehen speziell auf die Implementierung des SAS für Roboter [20] und autonome Fahrzeuge [24] ein. Garforth's Umsetzung [20] behandelt insbesondere, wie reaktive Verhalten durch ausreichend häufiges Auftreten oder mit erhöhter Aufmerksamkeit erlernt werden können und wie diese reaktiven Handlungen zeitweise durch gewollte Handlungen unterdrückt werden können.

Agenten wählen ausführbare Aktionen aus einem pool vorhandener Verhalten (Behavior) aus. Dabei hat sich der Gebrauch von *Fuzzy-Logik* für die Empfehlung einer Aktion nach [44], [61] und [64] für die robuste und autonome Navigation von Robotern bewährt, da eine Anpassung von weich kodierten Befehlen wie „scharf rechts abbiegen“ an Umgebungsänderungen einfacher umsetzbar ist als eine harte Kodierung (z.B. „90° rechts abbiegen“). Ein großer Vorteil der Fuzzy-Logik ist, dass sowohl numerische als auch symbolische Eingaben verarbeitet werden können, während sich viele andere Verfahren auf entweder numerische oder symbolische Inputs beschränken. Auch sind nach Anwendung von Fuzzy-Logik entstandene Regeln zur Verhaltenssteuerung abstrakt und selbst bei komplexen Handlungsabläufen leicht verständlich, da sie einer einfachen hierarchischen Struktur folgen:

IF *path-condition* AND *obstacle-condition* THEN
command ,

Leider birgt Fuzzy-Logik den Nachteil, dass alle Regeln im Vorfeld definiert werden müssen. Lösungsansätze hierzu sind im Abschnitt „Kombination von Verhaltenssteuerung und Vorhersagemodell“ (siehe Anhang) zu finden.
Ein weiteres, auf Vorhersageregeln basierendes Verfahren zum autonomen Lernen in unbekannter Umgebung stellt das in [54] vorgestellte Surprise-Based Learning dar. Lernen findet dabei statt, wenn die Vorhersage nicht mit dem Systemzustand nach Ausführen der Aktion übereinstimmt.
Die Auswahl und die Konfiguration der Behavior, und damit das Adaptieren des Verhaltens an die Umgebung, kann durch Lernen der Verhaltensweisen erheblich verbessert werden. Die hierzu könnten die beiden Ansätze *Reinforcement Learning (RL)* und *Learning from Demonstration (LfD)* zum Einsatz kommen. *LfD* benötigt einen Beobachter, der die Aktionen zunächst vorführt. Systeme auf *RL*-Basis könnten durch Erfahrung (vergangene Aktionen und Resultate) die positiveren Behavior und Aktionen auswählen. Eine Übersicht aktueller Reinforcement Learning und *Iterative Learning Control* Modelle finden sich in [10] und [6]. Zu beachten ist, dass das Erlernen von richtigem Verhalten durch Demonstration sehr umfangreich und zeitaufwendig ist. Ein Vergleich von Reinforcement Learning und *Genetic Algorithms (GA)* findet sich in [78]. Hierbei wird auch auf *Q-Learning* eingegangen. Des Weiteren existieren Arbeiten zur Reduzierung der Q-States im Q-Learning [29].

Im Bereich der *Multi-Agenten-Systeme* (MAS) existieren weitere Arbeiten, welche sich speziell mit dem Lernen und der Auswahl der Behavior mit Einschränkung durch das Gruppenziel befassen. Der Informationsaustausch zwischen den Systemen kann dabei direkt auf einer

niedrigeren Ebene erfolgen, beispielsweise durch den direkten Austausch der Sensordaten [52], oder auf einer höheren Ebene über die Bekanntgabe der eigenen Fähigkeiten des Systems [80, 79, 28].

Die Entwicklung eines solchen Multi-Agenten-Modells unterliegt dabei zusätzlichen Design-Entscheidungen nach dem *Top-Down*- oder *Buttom-Up*-Prinzip [12] und nach einem *zentralisierten* oder *dezentralisierten* Ansatz [73]. Die Vor- und Nachteile der *Top-Down*- und *Buttom-Up*-Architektur werden in [12] erläutert, wohingegen Arbeit [73] das bekannte 3-Schichten-Modell um eine direkte Interaktion von Robotersystemen auf jeder der drei Schichten erweitert. Diese vorgeschlagenen Erweiterung ist in **Abbildung 2.5** dargestellt.

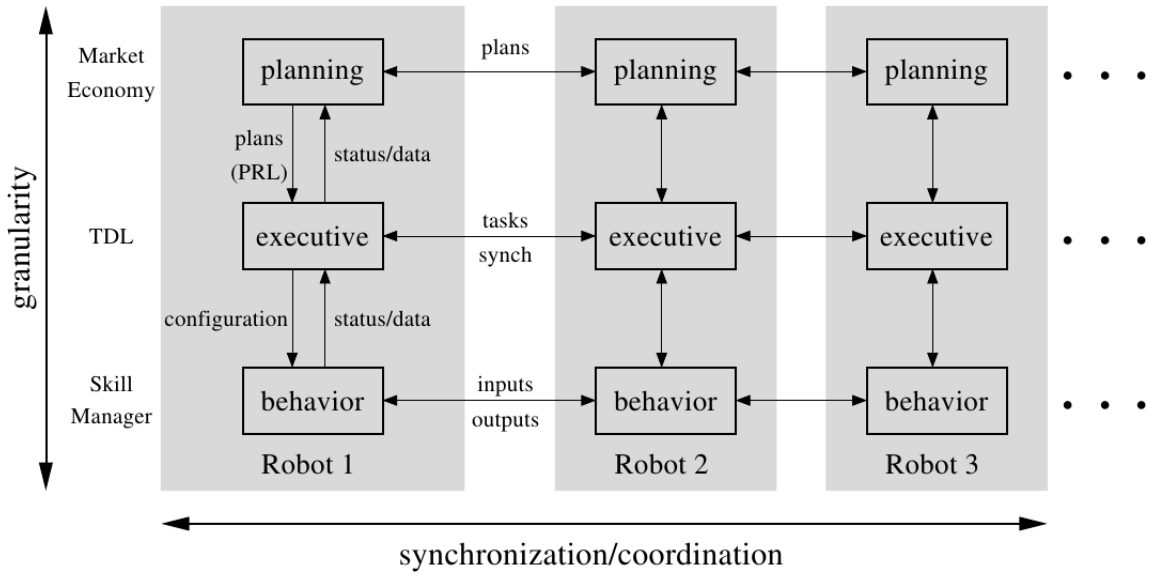


Abbildung 2.3 Erweiterung der 3-Schichten-Architektur nach Simmons, um Kommunikation von Robotersystemen auf unterschiedlichen Ebenen zu ermöglichen. ([73], Seite 4, Figure 1)

Ausgehend von dem verteilten Wissen kann ein System dann Entscheidungen treffen, welche Auswirkungen auf die Situation des Einzelnen als auch auf die Gesamtsituation der Robotersysteme hat. Beispiele dafür sind Systeme, die Aktionen basierend auf der Maximierung des kollektiven Wissens auswählen [75] und stark biologisch motivierte Ansätze, wie interagierende biologische Systeme [56].

Obwohl Lernen in verhaltensbasierenden Systemen aus zwei Richtungen zu betrachten ist (der des Einzelnen und der des Kollektivs), sind die genannten Ansätze zur Adaption von korrektem

Verhalten nicht auf eine der Richtungen beschränkt, sondern allgemein gültig anzuwenden.

Vorhersagemodell

Ein weiterer wichtiger Schritt zur Förderung der Autonomie verhaltensbasierter Systeme ist die Selbsteinschätzung und Vorhersage der Auswirkungen von Aktionen. Die Wahrnehmung der Umgebung sowie das Wissen über Konsequenzen einer Aktion ermöglichen erst die Auswahl eines Behavior, welches den größten Nutzen für ein System bietet. Dabei sind Vorhersage und Lernen eines Systems sehr stark miteinander gekoppelt, da erst durch die Erfahrung die Abschätzung von Konsequenzen ermöglicht wird. Verfahren zur Vorhersage von Positionen durch die Abschätzung von Richtungsvektoren existieren [16] und wurden bereits durch eine autonome Kollisionsvermeidung erweitert [51]. Fortführende Arbeiten nutzen biologische Daten von Tieren und zusätzliche Verhaltensexperimente mit Menschen [26], um Bewegungen auf ein Ziel vorauszusehen und Hindernissen auszuweichen. Aus der Kognitionswissenschaft stammende Modelle zur Wahrnehmung von Bewegung finden sich in [82] und [8].

Um speziell die Position eines mobilen Systems in der realen Welt vorherzusehen, ist die Wahrnehmung der Umgebung, als auch die Wahrnehmung von Bewegung notwendig. Die Wahrnehmung des Raums (*Space Perception*) ist häufig Gegenstand der Forschung zum Greifen von Objekten [66, 65]. Dabei findet die Abschätzung der Konsequenzen einer Aktion über die parallele Simulation von Greifmustern und die anschließende Evaluierung der zu erwartenden Resultate statt [67]. *Mental Imagery* ist ein Vertreter solcher künstlich erzeugten internen Sensordaten. Eine Anwendung dafür findet sich in der Selbsterkenntnis (*self-awareness*) von Systemen durch die Wahrnehmung der Umgebung und der mentalen Simulation von Bewegungsintentionen [30]. Diese Simulationen führen zu visuellen Ansichten, in denen sich der Agent selbst in einer möglichen zukünftigen Umgebung wiederfindet. Eine Methode zur Wahrnehmung der Umgebung und des eigenen Körpers eines Robotersystems über die Motorsensoren ist in [62] beschrieben.

Ausgehend von der Erkenntnis der eigenen Position in Raum und Zeit und der Abschätzung von zukünftigen Zuständen, kann daraufhin die ausgewählte und durchgeführte Aktion wahrgenommen werden. Diese als *Motion Perception* bezeichnete Fähigkeit von Systemen basiert oftmals auf dem optischen Fluss (*Optical Flow*) eines visuellen Systems [33, 15, 7, 57]. Durch die Analyse des optischen Flusses und der Kombination mit weiteren Daten bewegungswahrnehmender Sensoren, kann dann im späteren Verlauf eine Geschwindigkeits- und

Richtungsabschätzung rein über den optischen Fluss durchgeführt werden.

Lernen von Verhalten

Im Anhang werden verschiedene Verfahren zum Lernen von Verhalten vorgestellt. Die vorgestellten Methoden sollen einen Überblick darüber geben, wie verschiedene Algorithmen in mobiler Robotik angewandt werden. Statistische Verfahren wie Naive Bayes scheinen für den Bereich der Sensor Fusion und Objektklassifizierung gut geeignet, Hidden Markov Model zur Objektklassifizierung mit Hindernisumfahrung und Landkartenerstellung und Kalman Filter sowie Monte-Carlo Localization für die Lokalisierung eines Agenten. Neuronale Netze wurden vermehrt zur Balancierung angewandt, konnten erfolgreich mit Ausrutschen oder mit fehlenden Daten zur Lokalisierung umgehen (und können damit Kalman Filter und Monte-Carlo Localization unnötig machen), und passen sich sehr gut an dynamische Umgebungen an. Genetische Algorithmen erweisen sich vor allem in der Kombination mit anderen Verfahren als sinnvoll und die Einführung eines critic Elements des Reinforcement Learning verbessert einzelne Verfahren.

2.1.2 AP 4100: Spez. Levels-of-Behaviour-Modell

Referenzmodelle Bei der Auswahl eines Referenzmodells steht einerseits der biologisch motivierte Ansatz im Vordergrund, andererseits muss die technische Realisierbarkeit gewährleistet sein. Die vorgestellten technischen Ansätze des *ESA Functional Reference Models* (Abbildung 2.4) und das Kommunikationsmodell nach Simmons (Abbildung 2.5) gleichen dem ursprünglichen drei-schichtigen hierarchischen Levels-of-Behaviour Modell (Abbildung 2.5). Die Sensorik und Motorik eines Systems ist dabei hinzu tieferen Schichten stärker miteinander verknüpft.

In neueren neurobiologischen Überlegungen und Untersuchungen wird vermutet, dass Aktionen im menschlichen Gehirn mit unterschiedlichem Aufwand und Aufmerksamkeit ausgeführt werden (Schneider und Shiffrin [68], Norman und Shallice [50], Garforth [20]). Dabei reicht das Spektrum an Handlungen von reflexartigen Aktionen, welche sehr schnell auf Sensoränderungen reagieren können da sie ohne höhere kognitive Prozesse auskommen, bis zu kontrollierten Aktionen, welche durch ausreichend Aufwand auf neue unbekannte Situationen angewendet werden können. Garforth wählt das Norman & Shallice Modell als Grundlage für

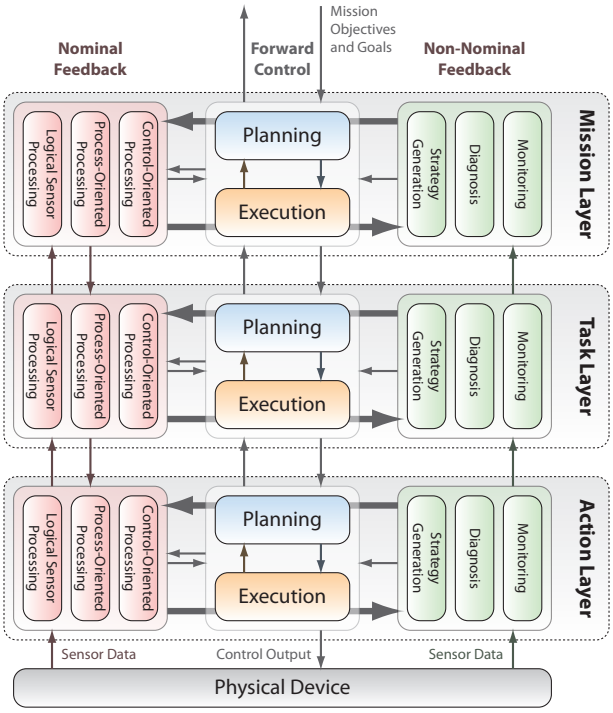


Abbildung 2.4 ESA Functional Reference Model

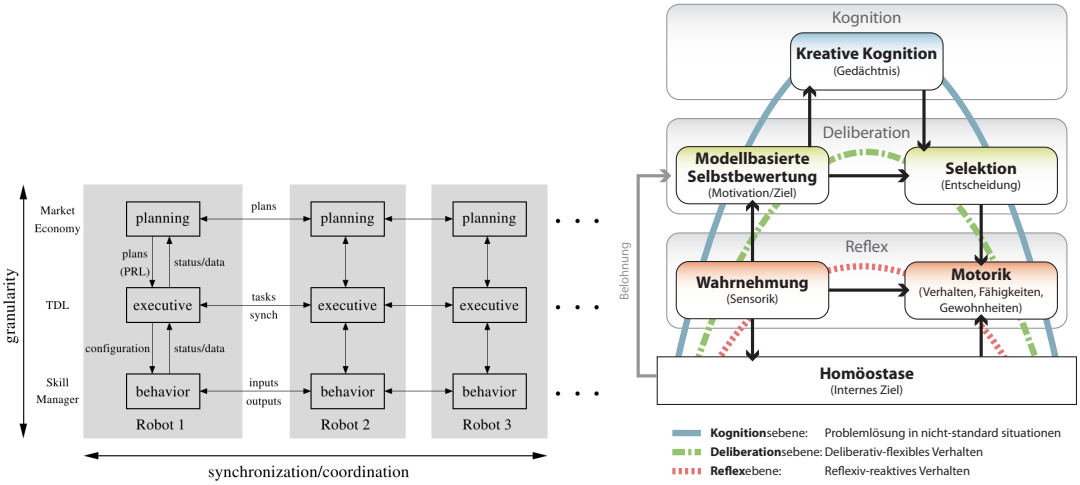


Abbildung 2.5 Links: Erweiterung der 3-Schichten-Architektur nach Simmons, um Kommunikation von Robotersystemen auf unterschiedlichen Ebenen zu ermöglichen ([73], Seite 4, Figure 1). Rechts: Levels-of-Behavior Modell.

seine Arbeit, da es seiner Meinung nach im Vergleich zu anderen Modellen die Auswahl von Aktionen kontrolliert und die Architektur des Modells stärker und klarer aufgeteilt ist.

Norman & Shallice (1986) Die Struktur des Modells nach Norman und Shallice ist in Abbildung 2.6 dargestellt. Das Modell beschreibt einen Zyklus von Sensorwahrnehmung und -verarbeitung, gefolgt von der Aktionsauswahl und -ausführung, der durch eine höhere Instanz moduliert wird. Die Wahrnehmung und Verarbeitung der Sensordaten erfolgt im Block *Sensory & Perceptual Structures*. Der durch die Sensorik wahrgenommene Zustand eines Systems wird über die *Trigger Database* mit einer Sequenz auszuführender Aktionen (*Behavioural Schemata*) verknüpft. Die Verknüpfung ist dabei nach Effektivität der Zielerreichung einer Aktionssequenz gewichtet. Sich gegenseitig behindernde Aktionen (gleichzeitiger Zugriff auf exklusive Ressourcen) werden durch das *Contention Scheduling* verhindert, indem nur eines der potentiellen *Behavioural Schemata* Zugriff auf die gewünschte Ressource erhält und letztendlich zur Ausführung an das *Effector System* übertragen wird.

Die bedeutendste Funktion des Norman & Shallice Modells ist die Möglichkeit, neu erworbene Handlungsmuster nach mehrmaliger erfolgreicher Anwendung als routinierte Handlungen zu speichern, sodass bei späterer Aktivierung weniger bis keine Aufmerksamkeit zur Überwachung der Aktionen benötigt wird. Für diese Aufgabe ist das *Supervisory Attentional System* (SAS) im Norman & Shallice Modell vorgesehen. Primär überwacht das SAS die Ausführung eines gerade gestarteten Handlungsmusters und vergleicht dessen Zustände mit den erwarteten Zuständen und dem gewünschten Zielzustand des Systems, um etwaige unbeabsichtigte Handlungen zu erkennen. Daraufhin adaptiert das SAS die Stärken bzw. Wichtungen, mit denen bestimmte Handlungsmuster und Zustände in der *Trigger Database* verknüpft sind. Verhaltensmuster, die sich als ungeeignet erwiesen haben, werden dabei gedämpft, alternative Verhaltensmuster werden hingegen bei der nächsten Aktivierung in der *Trigger Database* stärker berücksichtigt. Hierdurch werden alle Handlungen des Handlungsspektrums für den aktuellen Systemzustand berücksichtigt und erfolgreiche Handlungen hervorgehoben. Bilden sich effektive Handlungen heraus, wird deren Ausführung im Laufe der Zeit weniger vom SAS überwacht, womit diese schneller ausgeführt werden. Als weitere Aufgabe für das SAS ist im Modell von Norman & Shallice die Generierung von neuen Verhaltensmustern vorgesehen, um das Handlungsspektrum des Systems in neuen Situationen um angepasste Verhaltensmuster zu erweitern.

Wie Garforth in seiner Arbeit anmerkt, ist im Modell von Norman & Shallice zwar vorgesehen, dass häufig aufgerufene Verhaltensmuster kontinuierlich zu reflexartigen Handlungen werden,

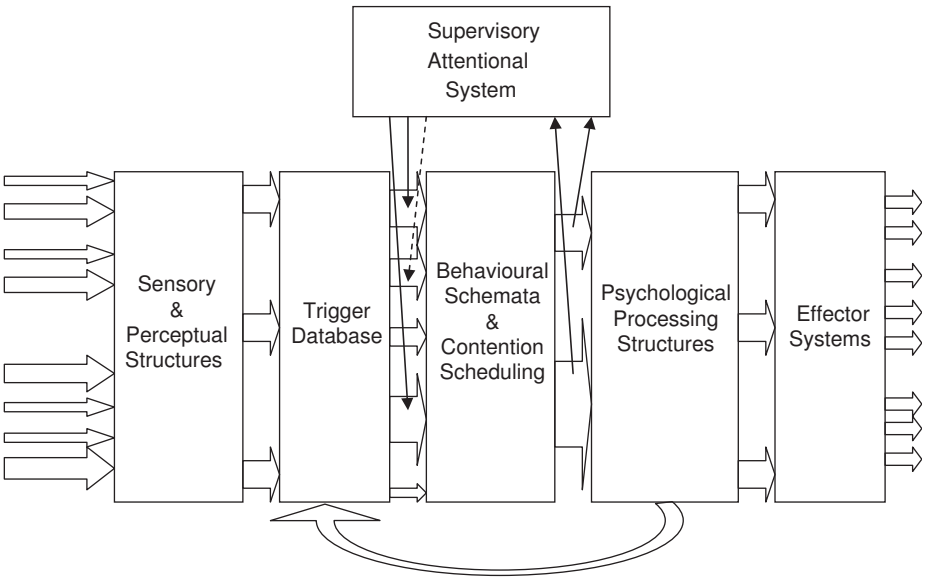


Abbildung 2.6 Modell nach Norman & Shallice (1986), entnommen aus [20].

jedoch wird von Norman & Shallice nicht konkretisiert, wie dies erfolgen soll.

Garforth (2006) Garforth implementiert und erweitert das Modell von Norman & Shallice durch neuronale Komponenten (Abildung 2.7). Dabei erfolgt eine strikte Trennung (gestrichelte Linie) in Ausführung (unten) und Überwachung (oben) von Handlungen. Das reaktive Verhalten ist im Modell durch die Verbindung von *Episodic Memory* (EM) und *Procedural Memory* (PM) durch den *Contention Scheduler* (CS) gegeben. In der EM werden Sensordaten aus dem *Perception Layer* entgegengenommen und verarbeitet. In der neuronalen Implementierung (Abbildung 2.8 links) erfolgt dies in unterschiedlichen Ebenen eines neuronalen Netzes. In der unteren Ebene werden Sensordaten direkt wahrgenommen, wohingegen in den höheren Ebenen die Wahrnehmungen weiter abstrahiert werden. Ergänzend dazu ist die PM ebenfalls als mehrschichtiges neuronales Netz aufgebaut (Abbildung 2.8 rechts). In den unteren Schichten der PM sind nur sehr einfache primitive Aktionen möglich, welche hingegen in den oberen Schichten zu abstrakteren Aktionen kombiniert werden. Die PM ist wiederum mit dem *Motor Layer* verknüpft, um Aktionen letztendlich von den Aktoren ausführen zu lassen.

Die Verbindung zwischen Sensorik und Motorik erfolgt durch den *Contention Scheduler*, welcher sowohl die Neuronen der EM und PM auf unterschiedlichen Ebenen verbindet, als auch den Zugriff auf exklusive Ressourcen verwaltet. Je niedriger Sensorik und Motorik auf neuro-

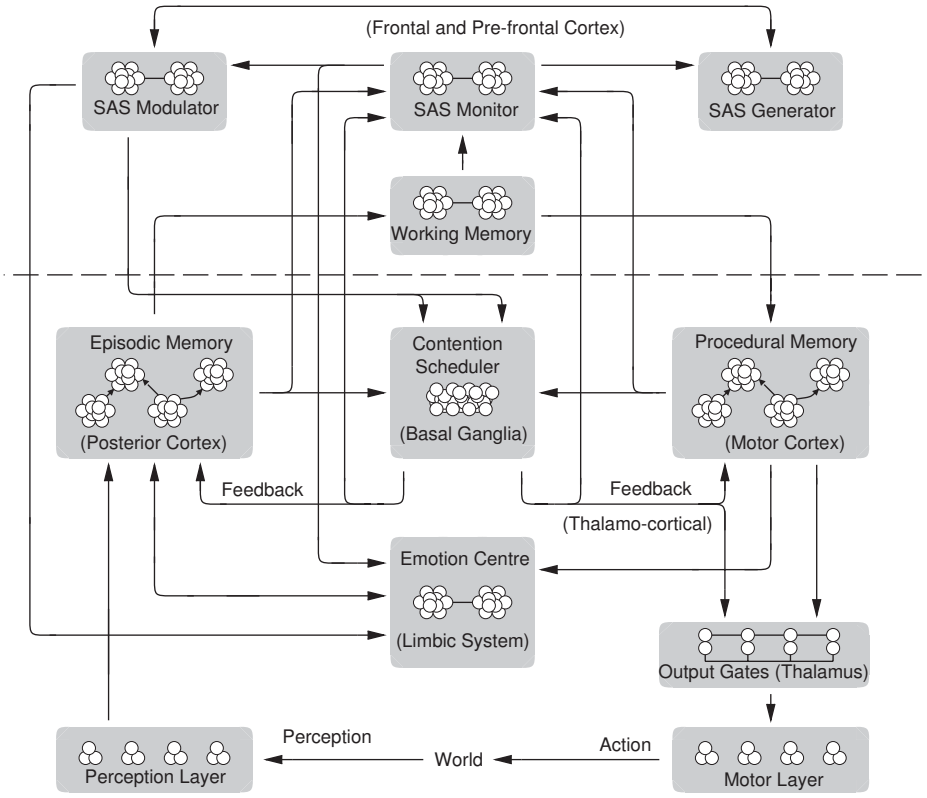


Abbildung 2.7 Modell nach Garforth (2006) auf Grundlage des Norman & Shallice Modells, [20].

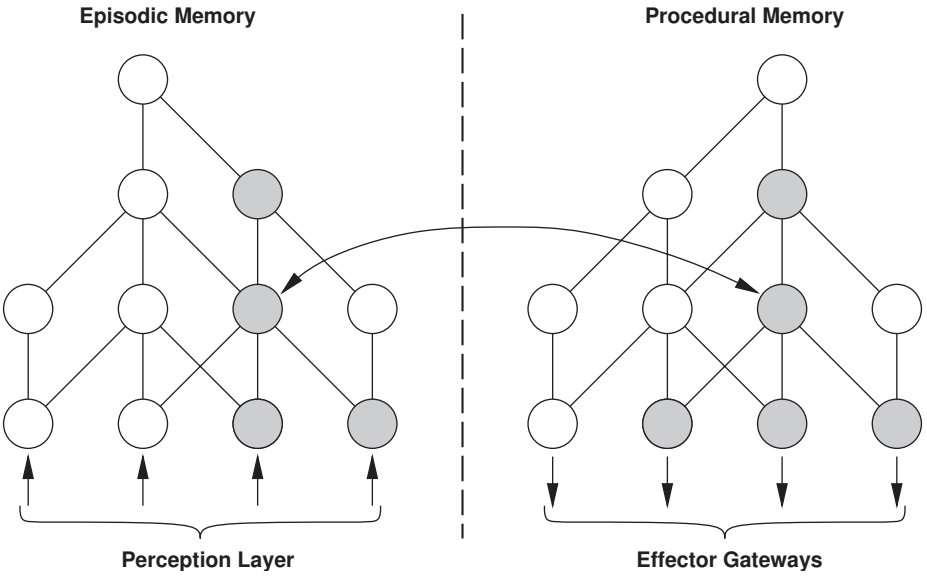


Abbildung 2.8 Episodic und Procedural Memory, [20]

naler Ebene verbunden sind, desto reaktiver ist das System für diese Sensorwahrnehmung. Abstraktere Verhalten werden im Gegensatz dazu über Verbindungen beider Systeme in höheren Ebenen erreicht.

Das SAS hat nun, wie auch im Modell von Norman & Shallice, unter anderem die Aufgabe, die Verknüpfungen im CS zu modulieren und die Wichtungen in EM und PM durch das *Emotion Centre* anzupassen. Der *SAS Monitor* überwacht dazu die Verbindung von EM und PM durch den CS. Der *SAS Monitor* ist dabei stark mit der *Working Memory* (WM) verknüpft, welche die inneren Zustände des System speichert. Über die WM hat der *SAS Monitor* Zugriff auf die aktuellen Ziele des Systems und kann erkennen, welches die beabsichtigten und durchgeführten Handlungen sind. Wird eine Diskrepanz zwischen beabsichtigten Handlungen in der WM und reaktiven Handlungen in der EM erkannt, aktiviert der *SAS Monitor* den *SAS Modulator*. Dieser passt die Wichtungen in der EM dahingehend an, dass das aktuelle Verhaltensmuster gedämpft wird und alternative Verhaltensmuster verstärkt werden, um die durchgeführten den beabsichtigten Handlungen anzupassen. Ist es unmöglich, entsprechende Verhaltensmuster zu verstärken, welche den beabsichtigten Handlungen des Systems entsprechen, wird der *SAS Generator* aktiviert um neue Verhaltensmuster zu erzeugen. Dieser zusätzliche Aufwand ist nötig, wenn in neuen Situationen die Fähigkeiten des aktuellen Systems nicht ausreichen um gesetzte Ziele zu erreichen.

Modellansätz für VirGo^4 Einige Ansätze der vorgestellten Arbeiten basieren auf der Idee, dass die unterschiedlichen Ebenen eines Verhaltensmodells über unterschiedlich stark abstrahierte Aktionen bzw. *Behaviour* verfügen. Suh beschreibt in [76] ein zweischichtiges Modell, in dem zwischen einem *action pattern layer* und einem *reactive behavior plan layer* unterschieden wird. Ein *action pattern* kann in einzelne primitive Motorbefehle aufgeteilt werden; ein *reactive behavior plan* wird wiederum aus mehreren *action pattern* zusammen gesetzt und anhand der Motivation bzw. der Ziele des Systems ausgewählt. Diese Struktur lässt einen hierarchischen Aufbau von Aktionen erkennen, welcher sich auch in der *Episodic Memory* und der *Procedural Memory* nach Garforth wiederfinden lässt. Die Hierarchie im Garforth-Modell ist jedoch nicht auf zwei Ebene begrenzt, sondern nur durch die Anzahl der neuronalen Ebenen in der EM und PM.

Ein solcher zweischichtiger Ansatz zum Kombinieren von kleineren Aktionen zu abstrakteren Handlungen wurde auch in einem ersten Konzept des VirGo^4-Modells integriert (Abbildung 2.9). In diesem dreischichtigen Konzept, welches die zwei Ebenen der Aktionskombination mit einer zusätzlichen Planungsebene verbindet und dem klassischen LoB-Modell entnommen wurde,

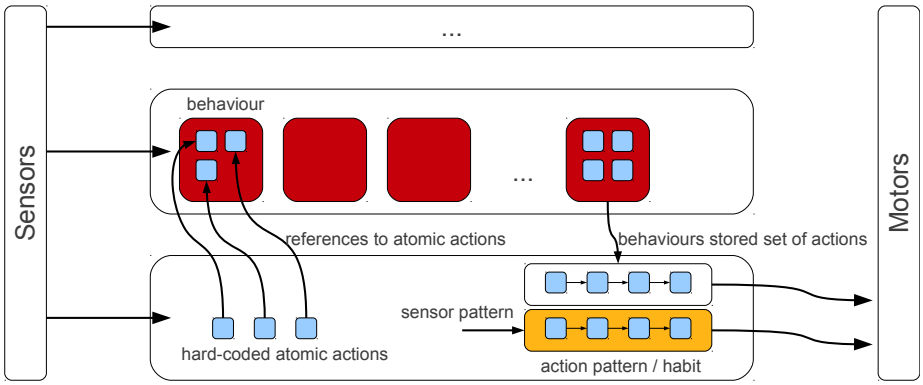


Abbildung 2.9 Erstes Konzept auf Grundlage der zweischichtigen Aktionskombination.

hat jede der drei Schichten Zugriff auf die Sensordaten. Den drei Schichten wird es jedoch nur über die unterste Schicht ermöglicht, Aktionen über die Motoren auszuführen. Grundsätzlich werden Aktoren nur durch sogenannte *atomic actions* (AA) angesprochen. Diese atomaren Aktionen stellen unteilbare Befehle dar, die aufgrund ihrer Eigenschaft nicht unterbrochen werden können. Um abstraktere Verhalten zu ermöglichen, werden mehrere dieser atomaren Aktionen zu Sequenzen gruppiert. Dies geschieht in den *behaviour* der mittleren Schicht. Die Funktion dieser *behaviour* besteht darin, ausgehend von den höheren Zielen des Systems, atomare Aktionen so zu verknüpfen, dass mit einer Sequenz von primitiven Aktoren-Befehlen bestimmte Zustände erreicht werden können. Erweist sich ein neu geformtes *behaviour* über längere Zeit als brauchbar, um einen Zielzustand zu erreichen, wird diese Kombination von atomaren Aktionen (*action pattern*) in der unteren Ebene abgelegt, um direkt Zugriff auf die Sensordaten zu erhalten und der kontinuierlichen Überwachung zu entgehen. Ein solches *action pattern* wird hier zusätzlich als *habit* bezeichnet und verdeutlicht, dass diese *action pattern* ohne Verzögerung zur Ausführung kommen, wenn die entsprechende Kombination von Sensorwerten wahrgenommen wird.

Dieses erste Konzept folgt klar dem zweischichtigen Ansatz von Suh, da es eine klare Trennung von reaktiven Handlungen in der unteren Schicht und zusammengesetzten Verhalten in der mittleren Schicht beschreibt. Sensoren und Aktoren sind in der untersten Schicht sehr direkt miteinander verknüpft, da sich deren Nützlichkeit bereits in einer früheren Phase erwiesen hat. Für neue Situationen, in denen sich keine der früheren Handlungssequenzen anwenden lässt, ist die mittlere Schicht mit ihren Verhalten zuständig. Dabei können Aktionen in der mittleren Schicht nur durch die den *behaviour* inhärenten atomaren Aktionen ausgeführt werden.

Betrachtet man die unüberschaubare Menge an Sensorinformationen, denen ein System in der

realen Welt ausgesetzt ist (vor allem aufgrund der visuellen Wahrnehmung), wird ersichtlich, dass *sensor pattern* nicht ohne Vorverarbeitung mit Aktionen verknüpft werden können. Garforth löst dieses Problem mit den mehrschichtigen neuronalen Netzen der *Episodic Memory* und der *Procedural Memory*. Durch diese mehrschichtige Struktur werden Sensorinformationen in höheren Schichten zunehmend abstrahiert. Demzufolge ist es in höheren Schichten möglich, Sensorinformationen zu kombinieren um nicht direkt messbare Eigenschaften der realen Welt zu erkennen. Ein Beispiel dafür wäre die Objektklassifikation, welche auf Sensorinformationen der unteren Schichten zugreift, jedoch nicht für jedes reaktive Verhalten benötigt wird.

Aus diesen Überlegungen heraus ist eine Folgekonzept entstanden (Abbildung 2.10). Das vorherige Modell wurde hier um eine Sensordatenvorverarbeitung und eine Sensordatenvorhersage erweitert, um einerseits die große Menge an Sensorinformationen auf Zustände abbilden zu können und andererseits zukünftige Zustände vorhersagen zu können. Durch diese Erweiterung ist es möglich, die vorhergesagten erwarteten Zustände mit den tatsächlichen Endzuständen einer Handlung zu vergleichen und eventuell die gewählte Handlung anzupassen oder zu ersetzen. Die Vorverarbeitung der Sensordaten beinhaltet dabei die spatiale Merkmalsextraktion in unterschiedlichen Bereichen wie u.a. Farbkanäle, Intensitäten und Entfernungen, sowie die nachträgliche Verarbeitung von Merkmalen unter der Berücksichtigung vorheriger Ereignisse und Wahrnehmungen, die in einem eigenen Speicher festgehalten werden. Analog zu den *action pattern* in der unteren Ebene werden in der mittleren Ebene die erwarteten zukünftigen Zustände von *behaviour* prognostiziert. Im Gegensatz zur unteren Ebene werden in den *behaviour* jedoch abstraktere Sensorinformationen bzw. Zustände prognostiziert statt den darunter liegenden Rohdaten.

VirGo^4 Verhaltensmodell Die Grundlage für Anwendungsfälle und Implementierung bildet das *Levels-of-Behaviour*-Verhaltensmodells der Abbildung 2.11. Die Grundidee der Aufteilung in eine reflexive Ebene mit Handlungen (atomare Aktionen und verschachtelter Handlungssequenzen), deliberative Ebene zur Generierung dieser Handlungen und eine kognitive Ebene für höhere Prozesse bleibt erhalten. Weitere spinale Reflexe werden unterhalb dieser Ebenen angenommen und sind nicht unterbrechbar, da Sensorik und Motorik direkt miteinander verbunden sind.

Die beiden unteren Ebenen enthalten jeweils einen eigenen Speicher, der u.a. den aktuellen und den gewünschten Zustand enthält, sowie einen Monitor, der die Abweichung beider Zustände

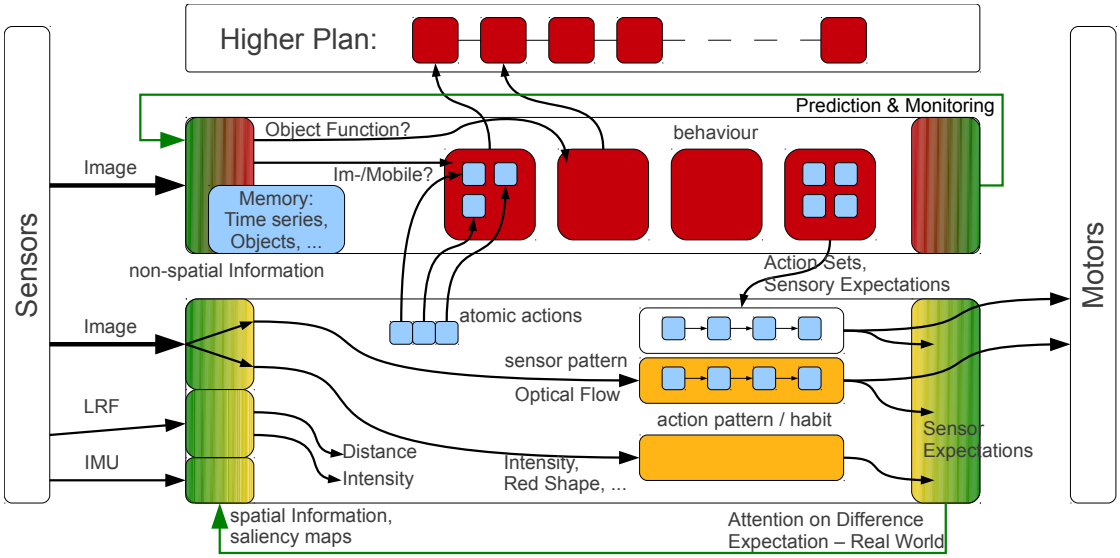


Abbildung 2.10 Erweitertes Modell mit Sensordatenvorverarbeitung und -vorhersage, sowie Speicher für vergangene Zustände und erweiterte Informationen.

beobachtet. Die Abweichung von Zuständen auf beiden Ebenen hat dabei unterschiedliche Auswirkungen. Auf der untersten Ebene werden über die Exafferenzen (Abweichung einer gewünscht ausgeführten Aktion zu einer wahrgenommenen Aktion) Reaktionen angestoßen, um kleinen Abweichungen entgegenzuwirken oder bei größeren Abweichung eine aktuelle Handlung zu unterbrechen. Der Monitor der deliberativen Ebene (SAS Monitor) dient der Zielverfolgung. Wird in der Homöostase eine Verschiebung des Zielzustandes festgestellt, so wird aktiv über den SAS Modulator in der unteren Ebene eine Handlung dafür ausgewählt. Der Contention Scheduler der unteren Ebene wählt von den parallel aktivierten Aktionen diejenige mit der höchsten Priorität (Wichtung) aus und verhindert damit zusätzlich, dass konkurrierende Aktionen auf exklusive Ressourcen zugreifen.

Die Episodic Memory der unteren reflexiven Ebene enthält die Wahrnehmung durch die Sensorik (Afferenzen) sowie die durch deren Aktionen hervorgerufenen Efferenzen über einen längeren Zeitraum. Auf diesen Speicher kann schnell zugegriffen werden. In der Working Memory der deliberativen Ebene sind durch Sensordaten beschriebene Zustände und die durch einen Planer vorgegebenen Ziele enthalten. Die Semantic Memory nutzt diese Informationen und die kreative Kognition der oberen Ebene, um über längere Zeiträume Wissen und Regeln über die Umwelt zu generieren. Diese semantischen Informationen stehen dann dem Generator zu Verfügung um in neuen Situationen angepasste Verhalten erstellen zu können. Der Generator ist über die Semantic Memory und den Planer sehr stark mit der kognitiven

Ebene verbunden, um bei fehlschlagender Behaviour-Generation auf zusätzlich generiertes Wissen zugreifen zu können.

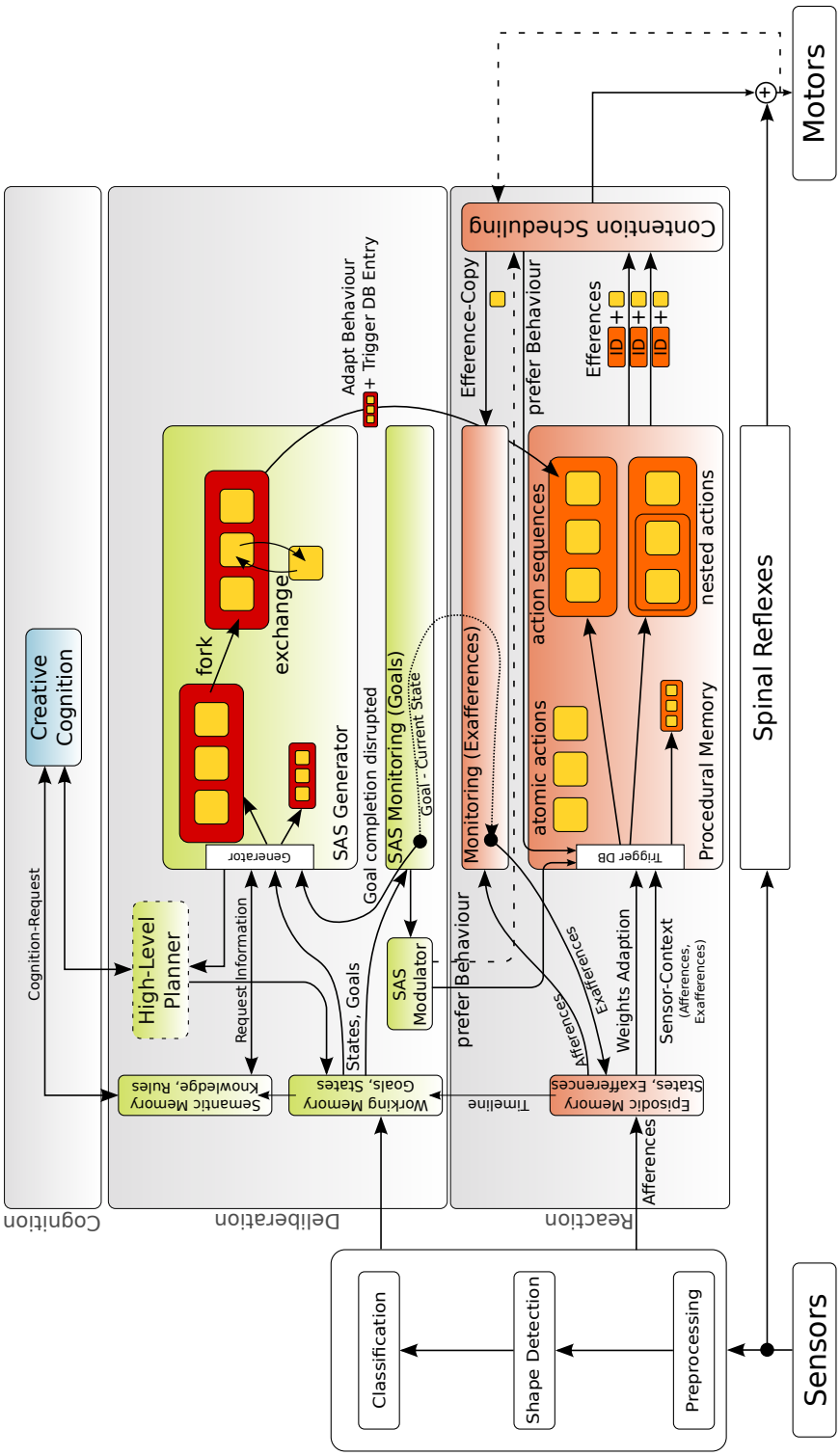


Abbildung 2.11 Finales Konzept des Verhaltensmodells. Sensordaten werden nun in verschiedenen Speichern (Episodic Memory, Working Memory, Semantic Memory) zur Weiterverarbeitung bereitgehalten. Die unteren beiden Schichten (Reflex, Deliberation) haben jeweils einen eigenen Monitor, um auf Abweichungen innerhalb jeder Ebene reagieren zu können.

2.1.3 AP 4200: Real. Levels-of-Behaviour-Modell

Erster Schritt Der erste Schritt soll die wichtigsten Funktionen der ersten, reaktiven LoB-Ebene darstellen. Gleichzeitig soll schnell eine laufende Anwendung gezeigt werden, die im Kern die Methoden und Werkzeuge verwendet, die später auch in komplexeren Szenarien zum Einsatz kommen.

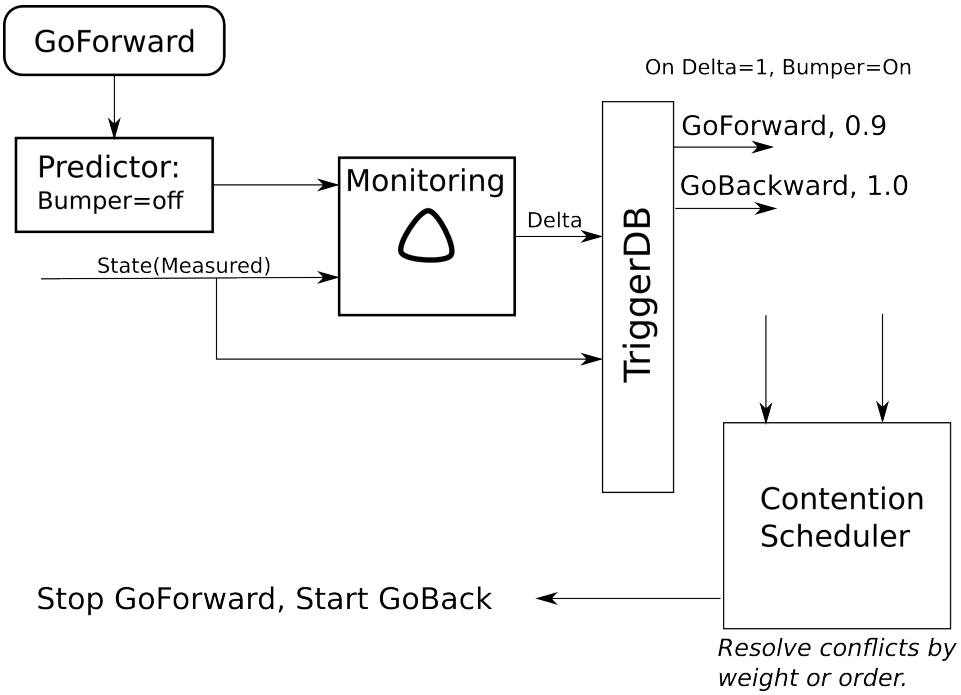


Abbildung 2.12 Der erste Anwendungsfall

Abb. 2.12 zeigt den Ablauf des ersten Implementierungsfalls. Der Roboter fährt mit einer festen Geschwindigkeit vorwärts (Verhalten: GoForward). Dabei wird erwartet, dass die Kontaktsensoren keinen Kontakt melden. Sollte dennoch ein Sensor ausgelöst werden, dann stoppt der Roboter und fährt rückwärts (Verhalten: GoBackward). Der generische Ablauf ist wie folgt:

1. Eine höhere Instanz bestimmt ein Verhalten zur Ausführung. Dazu wird diese Information in der Schaltdatenbank (TriggerDB) abgelegt.

2. Das Vorhersagemodul (Predictor) definiert den Erwartungswert der Sensoren während der Ausführung des Verhaltens.

3. In der Überwachungskomponente (Monitoring) wird die Vorhersage mit dem tatsächlichen Zustand verglichen.

4. Aufgrund des Unterschieds (Delta) wird in der Schaltdatenbank ermittelt, welche Verhalten mit welcher Gewichtung zur Ausführung vorgeschlagen werden. Das bezieht auch Verhalten mit ein, die von einer höheren Schicht zur Ausführung bestimmt wurden.

5. Eine Dispositionseinheit (Content Scheduler) ermittelt aus den Verhalten mit ihren Gewichtungen eine zulässige und konsistente Kombination, die dann tatsächlich zur Ausführung kommt.

Als Beschreibung der Verhalten soll später *Roby* (s. AP3100) verwendet werden. Roby ist in der Skriptsprache *Ruby* verwirklicht. Das bedeutet, das zusätzliche Funktionen leicht als Ruby-Skript eingebunden werden können. Dieser Weg wird im vorliegenden Fall eingeschlagen, um das oben dargestellte Beispiel zu verwirklichen. Es sollen ebenso als Ausgangspunkt für die weitere Entwicklung des LoBs dienen, in der Schritt für Schritt aufwändigere Szenarien implementiert werden. Das heißt auch, dass u.U. Funktionen von Ruby nach C/C++ ausgelagert werden. Neben der bereits erwähnten Methode Roby durch neue Ruby-Implementierungen zu erweitern bietet Roby auch die Möglichkeit sehr einfach bestehende C/C++-Implementierungen und somit bestehende Komponenten zu verwenden. Durch diese Fähigkeiten ist Roby optimal dazu geeignet modulare Systeme aufzubauen, bei denen mehrere Programmteile miteinander interagieren.

Prototypenimplementierung Der Wechsel von Verhalten aus der reaktiven und der deliberativen Schicht wurde anhand einer prototypischen Implementierung untersucht. Dazu wurde eine Szenario geschaffen (Abbildung 2.13), dass den Roboter veranlasst von der unteren reaktiven auf die obere deliberative Ebene zu wechseln. Der Roboter hält durch die Trigger der unteren Ebene einen Sicherheitsabstand, der größer ist als die dargestellte schmale Verengung im Szenario. Um den Zielpunkt zu erreichen und die enge Passage zu überwinden ist es notwendig, die getriggerten Aktionen (den Wänden ausweichen) mit den gewollten Aktionen (durch die Verengung fahren) zu überlagern.

Abbildung 2.14 zeigt im oberen Graphen den Abstand zum Ziel in Metern und im unteren Graphen die Wichtungen der einzelnen Aktionen. Rein reaktiv kann der Roboter die Dreh-Verhalten und das Rückwärtsfahren ausführen, wenn die entsprechenden Triggerbedingungen erreicht

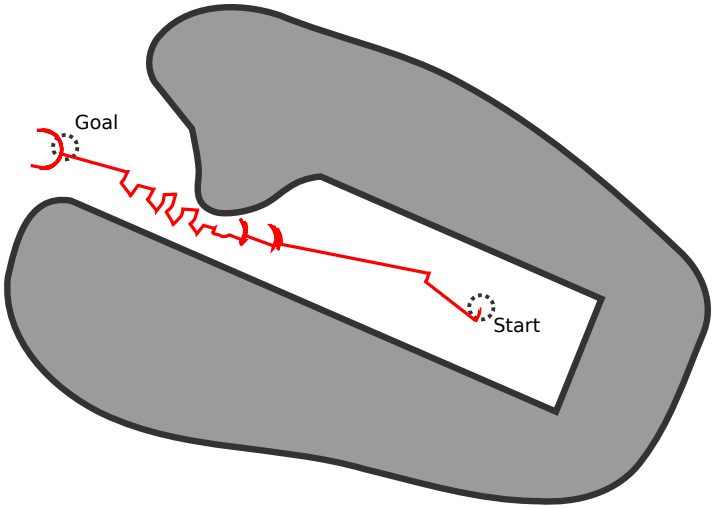


Abbildung 2.13 Szenario *Bottleneck* bei dem der Roboter eine Verengung überwinden muss, um vom Startpunkt zum Ziel zu gelangen.

werden. Diese Aktionen werden mit einer Wichtung von $w = 0.9$ aktiviert. Parallel zu den Triggern werden ebenfalls Aktionen vom *SAS Modulator* der deliberativen Ebene vorgegeben. Diese gewollten Aktionen beinhalten ebenfalls die Dreh-Verhalten und zusätzlich das Vorwärtsfahren mit initialen Wichtungen von jeweils $w = 0.5$. Die Wichtung für das Vorwärtsfahren wird nun erhöht, wenn im *SAS* festgestellt wird, dass sich dem Ziel nicht weiter genähert wird bzw. wenn die Ableitung der Distanz zum Ziel über die Zeit sehr gering ist.

Im gegebene Beispiel tritt eine solche Situation um den Zyklus 200 auf, bei dem sich der Roboter der Verengung nähert und damit die beiden Dreh-Verhalten abwechselnd ausgeführt werden. Dabei wird das Links-Drehen durch die *TriggerDB* ausgelöst (grün, $w = 0.9$) um die Kollision mit der rechten Wand zu vermeiden, und das Rechts-Drehen wird durch den *SAS Modulator* ausgelöst (blau, $w = 0.5$) um sich auf den Zielpunkt auszurichten. Durch die Stagnation der Distanz zum Ziel erhöht sich die Wichtung für das Vorwärtsfahren kontinuierlich (rot, $w \geq 0.5$), bis es um den Zyklus 275 herum das reaktive Links-Drehen (grün, $w = 0.9$) ablöst. Diese Situation tritt in den Zyklen 300 bis 400 erneut auf, wobei der Roboter nun die Verengung überwinden kann (Distanz zum Ziel sinkt kontinuierlich). Beide Situationen sind im Szenario (Abb. 2.13) durch die zwei roten Bereiche vor der Verengung erkennbar.

Erstellen eines Testdatensatzes für weitere Analysen Das Levels-of-Behaviour-Modell besteht, wie früher bereits erläutert, aus verschiedenen Modulen die einzelne Aufgaben über-

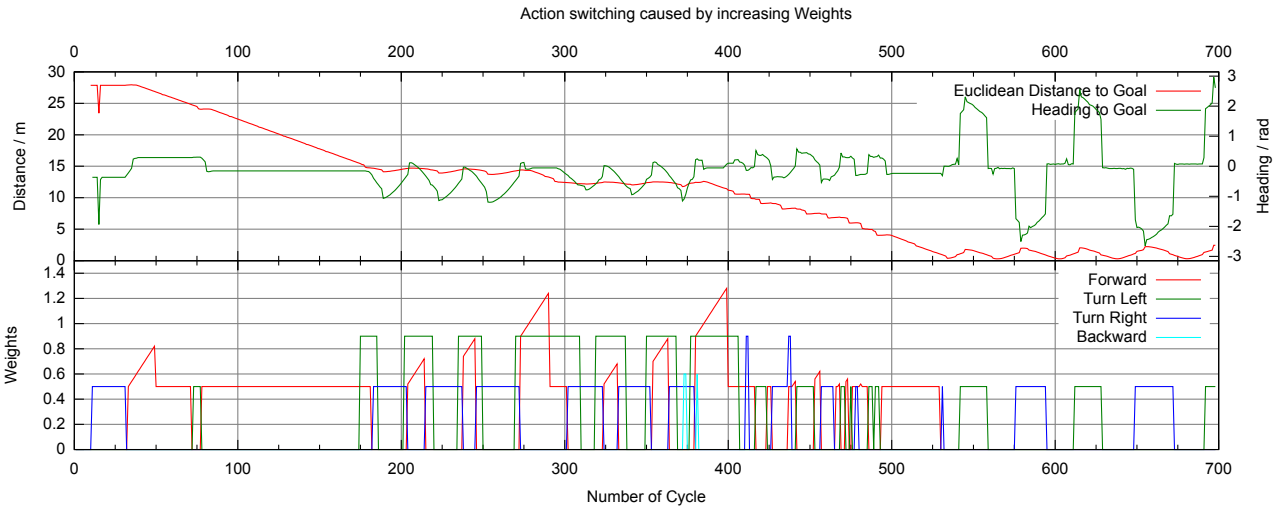


Abbildung 2.14 Die Wichtungen der einzelnen Verhalten *Forward* (rot), *Turn Left* (grün), *Turn Right* (blau), *Backward* (cyan) im unteren Graphen, die durch die Abstände zum Ziel *Euclidean Distance to Goal* (rot) und *Heading to Goal* (grün) im oberen Graphen beeinflusst werden.

nehmen sollen. Zur Lösung der Aufgabe der Erkennung von Objekten wurde, wie bereits beschrieben, ein erstes Verfahren implementiert und getestet. Damit die Objekterkennung aber in das Gesamtsystem eingebunden werden kann, müssen zusätzlich auch noch Methoden zur Vorverarbeitung der Sensordaten implementiert und integriert werden. Des weiteren müssen die Module für die Vorhersage und das Monitoring noch implementiert und getestet werden.

Damit diese einzelnen Module entwickelt und getestet werden können, müssen zunächst einmal exemplarisch Daten aller zur Verfügung stehenden Sensoren aufgezeichnet werden. Dieser Testdatensatz soll dann analysiert werden, um geeignete Verfahren und Algorithmen zur Lösung der Teilaufgaben auszuwählen und diese dann zu testen. Mit diesen Testdaten werden dann beispielsweise die Effekte verschiedener Vorverarbeitungsmethoden auf die Objektklassifikation analysiert, Vorhersagemethoden (siehe 2.1.4) getestet und falls nötig mehrere verglichen sowie weitere notwendige Tests durchgeführt. Dies soll vor allem dabei helfen, möglichst schnell Ergebnisse über die Anwendbarkeit der gewählten Methoden zu erhalten, ohne dass jedes mal neue Testdaten aufgezeichnet werden müssen.

Wie bereits im AP2300 erläutert, müssen zunächst jedoch die Treiber und Module implemen-

tiert werden, die ins Framework integriert werden, um die Daten der Sensoren auszulesen und für die weitere Verarbeitung zur Verfügung zu stellen. Bisher können die Daten der Odometrie, des SICK Laser-Entfernungsmessers, sowie der IMU ausgelesen und gespeichert werden. Die Integration des zuvor entwickelten Treibers für die beiden IDS-Kameras in das Framework hat jedoch viel Zeit beansprucht und muss noch abgeschlossen werden. Für den Testdatensatz soll zusätzlich noch die aktuelle Position der PTU und die Motorkommandos aufgezeichnet werden.

Implementierung Trigger-Algorithmus Der in AP3200 beschriebenen Trigger-Algorithmus wurde innerhalb des AP4200 für einen Matrix-Multiplikation-Algorithmus konkretisiert. Das vollständige Klassendiagramm ist in Abbildung 2.15 dargestellt. Es zeigt eine detailliertere Darstellung der Trigger-Klassen *Trigger*, *TriggerBuilder*, *TriggerBuilderInterface*, *TriggerAlgorithm* und *TriggerAlgorithmInterface*.

Jeder Trigger wird im TriggerBuilder erstellt. Dort wird er mit einem konkreten TriggerAlgorithm konfiguriert und die entsprechenden Datenquellen gesetzt. Dabei verwendet der TriggerBuilder das TriggerBuilderInterface und der TriggerAlgorithm das TriggerAlgoritmInterface um auf den Trigger zugreifen zu können. Die Datentypen *Behaviour*, *WeightedBehaviour* und *Description* werden für die Verwaltung der Verhalten im Trigger verwendet. In der Struktur WeightedBehaviour befinden sich die Verhalten und deren Wichtungen, die von jedem Trigger erstellt und weitergeleitet werden. Für die Fehlerbehandlung während der Laufzeit stehen die runtime_error-Klassen *BuilderError*, *TriggerError*, *DataError* und *AlgorithmError* bereit.

Eine konkrete Implementierung des TriggerAlgorithm ist der *MatAlgorithm* der sich außerhalb des trigger-Namensraums im gleichen Klassendiagramm befindet. Dieser einfache Algorithmus multipliziert die eingehenden Sensordaten als Vektor mit einer Matrix um Wichtungen für die Verhalten zu erhalten. Die Wichtungen w für n Verhalten erhält man nach:

$$w = (S.^{e})^{\mathsf{T}} \cdot M \tag{2.1}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} s_1^e & s_2^e & \dots & s_m^e \end{bmatrix} \cdot \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{21n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m1} & m_{m2} & \cdots & m_{mn} \end{bmatrix} \tag{2.2}$$

wobei der Vektor $S = [s_1, s_2, ..., s_m]$ die zusammengefügten Sensoreingangsdaten enthält und elementweise mit dem Exponent e potenziert wird ($S.^e$ bezeichnet dabei die elementweise Potenz des Vektor S). Damit besteht die Möglichkeit den Sensordaten durch einen negativen Exponent eine inverse Bewertung zu geben. Die $m \times n$ Matrix M repräsentiert die Verknüpfung jedes Sensorwertes mit jedem Verhalten.

Die Konfigurationsparameter des Algorithmus sind e und M. Zusätzlich muss in der Konfiguration die Reihenfolge der Verhalten angegeben werden, um eine korrekte Zuordnung zu den Spalten der Matrix zu erhalten. Dies kann vernachlässigt werden, wenn die Matrix nicht manuell definiert werden soll.

Die Konfiguration wird im YAML-Format angegeben. Die Konfiguration für einen MatAlgorithm mit drei Sensor-Kanälen und vier Verhalten könnte wie folgt aussehen:

```
behaviours:
  forward: 0
  left: 1
  right: 2
  backward: 3
exponent: -1.0
matrix:
 - [0.2, 0.2, 1.2, 0.4]
 - [0.1, 0.4, 0.4, 1.0]
 - [0.2, 1.2, 0.2, 0.4]
```

Die Konfigurationsdatei beinhaltet drei YAML-Knoten: `behaviours`, `exponent` und `matrix`. Der `behaviours`-Knoten listet Verhalten und deren Ordnungsnummer nach dem Schema `␣␣<name>:␣<position>` auf. Der `exponent` kann als Gleitkommazahl angegeben werden. Der `matrix`-Knoten wird zeilenweise aufgelistet und beinhaltet so viele Reihen wie Sensorkanäle verwendet werden.

Die Wichtung der einzelnen Verhalten errechnet sich nach Einsetzen der Konfigurationsparameter durch:

$$\begin{bmatrix} w_{forward} \\ w_{left} \\ w_{right} \\ w_{backward} \end{bmatrix} = \begin{bmatrix} s_1^{-1.0} & s_2^{-1.0} & s_3^{-1.0} \end{bmatrix} \cdot \begin{bmatrix} 0.2 & 0.2 & 1.2 & 0.4 \\ 0.1 & 0.4 & 0.4 & 1.0 \\ 0.2 & 1.2 & 0.2 & 0.4 \end{bmatrix} \tag{2.3}$$

Implementierung Trigger-Komponente Das Klassendiagramm in Abbildung 2.16 zeigt einige der bekannten Trigger-Klassen aus der Trigger-Algorithmus Implementierung. Hinzu kommen die Klassen *BaseTrigger* und *LaserTrigger* für die Trigger-Komponente. Der BaseTrigger definiert Methoden und Schnittstellen die von allen Triggern verwendet werden. Darunter fallen die Aktionen *setTriggerAlgorithm*, *addDataSource* und *addBehaviour* und der Ausgangsport *proposed_behaviours* für die gewichteten Verhalten. Der LaserTrigger übernimmt diese Schnittstellen und erweitert diese um spezifische Konfigurationsparameter und Eingangsports für die Laser-Daten. Innerhalb des LaserTrigger wird der TriggerBuilder verwendet um einen Trigger mit dem *MatAlgorithm* zu erzeugen. Über den Zeiger *mTrigger* kann der LaserTrigger die Methode *process()* des Triggers und damit des MatAlgorithm ausführen in der die Kalkulation nach Gleichung 2.1 bzw. Glg. 2.2 stattfindet.

Die LaserTrigger-Komponente kann nun im Komponenten-Netzwerk von *rock* eingesetzt werden. Die Trigger-Komponente übernimmt die Hooks der orogen-Komponente (Abbildung 2.17). Im *configureHook* wird die YAML-Konfiguration eingelesen und für den TriggerBuilder im *startHook* bereit gestellt. Dort wird der Trigger initialisiert indem Sensoren, Verhalten und der Algorithmus definiert werden. Der *updateHook* verwendet den *rock*-Datentyp *LaserScan* für die Laser-Daten und gibt die Entfernungswerte aus *ranges* als Vektor an den Trigger und dessen Algorithmus weiter. Die Verarbeitung der Sensordaten findet durch den Aufruf der *process()*-Methode des vorher initialisierten Triggers statt. Der *stopHook* und *cleanupHook* sind für das saubere Beenden des Prozess-Zyklus zuständig.

Voraussetzung Treibermodule Wie bereits erwähnt, gehören neben den Kernkomponenten des LOB auch die Treiber, die die Daten der Sensoren für das System zur Verfügung stellen, als wichtige Komponenten mit zu der Realisierung des LOB. Die Treiberkomponenten stellen die Daten der Sensoren zur Verfügung, auf die dann wiederum Trigger direkt reagieren

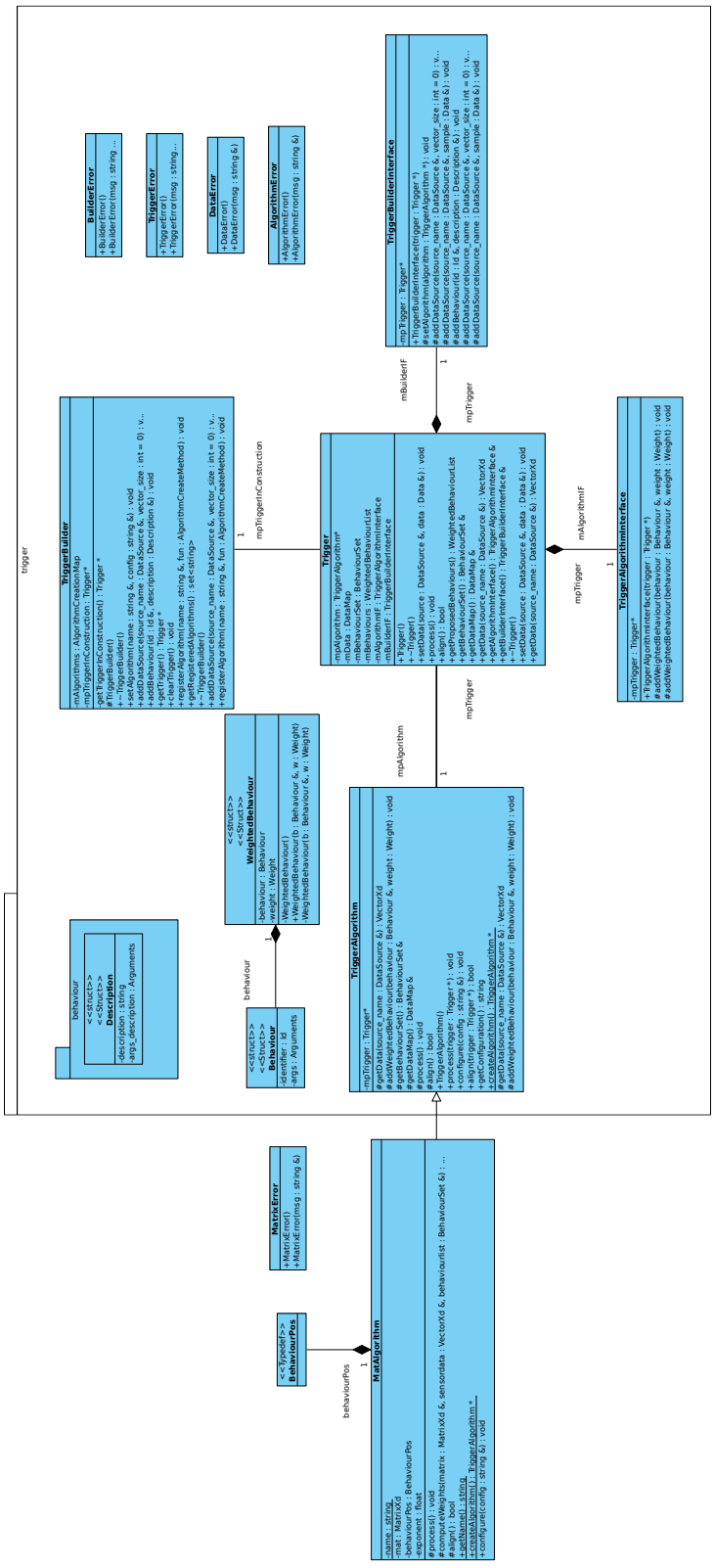


Abbildung 2.15 Klassendiagramm der Trigger-Komponente und einer konkreten Implementierung des Trigger Algorithmus (MatAlgorithm)

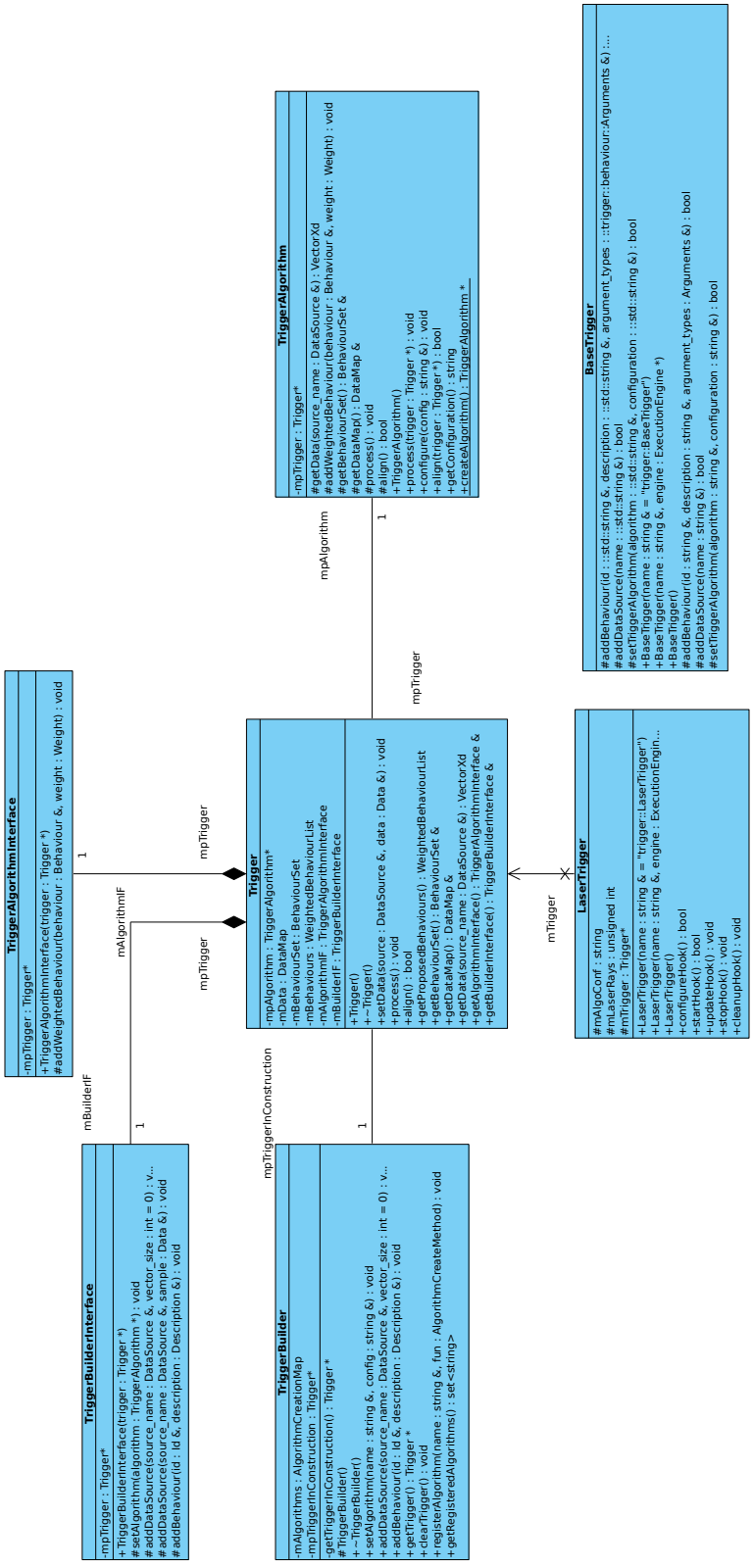


Abbildung 2.16 Klassendiagramm der Trigger-Komponente und einer konkreten Implementierung eines Triggers (LaserTrigger)

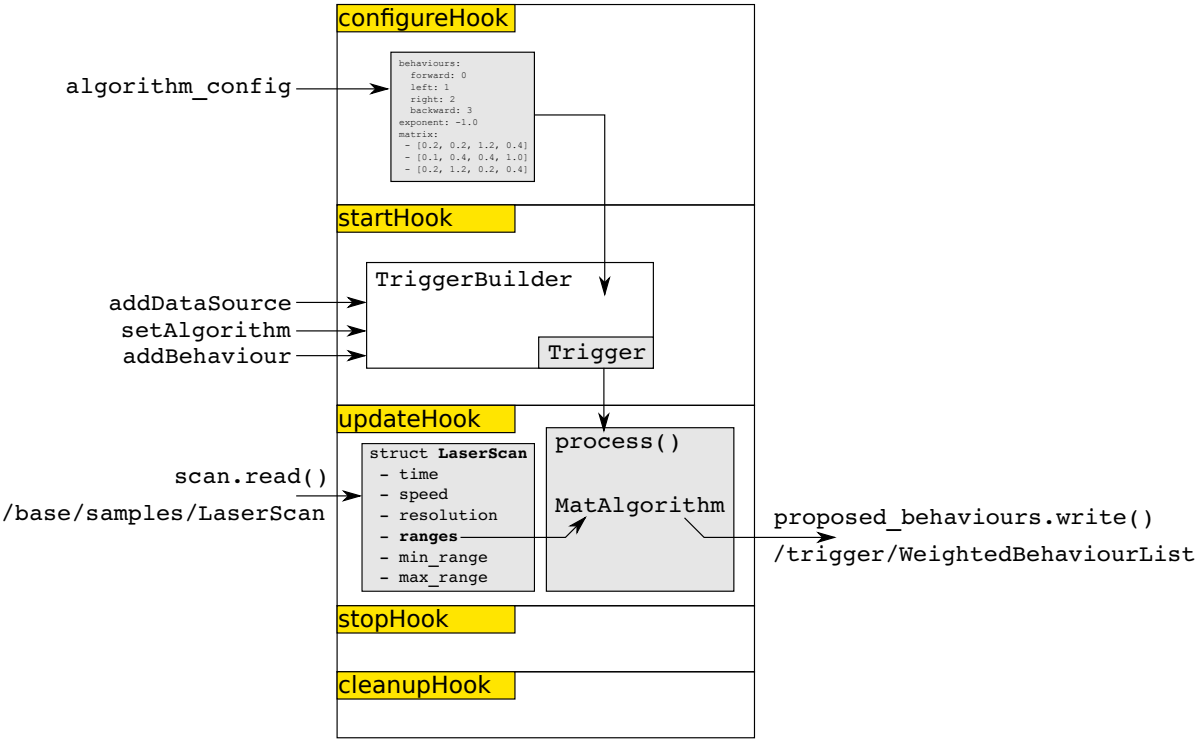


Abbildung 2.17 Verarbeitung innerhalb der LaserTrigger Komponente

können. Gleichzeitig sind diese Daten aber auch Voraussetzung für die weitere Verarbeitung und Gewinnung von Metainformationen wie das Vorhandensein bestimmter Objekte (wie beispielsweise anderer Roboter). Auch diese Metainformationen können dann wieder als Auslöser für bestimmte Verhalten verwendet werden und somit einen Beitrag zu dem gesamten Verhaltensmodell liefern.

Um also die Voraussetzung zur Verwendung dieser Daten zu liefern, wurde die Integration des Treibers für die IDS Kameras abgeschlossen. Zudem wurde ein Treiber für die PTU (Pan Tilt Unit) auf der die Kameras montiert sind, implementiert und ins System integriert. Ebenfalls können nun die Daten der Odometrie und der Fahrkommandos des Roboters ausgelesen und aufgezeichnet werden. Diese Daten sind speziell auch für die Entwicklung von Vorhersagemodellen notwendig. Zusammen mit den bereits implementierten und integrierten Treibern des SICK-Laser und der IMU (Lage und Beschleunigungssensor) können nun alle Daten der Sensoren verarbeitet und im LOB zur Verfügung gestellt werden.

Die *behave* Bibliothek In AP3200 wurde die neue *behave*-Softwarebibliothek eingeführt. Diese Bibliothek ist die Grundlage zur Realisierung des LoB-Modells innerhalb des Rock/Roby Frameworks. Sie stellt die Funktionalität bereit, die u.a. innerhalb der Ebenen des LoB verwendet wird.

Folgende Teile sind implementiert:

- Verhaltenspuffer (Behaviour Queue)
- Verhaltensdisponent (Contention Scheduler)
- Laufzeitintegration der Trigger

Trigger sind als *Action Mapper* umgesetzt, d.h. als Datenverarbeitungskomponenten, die beliebige Daten als Eingabe erhalten und eine Liste von Verhaltenskandidaten ausgeben - sie bilden Daten auf Verhalten/Aktion ab. Diese Kandidaten werden dann vom *Contention Scheduler* zu einem Satz auszuführender Verhalten aufgelöst.

Die von den Triggern ausgegeben Verhaltenkandidaten sind solche, die dazu geeignet sind, während des Betriebs ein erkanntes Problem zu beheben oder auf eine bestimmte Situation zu reagieren. Beispielsweise würde ein „Hindernis-Vorraus-Trigger" die Aktionen „Drehe-Links", „Drehe-Rechts" oder „Fahre-Rückwärts" nicht aber „Fahre-Vorwärts" vorschlagen. Die vorgeschlagenen Verhalten von allen aktiven Triggern werden zu einer Liste zusammengefasst, aus der ein Satz von auszuführenden Verhalten ermittelt wird (s. Abb. 2.18).

Die *behave*-Software baut im Wesentlichen eine Prioritätsstruktur auf, die von der Dispositionseinheit aufgelöst wird. Dabei verfolgt das System, was die Trigger überwachen, d.h. von welchen Teilen des Systems die Trigger aktiviert wurden. Diese eher abstrakten Aktionen (z.B. "go charge" oder "navigate to setpoint") haben eine Priorität, die die Wichtigkeit angibt. Die Priorität wird von den Trigger geerbt. Damit wird festgelegt, auf welche Aktionen sich das System konzentrieren soll. Ein Aufösungsalgorithmus ermittelt aus diesen Informationen den Satz Aktionen der ausgeführt werden soll. Zur Zeit wird nur der am höchsten gewichtete Trigger ausgewählt. Später können dafür auch andere Regeln herangezogen werden. Das wäre z.B. die Trigger nach der Priorität der aktivierenden Aktion zu sortieren, und dann soviel Verhalten wie möglich auzuführen, ohne dass Konflikte auftreten. Eine andere Vorgehsweise würde jedem Trigger eine Erfolgswahrscheinlichkeit zuordnen, und das auszuführenden Verhalten würde durch Maximierung der Erfolgswahrscheinlichkeit über alle Vorschläge bestimmt.

Um die Beziehung zwischen Trigger und Aktion zu verfolgen, werden die Trigger in die Abhängigkeiten der Aktion aufgenommen (s. Abb. 2.19 links). U.u. verwenden mehrere Aktionen Trigger, mit der gleichen Definition. In diesem Fall ist eine Variante, ein Trigger für mehrere Aktionen zu aktivieren (s. Abb. 2.19 rechts). Diese Variante wurde allerdings verworfen, da mit ihr die Information verlorgen ginge, dass eine Trigger für zwei Aktionen mit unterschiedlichen Prioritäten aktiv ist und somit auch zweimal mit den verschiedenen Prioritäten berücksichtigt werden muss. Dieser Zusammenhang könnte ohne diese Information nicht von der Dispositionseinheit aufgelöst werden. Das könnte ein Nachteil oder sogar eine Fehlerquelle sein, da die Robustheit des LoB in wesentlichen Teilen davon abhängt, wie die auszuführenden Verhalten ausgewählt werden.

In Roby können die Trigger entweder als Komponentennetzwerk oder als Anweisungsblock definiert werden. Beispielsweise würde folgende Anweisung mehrere Komponenten zusammen mit dem LaserTrigger-*ActionMapper* starten, der dann der LoB-Verarbeitungseinheit hinzugefügt wird:

```
scheduler.trigger LaserTrigger
```

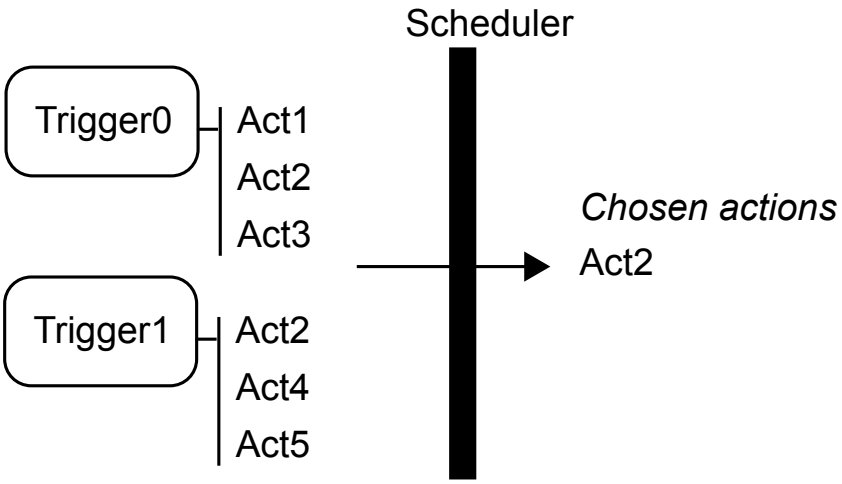


Abbildung 2.18 Aufbau des Triggerns in der *behave*-Software. Jeder Trigger ist ein Netz von Rock-Komponenten, die Daten verarbeiten und daraus eine (u.U. leere) Liste von Verhalten generieren, die ausgeführt werden sollen. Der Disponent wählt in jedem Verarbeitungs-/Zeitschritt den/die besten Aktionen aus den Vorschlägen aus. Dabei werden die Konflikte unter Berücksichtigung von zugewiesenen Prioritäten aufgelöst.

Dabei ist `LaserTrigger` definiert als:

```
composition LaserTrigger do
    add Srv::LaserRangeFinder, :as => 'sensor'
    add Triggers::LaserTas, :as => 'trigger'
    connet sensor => trigger

    export trigger.behaviours
    provides Src::BehaviourCandidates
end
```

Diese Definition legt fest das der `LaserTrigger` aus einem Laserentfernungsmesser (Laser Rang Finder) und dem Laser-Trigger selbst besteht. Diese Spezifikation wird dann benutzt, um

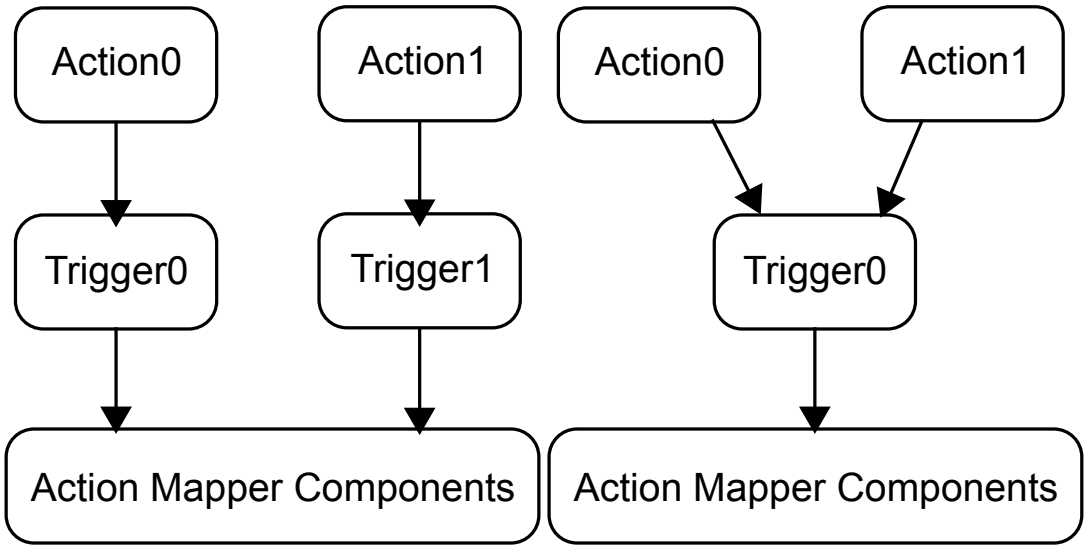


Abbildung 2.19 Bei der ausgewählte Umsetzung der Trigger-Aktions-Beziehung (links) sind die Trigger einer Aktion zugeordnet, die sie überwachen. Das erlaubt, die Wichtigkeit der Trigger zu bestimmen, und bietet wichtige Informationen für den Auflösemechanismus im Disponenten. Eine Alternative diese Beziehung darzustellen (rechts) ist, dass ein Trigger für mehrere Aktionen zuständig ist, wenn diese Trigger mit der gleichen Definition erfordern. Dabei kann jedoch die Priorität des Triggers nicht eindeutig ermittelt werden. Die tatsächliche Komponente, die die Daten verarbeitet, wird in beiden Fällen nur einmal ausgeführt.

den Disponente anzuweisen, diese Komposition als Trigger auszuführen.

2.1.4 AP 4300: Spez. Vorhersage und Selbstbewertung

Mögliche Verfahren zur Vorhersage und Selbstbewertung Im Folgenden werden mögliche Arten der Vorhersage und Selbstbewertung, die im Projekt angewandt werden könnten, vorgestellt.

Sensordatenvorhersage in der reaktiven Ebene mittels Clusteranalyse oder Support Vector Machine Zunächst soll auf die Vorhersage in der reaktiven Ebene nach Abbildung 2.20 eingegangen werden. Hier kann man von einer Vorhersage oder Klassifikation des Datenstroms pro Sensor sprechen, da die Datenströme einzeln ausgewertet werden und zu einer Vorhersage führen können. Zum Beispiel kann mit Hilfe der Analyse der vorhandenen IMU[1]-Daten eine Beschleunigung oder Richtungsänderung vorhergesagt werden.

In Abbildung 2.20 kann man die angesprochenen Sensordaten von IMU (Beschleunigung und Ausrichtung), Kamera, Laserscanner und ARIA[2] (Geschwindigkeit, Position und Orientierung) unter dem Begriff Afferenzen [3] wiederfinden. In der *Episodic Memory* könnten die zu bestimmten Datenströmen gehörenden Klassifikationen/ Vorhersagen abgespeichert werden um bei Änderung des Datenstroms abrufbar zu sein. Durch einen Abgleich der neuen Daten mit den gespeicherten Daten aus einer vorherige *Clusteranalyse* oder einer Gruppierung mittels *Support Vector Machines (SVM)* können einzelne Sensordatenströme schnell einer Gruppe zugeordnet und damit klassifiziert werden. Ein Cluster oder eine Gruppe von Daten hat dabei eine ähnliche Struktur und Bedeutung, zeigt also z.B. eine Drehung oder Fortbewegung des Roboters. Der um die Gruppenzugehörigkeit erweiterte Datenstrom wird dann zur Auswertung an das Monitoring-Modul weitergeleitet.
Die Motorsignale werden als Efferenzen bezeichnet[4]. Im Monitoring-Modul gibt es eine gelernte Liste von Motorimpulsen (Efferenzen) und den zugehörigen, erwarteten Sensordatenströmen

[1] *Internal Measurement Unit*
[2] *Advanced Robot Interface for Applications*
[3] Afferenz: Von den Sensoren zum zentralen Nervensystem laufende Nervenfasern zum Transport von Nervenimpulsen.
[4] Efferenz: Gegenstück zur Afferenz; leitet Muskel-/ Motorimpulse in die entgegengesetzte Richtung zum Sensordatenstrom.

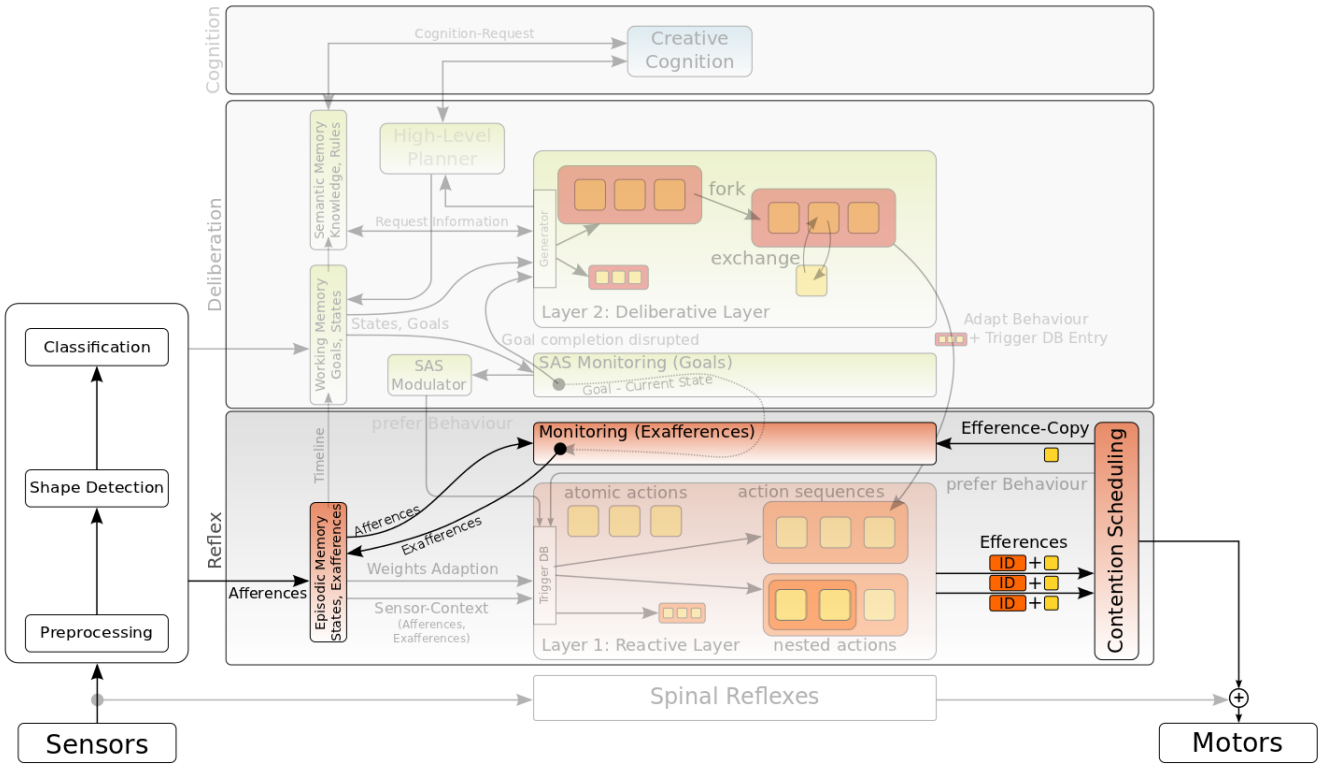


Abbildung 2.20 Die Vorhersage auf der reaktiven Ebene findet vor allem im Rahmen der Exafferenz-Berechnung statt. Einen Einfluß haben auch das "Episodic Memory" und der "Contention Scheduler".

(Reafferenzen[5]). Hier wird die Efferenz-Kopie mit der Reafferenz abgeglichen (Exafferenz[6]). Im Normalfall stimmt die Afferenz mit der Reafferenz überein und es muss nichts unternommen werden. Bei einer signifikanten Abweichung jedoch (wird z.B. der Motorimpuls "fahre vorwärts"gegeben und keine Fortbewegung im Sensordatenstrom erkannt) wird die Deliberationsebene angesprochen um das Problem zu lösen.

Sensordatenvorhersage in der reaktiven Ebene mittels Zeitreihenvorhersage Eine Alternative zur oben vorgestellten Sensordatenvorhersage mittels Clusteranalyse oder Support Vector Machine stellt die von Saegusa et al. ([63]) vorgestellte Sensordatenvorhersage dar. Dabei unterscheidet sich die Grundidee des hier beschriebenen Verfahrens nicht von der oben

[5] Reafferenz: Antwort des Sensors auf die Efferenz.
[6] Exafferenz: Unterschied zwischen Afferenz (Istzustand) und Reafferenz (Sollzustand).

beschriebenen Vorhersage. Es wird versucht, mit Hilfe des aktuellen Sensordatenstroms und der daraus abgeleiteten Aktion eine Änderung der Umgebung vorherzusagen. Dazu benutzen Saegusa et al. ein *Multi Layer Perceptron (MLP)* mit drei Ebenen und der *gradient method* als Lernverfahren des Netzes. In Simulationen und Experimenten an einem realen Roboter wird die Effizienz des Verfahrens gezeigt.
Der Vorteil dieses Verfahrens gegenüber dem vorherigen liegt darin, dass mehrere Zeitschritte vorhergesagt werden können. Dies wird von Saegusa et al. dadurch erreicht, indem jeweils das Ergebnis einer Vorhersage als Eingangswert für eine weitere Berechnung genommen wird. So werden gute Ergebnisse auch bei der Vorhersage von Zeitreihen erzielt. Es ist jedoch zu prüfen, ob die rechenintensivere Anwendung eines MLP und die anschließende Berechnung von Zeitreihen eine zu hohe Belastung darstellt. Zur Minimierung der benötigten Rechenleistung wäre denkbar, eine Zeitreihenvorhersage auf bestimmte Sensordaten zu beschränken.
Sollte es Probleme mit dem grundsätzlichen MLP-Verhalten geben, beispielsweise eine Mittelung über separat zu lernende Teilmengen der Trainingsdaten, so kann eventuell das Verfahren "Neural Gas" eine Lösung liefern [42]. Dabei handelt es sich um ein Vektorquantisierungsverfahren, das 1993 von Martinetz et al. vorgestellt wurde. Prinzipiell ist damit eine Anpassung an bzw. ein Lernen von beliebigen Datenverteilungen möglich. Die Anwendung zur Zeitreihenvorhersage ist in dem Artikel explizit beschrieben.

Zeitpunktvorhersage von Ressourcen Neben den vorgestellten Sensordatenvorhersagen ist es denkbar und sinnvoll, eine Zeitpunktvorhersage von Ressourcen durchzuführen. Dadurch soll z.B. die Restlaufzeit der Batterie vorhergesagt werden, um entsprechend Aktionen ableiten zu können. Dazu können entweder *Monte Carlo* Verfahren oder aber das von Liou et al. [39] vorgestellte, auf Wahrscheinlichkeitstheorie basierende, Verfahren zur Vorhersage verwendet werden. Hierbei kann die durchschnittliche Batterielaufzeit gelernt werden, aber auch ein erhöhter Energieverbrauch bei der Ausführung bestimmter Aktionen kann gelernt und bei der Vorhersage berücksichtigt werden.

Zeitreihenvorhersage im Zustandsraummodell Als alternativer Ansatz zu der genannten Sensordatenvorhersage in der reaktiven Ebene mittels Zeitreihenvorhersage soll ein auf *Vektorautoregressiven Prozessen* (VAR-Prozesse) und dem *Zustandsraummodell* (ZRM) [71] basierender Ansatz evaluiert werden. Die Annahme bei VAR-Prozessen ist, dass sich der Zustand eines Systems linear aus den vergangenen Zuständen vorhersagen lässt. Im Zustandsraummodell kommt zu dieser linearen Systemdynamik noch eine Beobachtungsfunktion hinzu,

da die System- bzw. Roboterzustände nicht direkt, sondern nur durch die zur Verfügung stehenden Sensoren und versehen mit etwaigem Beobachtungsrauschen beobachtet werden können. Um dieses Modellierungskonzept im Projekt zu verwenden, muss zunächst die Systemdynamik unter Verwendung von Testdaten geschätzt werden. Dies geschieht mit Hilfe von *Kalman–Filter* [31] und dem *Expectation-Maximization-Algorithmus* [13]. Anschließend lässt sich das Kalman–Filter zur Vorhersage der Systemzustände verwenden. Daraus lassen sich die erwarteten Sensordaten generieren, die dann wiederum entsprechend dem Reafferenzprinzip mit den tatsächlich gemessenen Sensordaten verglichen werden können.

Aufzeichnung von Testdaten Zum Test von Methoden zur Identifizierung von Objekten oder Erkennung anderer Muster in Sensordaten im allgemeinen ist es erforderlich, einen Datensatz zum Testen und zum Training der entsprechenden Algorithmen zu haben. Auch für die anderen Vorhersagemodelle ist es erforderlich, zunächst einmal die Struktur der Daten zu analysieren, die im späteren Betrieb zu erwarten sind. Auf der Basis dieser Erkenntnisse können dann sinnvolle Ansätze zur Vorhersage von Daten spezifiziert werden.

Zu diesem Zweck wurden Testdaten aufgezeichnet, die dann analysiert und zur Spezifikation sinnvoller Vorhersagemodelle verwendet werden können. Der Testdatensatz soll aber auch noch für andere Zwecke verwendet werden, wie zum Beispiel für das Training und zum Test von Objektklassifikationsalgorithmen oder zum Testen eines Detektors für optischen Fluss auf den Kameradaten. Da diese Daten zur Spezifikation der Vorhersage beitragen und im weiteren Verlauf des Projektes noch weiter Verwendung finden, wird hier der Versuchsaufbau sowie die aufgezeichneten Sensordaten und die geplante Verwendung kurz erläutert.

Versuchsaufbau Die Daten wurden in der Space Explorationshalle des DFKI RIC mit dem SeekurJr Roboter aufgezeichnet. Der SeekurJr wurde zu diesem Zweck vor einem Modell einer Wand eines Mondkraters aufgestellt. Auf dem Boden der Halle ist ein Granulat verteilt, welches die Oberflächeneigenschaften des Mondes nachempfindet. Der Aufbau ist in Abbildung 2.21 dargestellt. Während des Versuches befand sich dann noch ein Modell eines mehrfüßigen Laufroboters (SpaceClimber) als Beispielobjekt in der Nähe des SeekurJr.

Sensoren und aufgezeichnete Daten Der Aufbau der Sensoren auf dem SeekurJr wurde bereits in früheren Berichten erläutert. Allerdings wurde dieses mal erstmals der finale Aufbau mit den zwei IDS Kameras verwendet. Aufgezeichnet wurden die Daten der folgenden Sensoren:

Abbildung 2.21 Übersicht über Aufbau des Testszenarios

beider IDS Kameras, SICK-Laserentfernungsmesser, IMU (Lage- und Beschleunigungssensor), PTU (Pan Tilt Unit, auf der die Kameras montiert sind) sowie der Odometriedaten. Zusätzlich zu den Sensoren, die auf dem Roboter verbaut sind, wurde noch eine Markierung für das Vicon Verfolgungssystem, welches in der Explorationshalle montiert ist, angebracht. Somit kann die tatsächliche Bewegung des Roboters verfolgt und aufgezeichnet werden.

Für den bisher aufgezeichneten Datensatz wurde das SpaceClimber-Objekt an verschiedenen Stellen vor dem Krater platziert. Der SeekurJr wurde in einem Abstand von ca. 4 m positioniert. Während der Datenaufzeichnung hat sich der SeekurJr dann mit einer definierten (gleichbleibenden) Geschwindigkeit auf das Objekt zu und anschließend von dem Objekt weg bewegt. Für jede der gewählten Geschwindigkeiten (0.25, 0.5 und 1 m/s) wurde die Bewegung 10 mal wiederholt. Anschließend wurde ein weiterer Datensatz aufgezeichnet, bei dem der Roboter mittels Joystick manuell auf einer freien Bahn mit verschiedenen Geschwindigkeit gesteuert wurde. Bei diesem letzten Datensatz fuhr der SeekurJr mehrfach auf verschiedenen Bahnen auf das Objekt zu oder von diesem weg.

Verwendung der Daten Die aufgenommenen Daten sollen für verschiedene Zwecke verwendet werden. Bereits für die Analyse zur Spezifikation der Vorhersage war eine erste Sichtung und Auswertung der Daten wichtig. Nur so ließ sich ein erstes Bild der Eigenschaften der

Daten gewinnen und potentiell geeignete Vorhersagemethoden zu finden. Des weiteren sollen die Daten verwendet werden um Algorithmen zur Vorverarbeitung (wie z.B. Erkennung von optischem Fluss, oder Feature-Extraktion) sowie für Klassifikationsalgorithmen (wie z.B. Objekterkennung) verwendet werden. Besonders die ersten Datensätze mit der fest eingestellten Geschwindigkeit können sowohl für die Erkennung des optischen Flusses aber auch für die Vorhersagealgorithmen verwendet werden, da hier die "Wahrheit" über die tatsächliche Bewegung von außen durch das Verfolgungssystem (Vicon) gemessen wurde. Die Vorhersage könnte hier exemplarisch die zu erwartenden Sensorwerte für das Vorwärtsfahren lernen und zur Erstellung erster Auswertungen dienen. Der Datensatz mit der freien Bewegung hingegen dient hauptsächlich zur Extraktion verschiedener Einzelbeispiele der Sensordaten zum Training und zum Testen von Objekterkennung. Die ersten Datensätze können ebenfalls für die Analyse der Objekterkennung verwendet werden, da das Objekt die meiste Zeit im Sensorbereich des Roboters verblieben ist. Somit kann eine breite Auswahl an Analysen mit dem Datensatz realisiert und erste Ergebnisse zu den verschiedenen Anwendungen produziert werden.

Sowohl für die Auswertung der Vorhersage wie auch für die anderen Anwendungen sollen später noch weitere Datensätze aufgezeichnet werden. Beispielsweise ist auch die Aufnahme von Daten im Außenbereich des DFKI oder auf dem Testgelände geplant. Im Außenbereich ergibt sich lediglich die Schwierigkeit, den Roboter mit einem Verfolgungssystem zu beobachten, um die tatsächliche Bewegung messen zu können.

2.1.5 AP 4400: Real. Vorhersage und Selbstbewertung

Die Vorhersage von Sensorwerten ist entscheidend für die Bewertung des aktuellen Zustandes des Roboters. Dies bedeutet, dass das Vorhersagesystem in der Lage sein muss, auf Basis vergangener und aktueller Daten (wie z.B. Motorkommandos und Sensorwerten) die für einen oder mehrere Sensoren in naher Zukunft zu erwartenden Werte vorherzusagen. Im folgenden wird dieses Problem näher erläutert, Datensätze und Anwendungszenario skizziert sowie drei Ansätze zur Sensordatenvorhersage vorgestellt, die zu Vergleichszwecken integriert und implementiert wurden.

Lernen und Modellieren der Systemdynamik zur Vorhersage

Zur Vorhersage wird der Zusammenhang von Steuerkommandos und vergangener Sensorwerte auf der einen Seite und der vorherzusagenden Sensorwerte auf der anderen Seite hergestellt. Es werden drei Modelle vorgestellt, um diesen dynamischen Zusammenhang abzubilden. Hierbei unterscheiden sich die Ansätze in einigen Punkten. Im wesentlichen sind die Verfahren „Neural Gas“ und „Multilayer Perzeptron“ maschinelle Lernverfahren bei denen versucht wird eine möglichst genaue Repräsentation dieser Dynamik zu lernen. Ein weiterer etwas anderer Ansatz ist, die Dynamik analytisch in geschlossener Form mittels eines PT3-Gliedes zu beschreiben.

Allen Verfahren gemein ist hier, dass sie über Parameter beeinflusst werden um ein möglichst gutes Modell für die entsprechende Vorhersage zu liefern. Ein Beispiel welches im folgenden betrachtet wird, ist die Vorhersage des Wertes der IMU (hier, Rotationsgeschwindigkeit um die Z-Achse) auf Basis der Motorkommandos. Um nun ein Vorhersagesystem für die spezielle Aufgabe zu optimieren muss es mit Trainingsdaten auf das entsprechende Problem optimiert werden. Diese Optimierung kann mit Hilfe des Frameworks pySPACE realisiert werden. Das optimierte Vorhersagesystem wird dann auf Testdaten, die einen Fehlerfall enthalten, getestet und auf dem Roboter integriert um es unter realen Bedingungen ebenfalls zu evaluieren.

Sowohl für die Bewertung als auch für die Optimierung der Verfahren werden Datensätze benötigt, die bestimmten Kriterien genügen. Das Ziel der Lernverfahren als auch der Optimierung (ebenfalls des analytischen Ansatzes) ist es, ein Modell für die Vorhersage zu generieren, welches später auf dem Roboter für neue Situationen eine möglichst gute Vorhersage liefert. Beispielsweise müssen für verschiedene Bewegungen des Roboters mehrere definierte Wiederholungen aufgezeichnet werden, damit die Lern- und Optimierungsverfahren eine möglichst generelle Lösung finden die für neue Daten ebenfalls gut passend ist. Würde lediglich auf einem einzigen Versuch gelernt beziehungsweise optimiert werden, so würden Artefakte (Rauschen) das Ergebnis zu sehr beeinflussen und die Vorhersage keine guten Ergebnisse liefern. Auch für die Bewertung welche Qualität die Vorhersage in späteren unbekannten Situationen erreichen wird, muss auf verschiedenen Ausschnitten des Datensatzes getestet werden um den erwarteten Fehler zu ermitteln. Daher wurde ein Datensatz mit mehreren Wiederholungen aufgezeichnet der den Kriterien sowohl für Training, Optimierung und Bewertung erfüllt. Der Versuchsaufbau und die Daten werden im folgenden zunächst beschrieben und im Anschluss die Vorhersageverfahren.

(a) Rampe (b) Stein auf Kraterboden

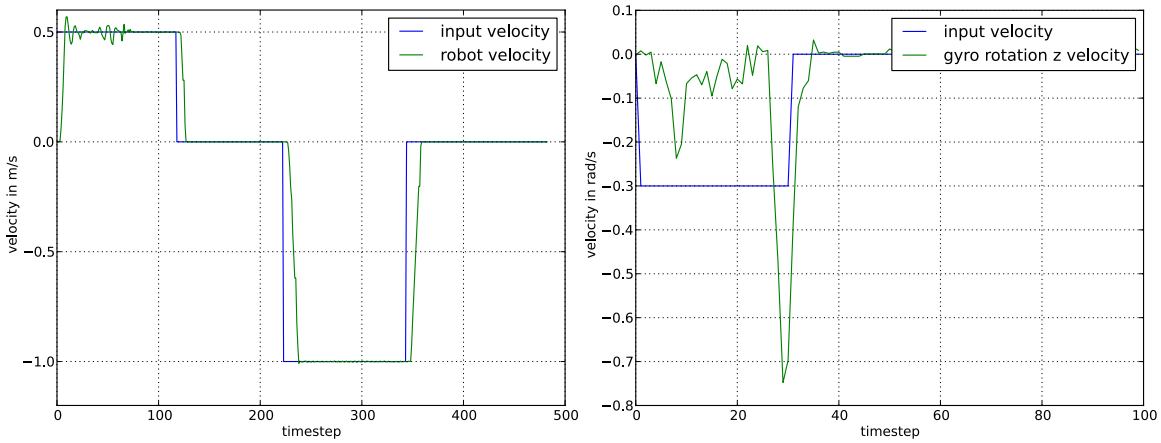
Abbildung 2.22 Die zu erkennenden Fehlerfälle während der Datenaufnahme

Sensordaten für Szenarien Der aufgenommene Datensatz beinhaltet zwei verschiedene Schwerpunkte. Wie in Abschnitt 2.1.5 bereits beschrieben, wird ein Datensatz benötigt, anhand dessen die Vorhersagemethoden trainiert, optimiert und bewertet werden können. Dies ist der erste Schwerpunkt und repräsentiert die Daten des *Normalfalles*. Dieser Datensatz repräsentiert die Sensordaten, die unter normalen Bedingungen erwartet werden. Der zweite Aspekt der untersucht werden muss ist die Möglichkeit die Vorhersage anschließend dazu zu verwenden, um während der späteren Mission des Roboters Fehlerzustände bzw. abweichendes Verhalten zu erkennen und gegebenenfalls zu korrigieren. Dieser zweite Schwerpunkt ist die Daten für den sogenannten *Fehlerfall*, bei denen getestet wird, ob sich die Vorhersage zum Zwecke der Systembewertung und Fehlererkennung nutzen lässt.

Für den Normalfall wurden je 10 Wiederholungen von vorwärts und rückwärts fahren sowie links und rechts drehen mit verschiedenen Geschwindigkeiten aufgezeichnet. Zusätzlich wurden diese Daten auf verschiedenen Untergründen aufgezeichnet, einmal auf Mondsand, auf glattem Hallenboden und dem Untergrund des Kratermodells.

Die in Abbildung 2.22 dargestellten Fehlerfälle sollen erkannt werden. Die Rampe (a) hindert den Roboter am Vorwärtsfahren, indem der Roboter den Kontakt zum Boden verliert und die Räder durchdrehen. Der Stein auf dem geneigte Kraterboden (b) hindert den Roboter am Drehen. Er blockiert die Drehung jedoch nicht komplett, da sich der Roboter bei ausreichender Leistung vom Stein wegdrücken kann.

Für die beiden Fehlerfälle müssen die entsprechenden Sensordaten (Translationsgeschwindig-



(a) Fahren auf Rampe (Roboterodometrie) (b) Drehen gegen Stein (Gyro der IMU)

Abbildung 2.23 Sensordaten bei Kontakt mit den Hindernissen

keit der Odometrie für Rampe und Rotationsgeschwindigkeit des Gyroskop der IMU für Stein) untersucht werden. Abbildung 2.23 zeigt eine Aufnahme der entsprechenden Sensordaten für die Zeiträume zu denen der Roboter die Hindernisse berührt. Im Plot für das Auffahren auf die Rampe (a) zeigt sich eine kleine Abweichung der tatsächlich gemessenen Translationsgeschwindigkeit (grün) von der Zielgeschwindigkeit (blau) nach dem Einschwingvorgang, bevor die Räder durchdrehen und keine Abweichung mehr gemessen werden. Bei der Drehung gegen den Stein (b) ist zu sehen, wie die Rotationsgeschwindigkeit (grün) anfangs ansteigt und dann rückläufig ist, nachdem die Drehung des Roboters durch den Stein gebremst wird. Nach einer kurzen Phase in der sich der Roboter am Stein vorbei schiebt, wird die Blockade aufgehoben und der Roboter beschleunigt.

Ein weiterer Aspekt, der auf Basis zuvor aufgezeichneter Sensordaten untersucht werden soll, ist die Möglichkeit, auf unerwartete Ereignisse einerseits zu reagieren aber sie andererseits auch zu lernen, falls diese zu einem gewünschten Verhalten gehören. In den bisher beschriebenen Datensätzen waren lediglich Normalfall und Fehlerfall enthalten, wobei hier der Fehlerfall ein unerwünschtes zu detektierendes Ereignis darstellt (z.B. Kollision mit einem Objekt). In dem nun beschriebenen Datensatz soll ein etwas anderer Fall untersucht werden, bei dem ein unerwartetes Ereignis auftritt, welches aber keinen Fehlerfall sondern lediglich einen bisher noch unbekannten Fall darstellt. Ein solcher Fall kann zum Beispiel auftreten, wenn der Roboter bisher nur auf horizontalem Untergrund operiert hat, nun aber beispielsweise eine Rampe hinunter fahren soll. Sowohl bei dem Herunterfahren einer Rampe als auch bei einem Fehlerfall, der auftritt wenn der Roboter droht in nachgebendem Boden einzubrechen, treten

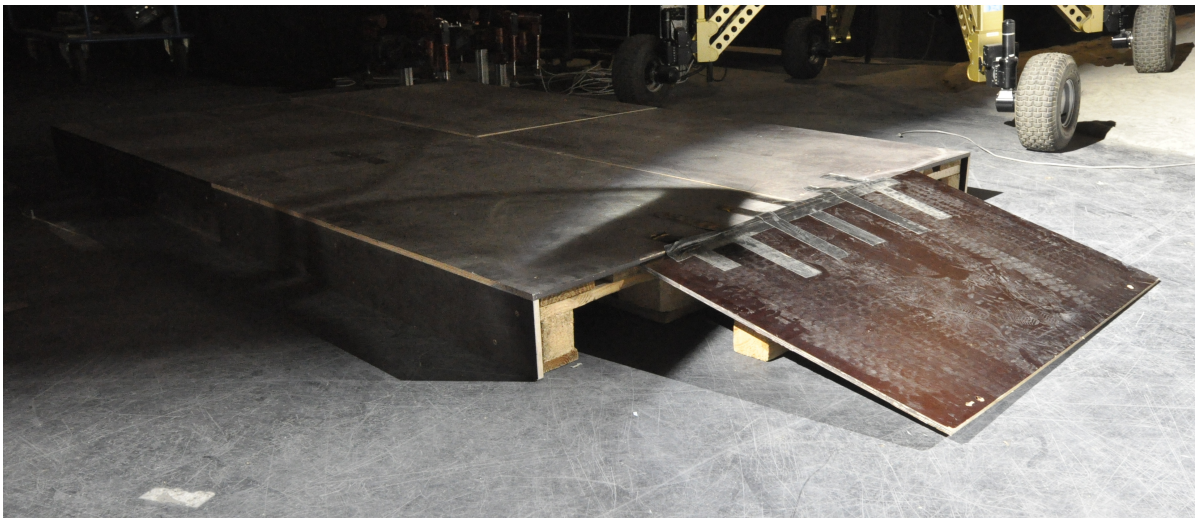


Abbildung 2.24 Szenario Rampe. Das Podest hat mit vier Paletten die Maße $164 \times 16 \times 244$ ($B \times H \times T$ in cm). Die Rampe hat die Maße 64×103 ($B \times L$ in cm).

zunächst relativ ähnliche Ereignisse in den Sensordaten auf. In beiden Fällen kippt der Roboter zunächst nach vorne und es ist mit entsprechenden Sensorwerten im optischen Fluss und der IMU zu rechnen, die sich deutlich von denen beim Fahren auf ebenem Untergrund unterscheiden. Hier wird nun zunächst auch ein großer Fehler entstehen, der zu dem Auslösen eines Fehler-Triggers führt. Auf der höheren Ebene im deliberativem Layer muss dann entschieden werden, ob es sich tatsächlich um eine Gefahrensituation handelt (z.B. nachgebender Boden) oder aber um eine Situation die in Zukunft als in diesem Zusammenhang normal betrachtet werden soll (z.B. Lernen des Herunterfahrens einer Rampe).

Um diese Aspekte untersuchen zu können und auch eine Vorhersage für den Fall "Rampe hinunter fahren" zu trainieren und zu testen, wurde ein neuer Datensatz aufgenommen. Für diesen Datensatz wurde der Roboter auf einem kleinen Podest positioniert, welches eine Höhe von ungefähr 16 cm hat. An dem Podest befindet sich eine Rampe, die in etwa einem Winkel von 14 zum Boden befestigt ist. Die Rampe ist länger als der Radstand des Roboters. So befindet sich der Roboter beim Herabfahren kurzzeitig komplett auf der Rampe. Das Podest und die Rampe sind in Abbildung 2.24 zu sehen.

Während die Daten aufgenommen wurden, ist der Roboter mehrfach mit verschiedenen Geschwindigkeiten die Rampe vorwärts hinunter und rückwärts wieder hoch gefahren. Somit sind auch für diesen Datensatz mehrere Wiederholungen verfügbar, um die Verfahren zu trainieren

und auf einem Testteil der Daten zu testen. Der Hintergrund in diesem Versuch stellt wieder das Modell des Mondkraterrandes dar.

Zusätzlich wurden ebenfalls noch Daten aufgenommen für einen Fall, der nicht als Normalfall gelernt werden soll. Zu diesem Zweck ist der Roboter ebenfalls mehrfach von dem Podest herunter gefahren, jedoch über davor liegende Kanthölzer. Dies soll das Einbrechen auf nachgebendem Boden darstellen und sollte sich in den Sensordaten unterscheiden. Diese Daten sollen auch genutzt werden um die Entscheidung vor dem tatsächlichen Herunterfahren des Roboters im deliberativen Layer zu definieren.

Da später auch die Komponente zur Berechnung des optischen Flusses in den Kamerabildern fertiggestellt wurde, ist diese nachträglich auf die aufgezeichneten Daten angewendet worden. Damit die Werte für optischen Fluss ebenfalls zur Vorhersage und Fehleranalyse verwendet werden können, wurden diese nachträglich generiert. Hierzu wurden die zuvor aufgenommenen Kameradaten wieder abgespielt, an die Komponente für den optischen Fluss weitergeleitet und dort verarbeitet.

Methoden zur Sensordatenvorhersage

Im diesem Abschnitt werden nun die bereits erwähnten drei Methoden NG, MLP und PT3 vorgestellt.

Analytische Modellierung durch PT3-Glied In Abbildung 2.25 dargestellt ist der Einschwingvorgang für die Translationsbewegung des Roboters auf ebenem Boden ohne Störungen. Die Sprungantwort des Roboters (grün) lässt vermuten, dass es sich dabei um ein Masse-Dämpfer-System handelt, welches durch ein PT2-Glied beschrieben werden kann. Mit einer zusätzlichen Verzögerung durch die Signallaufzeit ergibt sich daraus ein PT3-Glied. Gleichung 2.4 gibt die Übertragungsfunktion eines PT3-Gliedes.

$$X_{out}(s) = \frac{1}{(1 + 2DT_1 \cdot s + T_1^2 \cdot s^2) \cdot (1 + T_2 \cdot s)} \cdot X_{in}(s) \tag{2.4}$$

Gesucht sind nun die drei Parameter D, T_1, T_2 für die der Vorhersagefehler im Normalfall für das gesamte Trainingsset minimal wird.

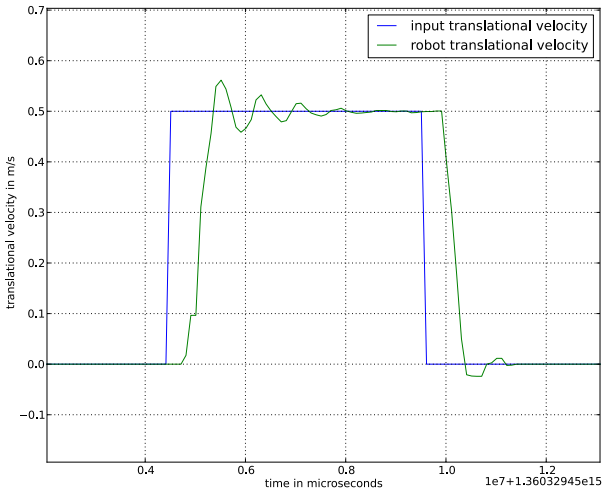


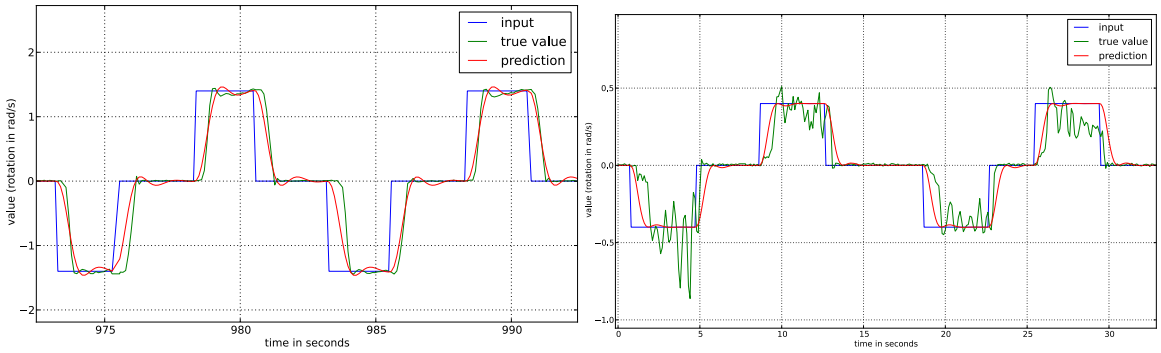
Abbildung 2.25 Einschwingverhalten des Roboters bei Translationsbewegung auf dem Boden

Das Trainingsset besteht aus 10 typischen Bewegungen (Translation oder Rotation) in beide Richtungen für drei diskrete Geschwindigkeiten. Pro Szenario ergeben sich damit 60 Beispiele bzw. 120 Sprungfunktionen für den Normalfall (jeweils zur Zielgeschwindigkeit und zur Ausgangsgeschwindigkeit).

Vorhersage für den Normalfall: Das Vorhersagemodell wird für das Dreh-Szenario trainiert, da hierbei die Rotation des Roboters und der IMU unabhängig voneinander sind. Das Rutschen und im Extremfall das Durchdrehen der Räder ließe sich in der Roboterodometrie nur schwer feststellen, da diese von der Drehung der Räder und nicht der tatsächlichen Drehung in der Welt ausgeht.

Die Vorhersage des PT3-Glieds mit optimalen Parametern wurde für das Drehen auf ebenem Boden und auf dem geneigten Kraterboden ohne zusätzliche Störung untersucht. Die Vorhersage für beide Normalfälle ist in Abbildung 2.26 dargestellt. Das Vorhersagemodel wurde für den Boden und den Krater separat trainiert.

Für den Normalfall auf dem Boden ohne jegliche Störung bildet das Vorhersagemodell die tatsächliche Drehung der IMU gut ab. Das Einschwingverhalten, welches beim Drehen wesentlich geringer ausgeprägt ist, wird nachgebildet. Für die Drehung auf dem Kraterboden ergeben sich wesentlich verrauschtere Sensorwerte für das Gyroskop. Die Unebenheiten des Krater bewirken, dass das Roboter sehr oft in seiner Drehung gebremst wird. Das Optimieren der Modelparameter des PT3-Glieds über mehrere dieser Beispiele führt zur Mittlung über



(a) Boden, ±1.4 rad/s (b) Krater, ±0.4 rad/s

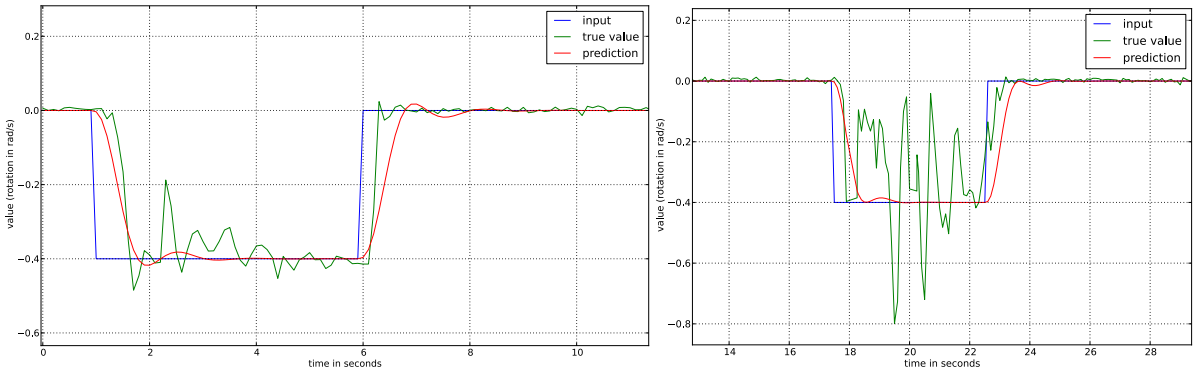
Abbildung 2.26 Normalfall ohne Hindernisse für Drehung auf dem Boden (a) und Krater (b)

die Sensorwerte des Gyroskops. Das Resultat ist eine Vorhersage, die die Charakteristik des Überschwingverhaltens nur noch sehr schwach abbildet. Es kommt damit bereits zu Vorhersagefehlern im Normalfall auf dem Krater, da sich die hohe Dynamik des Gyroskops auf dem Krater nicht auf das PT3-Glied abbilden lässt.

Vorhersage für den Fehlerfall: Um den Vorhersagefehler für die Fehlerfälle in den Szenarien (Drehung auf Boden und Krater) zu testen, wird die im jeweiligen Normalfall gelernte Vorhersage auf einen entsprechenden Fehlerfall angewendet. Das Verhalten des Gyroskops bei Drehung gegen Hindernisse auf Boden und Krater ist in Abbildung 2.27 dargestellt. Als Hindernis wurden mehrere 20 kg schwere Gewichte auf dem Boden platziert (a) und der Roboter dagegen gedreht. Die Beschleunigungsphase ohne Hindernis-Kontakt ist hierbei sehr kurz; bereits nach 1 s wird die Rechtsdrehung des Roboters kurz gedämpft. Danach bewegen sich Hindernis und Roboter relativ auseinander und der Roboter dreht sich ungebremst weiter. Auf dem Krater kommt der bereits bekannte Stein aus Abbildung 2.22 (b) zum Einsatz. Auch hier ist zu erkennen, dass kurz nach der Beschleunigungsphase die Drehbewegung kurzzeitig für ca. 1 s blockiert wird. Hierbei drückt sich der Roboter vom Stein weg und bewegt sich danach ohne zusätzliche Blockade weiter. In der Phase danach ist wieder die hohe Dynamik in den Gyroskop-Werten, hervorgerufen durch die Unebenheiten im Krater, ersichtlich.

Implementierung:

Das in Gleichung 2.4 dargestellte Übertragungssystem wird dann in den Zustandsraum übertragen, der durch die beiden Gleichungen



(a) Boden, 4 Gewichte á 20 kg (b) Krater, fest montierter Stein

Abbildung 2.27 Drehung auf gegen Hindernisse auf dem ebenen Boden und dem Krater

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \tag{2.5}$$

$$y(t) = C \cdot x(t) + D \cdot u(t) \tag{2.6}$$

beschrieben wird. Dabei beschreibt Gleichung 2.5 die Zustandsänderung des Systems durch die Systemmatrix A und durch das Eingangssignal u und der Matrix B. Gleichung 2.6 gibt die Systemantwort y im aktuellen Zustand bei anliegendem Eingangssignal u.

Diese als Orocos-Komponente implementierte Vorhersage wird mittels dieser 4 Matrizen konfiguriert. Damit ist die Vorhersage-Komponente nicht nur auf die Abbildung von PT3-Glieder beschränkt, sondern lässt sich allgemein auf Übertragungssysteme anwenden, deren Zustandsraummodel bekannt ist.

Für die Vorhersage-Komponente wurden die Parameter

$$A = \begin{bmatrix} -4.90557171 & -23.25811384 & -38.11203093 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 \\ 0 \\ 38.11203093 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

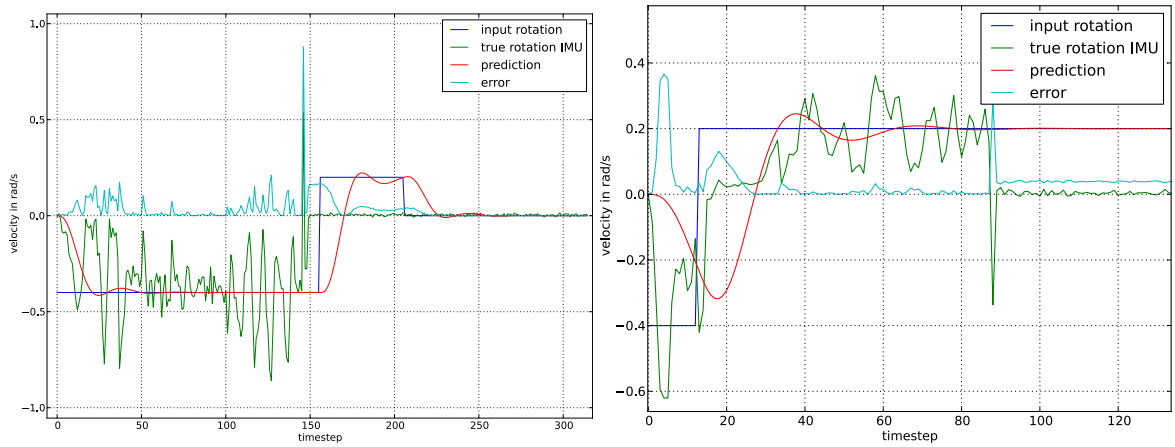
verwendet, welche das System auf ebenem Boden beschreiben.

Innerhalb der Vorhersage-Komponente wird bereits der Vorhersagefehler ausgewertet. Dabei werden die letzten 5 Vorhersagefehler, welcher über den mittleren quadratischen Fehler (MSE) von Vorhersage und tatsächlichem Sensordatum gebildet wird, gemittelt.

Abbildung 2.28 zeigt den Fehler der Vorhersage auf dem Boden (a) und auf dem Krater (b) beim Drehen gegen ein Hindernis. Der Roboter wurde angewiesen sich mit der Rotationsgeschwindigkeit von 0.4 rad/s im Uhrzeigersinn gegen ein Hindernis zu drehen. Überschritt der gemittelte Vorehrsagefehler einen bestimmten Schwellwert im Trigger, wird dort das Verhalten vorgeschlagen, das den Roboter mit 0.2 rad/s in die entgegengesetzte Richtung drehen lässt.

Beim Drehen auf dem Boden (a) wurde der Roboter durch manuelles festhalten kurzzeitig blockiert. Der Schwellwert im Trigger ist auf 0.5 gesetzt, wodurch, kurz nachdem der Fehler den Schwellwert übertrifft (nach etwa 145 Zeitschritten), die Rotation in die entgegengesetzte Richtung ausgelöst wird.

Wird die Vorhersage für den Normalfall auf dem ebenen Boden auf den Krater angewendet (b) ergibt sich bereits im Einschwingvorgang ein größerer Vorhersagefehler der höher als der hier gesetzte Schwellwert von 0.3 ist. Die Blockierung durch den Stein (ca. Zeitschritt 6 bis 14) findet sich nicht im Vorhersagefehler wieder. Der Trigger reagiert demzufolge bereits im Normalfall auf dem Kraterboden mit der entgegengesetzten Rotation.



(a) Manuelles Blockieren auf Boden (b) Blockieren durch Stein auf Krater

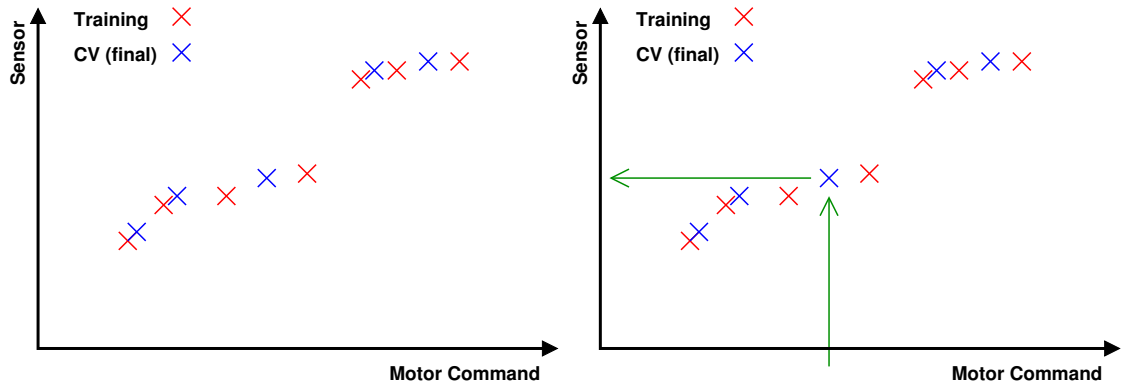
Abbildung 2.28 Test der Vorhersage-Komponente auf Boden und Krater

Vorhersage mittels Lernverfahren "Neural Gas" In diesem Abschnitt wird das Verfahren NG im Detail beschrieben.

Prinzip Ein Lernverfahren, das im Rahmen dieses Arbeitspaketes untersucht wurde, ist das Verfahren "Neural Gas". Es wurde 1991 von Thomas Martinetz und Klaus Schulten vorgestellt [43]. In [42] wird ein Beispiel für die Verwendung für Zeitreihenvorhersagen beschrieben. Eine relativ neue Erweiterung ist das Verfahren "NGPCA", das eventuell auch für die Vorhersage eingesetzt werden kann. Wie beim Lernverfahren "MLP" (siehe eigener Abschnitt) wird kein explizites Modell benötigt. Mittels "Neural Gas" können prinzipiell beliebige Datenverteilung gelernt, d.h. sogenannte "Center-Vektoren" daran angepasst werden. Damit besteht insbesondere eine Toleranz gegenüber Mehrdeutigkeiten in den Trainingsdaten. Als weiterer Vorteil ist es bei diesem Verfahren relativ leicht möglich, Vorhersagen für verschiedene (auch getrennt gelernte) Verhalten zu kombinieren. Auch der Wechsel zwischen unterschiedlichen Konfigurationen ist leicht möglich.

Ein Nachteil ist das generelle Problem bei Lernverfahren, dass die Anwendbarkeit auf ein Problem sich nicht apriori bestimmen lässt. Im Rahmen der in diesem Arbeitspaket durchgeführten Untersuchungen konnte die Anwendbarkeit von "Neural Gas" für das Vorhersageproblem jedoch festgestellt werden.

Verglichen mit MLPs könnte es einen höheren Aufwand beim Recall, das heißt bei der An-



(a) Die Center-Vektoren passen sich beim Lernen den Trainsdaten an.

(b) Abruf der gelernten Sensorwerte mit Hilfe des Motorkommandos.

Abbildung 2.29 Das Lernverfahren "Neural Gas".

wendung des gelernten Vorhersagemodells, geben. Dies muss in noch durchzuführenden Vergleichen untersucht werden. Gleiches gilt für den Lernaufwand.

Beim Lernen werden die initial zufällig verteilten Center-Vektoren schrittweise den Trainingsdaten angenähert. Dabei werden Vektoren, die nah an einem Datum sind stärker korrigiert als weiter entfernte. Nach einer ausreichenden Anzahl von Lernschritten sollten sich die Center-Vektoren an die zu lernenden Daten angepasst haben (siehe Abbildung 2.29(a)).

Für die Vorhersage wird das bekannte Motorkommando zur Suche des am besten passenden Center-Vektors verwendet. Der Center-Vektor mit dem geringsten Abstand wird für die Generierung der Vorhersage verwendet. Der jeweils gesuchte (erwartete) Sensorwert kann direkt über Center-Vektor abgelesen werden. Für die Generierung der Vorhersagen ist so die Suche des korrekten Center-Vektors der größte Aufwand.

Vorhersage mittels MLP Das *Multilayer Perceptron* (MLP) ist ein Lernverfahren aus dem Bereich der Neuronalen Netze. Es handelt sich hierbei um ein bekanntes Feed-forward Netz, welches ein genereller Funktionsapproximator ist. Die Flexibilität des Netzwerkes wird durch eine oder mehrere versteckte Schichten erreicht. Die Schichten sind mit der jeweils folgenden voll vernetzt, dies bedeutet, dass jedes Neuron einer Schicht mit jedem Neuron der Folgeschicht verbunden ist. Die Verbindungen zwischen den Neuronen können unterschiedlich "stark" sein, dies wird durch Gewichtswerte repräsentiert. Eine schematische Darstellung des Aufbaus eines MLP findet sich in Abbildung 2.30. Die Eingabeschicht x erhält den Eingabe-

vektor $\vec{x}$ (der in diesem Anwendungsfall die Sensordaten enthält). Jedes Element $x_i \in \vec{x}$ des Eingabevektors wird von einem Neuron der Eingabeschicht repräsentiert und von diesem unverändert übernommen. Für die Aktivierung der Neuronen in einer Schicht werden alle Ausgaben der Neuronen der vorigen Schicht mit dem Gewichtsvektor des aktuellen Neurons multipliziert. Die Ausgabe des Neurons für die nächste Schicht bestimmt eine interne Aktivierungsfunktion die auf das Ergebnis der Vektor-Multiplikation angewendet wird, im Falle der Hidden Layer ist dies in der Regel eine sigmoide Funktion (z.B. tanh). In der Ausgabeschicht ist die Aktivierung häufig eine lineare Funktion, damit das Netz prinzipiell jeden Wertebereich der Ausgabe erfassen kann. Die Aktivierung der Ausgabeschicht ist dann das Ergebnis (die Vorhersage) des Netzes.

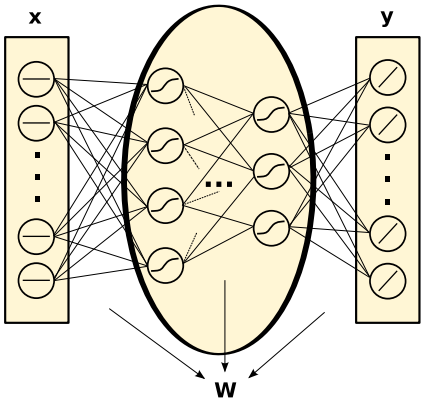


Abbildung 2.30 Schematische Darstellung eines MLP

Die Gewichte des MLP werden im allgemeinen durch eine Gewichtsmatrix **W** beschrieben. Somit lässt sich die Funktionsweise eines MLP mit der folgenden allgemeinen Beschreibung darstellen:

$$\vec{t} = y(\vec{x}, \mathbf{W}) \tag{2.7}$$

Hierbei ist $\vec{t}$ der tatsächlich für die Eingabe $\vec{x}$ gemessene Ergebnisvektor. Die grundlegende Annahme ist nun, dass die Werte für $\vec{t}$ einem funktionalen Zusammenhang zu $\vec{x}$ unterliegen, der durch die Funktion $y()$ abgebildet werden kann. Die dann in einem Versuch tatsächlich gemessenen Werte für $\vec{t}$ unterliegen in der Realität aber noch weiteren, in der Regel nicht vollständig zu beschreibenden, Störeinflüssen. Das Ziel ist also, die Funktion $y()$ so zu wählen, dass die Abweichung zu den tatsächlichen Werten von $\vec{t}$ im Mittel so gering wie möglich wird.

Für das MLP bedeutet dies nun die Gewichtsmatrix **W** anhand von Trainingsbeispielen so zu lernen, dass für diese der mittlere Fehler so gering wie möglich wird. Im mathematischen

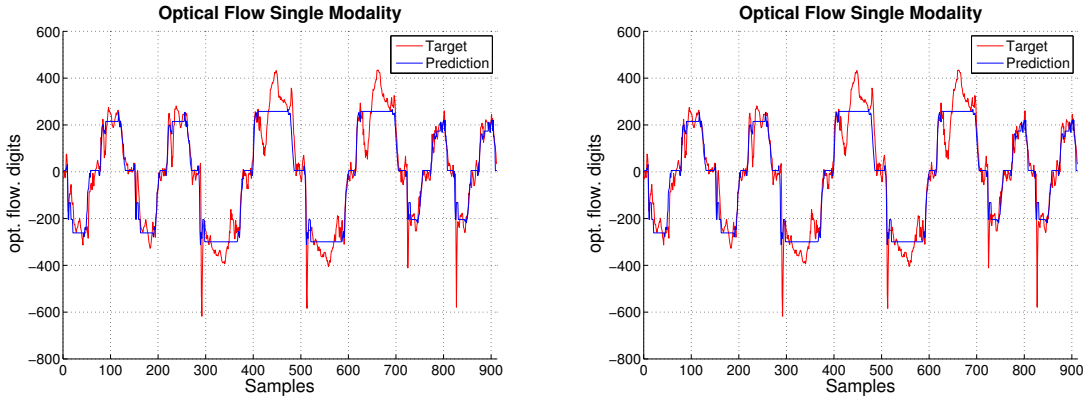
Sinne bedeutet dies, dass eine Fehlerfunktion minimiert werden soll. Eine gängige hierfür verwendete Fehlerfunkton ist der mittlere quadratische Fehler, welcher durch die folgende Funktion gegeben ist:

$$E(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N} \|y(x_n, \mathbf{W}) - t_n\|^2 \tag{2.8}$$

Für das MLP existieren hier mehrere Verfahren um während des Trainings die Gewichtsmatrix $\mathbf{W}$ so zu wählen, dass die Fehlerfunktion minimiert wird. Einer der bekanntesten und ersten Algorithmen zum Training eines MLP ist das sogenannte *Backpropagation* welches von Rummelhart et al. 1986 in [60] beschrieben wurde. Dieses Verfahren beruht auf einem Gradientenabstiegsverfahren welches den Fehler der letzten Schicht auf die vorhergehenden Schichten zurück propagiert. Für die folgenden Analysen wurde die bereits erwähnte openANN Bibliothek verwendet und die in dieser implementierte Weiterentwicklung des ursprünglichen Backpropagation Ansatzes verwendet. Unter Anderem kam hier der als *RPROP* bezeichnete Algorithmus zur Fehlerminimierung zur Anwendung der 1993 von Riedmiller et al. in [58] beschrieben wurde.

Zusätzlich diesen Analysen für das MLP haben wir zusätzlich die Möglichkeit getestet auch mehrere Sensorkanäle, also multi-modale Daten, vorherzusagen. Hierbei hat das MLP auf Basis der Motion Commands der letzten 15 Zeitschritte eine Vorhersage für die Werte der Z-Achsenrotation der IMU, der Geschwindigkeit für linkes und rechtes Rad sowie den horizontalen Anteil des optischen Flusses vorhergesagt. Hierbei hat sich herausgestellt, dass aufgrund der stark unterschiedlichen Wertebereiche der Sensoren und somit der Zielwerte der einzelnen Neuronen in der Ausgabeschicht des MLP bei Hinzunahme des optischen Flusses auch die Vorhersage der übrigen Sensoren deutlich verschlechtert. Eine Lösung dieses Problems ist die Normierung der Sensordaten, insbesondere des optischen Flusses, hierzu werden auf der Trainingsdatenmenge der Mittelwert und die Varianz der auftretenden Sensorwerte berechnet. Sowohl die Trainingsdaten als auch die später verwendeten Testdaten werden dann auf diese Werte normiert, indem von allen Werte der Mittelwert abgezogen wird und anschließend durch die Varianz geteilt wird. Diese Abbildung muss dann natürlich für die finale Ausgabe der Vorhersage wieder umgekehrt werden. Erst nach dieser Normierung war es mit dem MLP möglich die Sensorwerte für die verschiedenen Sensoren sinnvoll vorherzusagen.

Zudem wurde noch eine weitere Optimierungsmethode für das MLP getestet, da der normale Gradientenabstieg insbesondere für die Vorhersage des optischen Flusses keine sehr guten Ergebnisse geliefert hat. Das hier zur Optimierung eingesetzte Verfahren basiert auf den Veröffentlichungen von Levenberg und Marquardt in [37, 41]. Zudem wurde für die multi-modale



(a) (b)

Abbildung 2.31 Auf der linken Seite ist das Ergebnis für die Vorhersage des optischen Flusses dargestellt, mit einem MLP welches 2 hidden layer zu je 10 Neuronen enthielt und ausschließlich den optischen Fluss vorhersagt. Auf der rechten Seite ist das Ergebnis dargestellt welches mit einem MLP erzeugt wurde, welches ebenfalls aus 2 hidden layers aber mit je 20 Neuronen bestand. Im zweiten Fall wurde zudem mit dem gleichen MLP direkt auch die Z-Achse der IMU und die Geschwindigkeiten für die linken und rechten Räder vorhergesagt.

Vorhersage ein Netz mit zwei hidden Layer und jeweils 20 Neuronen pro Layer verwendet.

Die Vorhersagen der bereits dargestellten Modalitäten (Z-Achse IMU, linke und rechte Radgeschwindigkeit) wurden durch die multi-modale Vorhersage kaum beeinflusst (zumindest nach der Normalisierung der optischen Fluss Werte). Insgesamt ist die Vorhersage der Werte des optischen Flusses rein basierend auf den vergangenen Motorkommandos jedoch relativ schwierig. Hier entstehen größere Fehler als bei den anderen Modalitäten, da die Werte des optischen Flusses deutlich stärker schwanken. Der Vergleich wird in einem späteren Abschnitt noch genauer behandelt. Auch die Vorhersage für den optischen Fluss wird erwartungsgemäß leicht verschlechtert, wenn ein Netz verwendet wird, welches mehrere Modalitäten gleichzeitig vorhersagt. In Abbildung 2.31(a) ist die Vorhersage eines MLPs dargestellt, welches lediglich die optischen Fluss Daten vorhergesagt hat. Der RMS-Fehler beträgt hier 66.91 digits. In Abbildung 2.31(b) wird die Vorhersage auf dem gleichen Testdatensatz, jedoch mit einem Netz welches zusätzlich noch weitere Modalitäten vorhergesagt hat, dargestellt. Der RMS-Fehler im zweiten Fall beträgt 68.44 digits. Gemessen an den Wertebereichen ist der Unterschied hier also sehr gering, jedoch im Fall der Single-Modalität etwas besser.

Integration Ein Vorhersagemodul welches das MLP verwendet wurde in das Framework integriert und mittels Konfigurationsdatei so konfiguriert, dass es einem Netzwerk entspricht, welches vorher auf Trainingsdaten trainiert wurde. Das Vorhersagemodul generiert auf Basis der letzten 15 Motorkommandos sowie den aktuellen Odometriewerten für linke und rechte Radgeschwindigkeit eine Vorhersage für die Gyroskope der IMU. Hierbei werden alle drei Gyroskopwerte vorhergesagt, also die Drehung um die X, Y und Z Achse des Roboters. Bei der Bewegung auf flachem Untergrund ist hier allerdings nur mit einer relevanten Drehung um die Z Achse zu rechnen während die X und Y Werte nur leicht um den Null-Wert rauschen.

Zusätzlich zu dieser Vorhersage wurde ein weiteres Netz trainiert und als Vorhersagemodul in das Framework integriert, welches die Standardabweichung für die Sensorwerte (IMU) vorhersagt. Zu diesem Zweck wurde das Netz zuvor mit Trainingsdaten trainiert in denen die tatsächliche Standardabweichung der vorherzusagenden Werte (hier IMU Gyroskop) aus den letzten 7 dem aktuellen und den nächsten 7 Werten enthalten ist.

Diese zwei Vorhersagemodule können nun verwendet werden um im Rahmen der Selbstbewertung des Systems auf Abweichungen vom erwarteten Zustand zu reagieren. In den vorangegangenen Versuchen ist deutlich geworden, dass ein Trigger der ausschließlich auf den vom ersten Vorhersagemodul vorhergesagten Sensorwerte der Gyroskope schwierig zu realisieren ist, da zu Beginn und zum Ende einer jeden Bewegung recht große Fehlerwerte entstehen können. Dies hat damit zu tun, dass die Flanke im Signalverlauf an diesen stellen recht steil ist, obwohl der Verlauf der Kurve durch die Vorhersagemethoden relativ gut beschrieben wurde reicht hier schon eine kleine Zeitliche Abweichung des tatsächlichen Sensorwertes um eine relativ große Abweichung zu erzeugen. Daher erscheint es sinnvoller einen Trigger zu verwenden, der in der Lage ist in bestimmten Situationen wie z.B. zu Beginn oder am Ende einer jeden Bewegung eine höhere Abweichung zu akzeptieren als zu den konstanten Zeitintervallen bei denen in der Regel eine deutlich geringere Abweichung auftreten sollte. Zu diesem Zweck kann die zuvor beschriebene Vorhersage der zu einem Zeitpunkt zu erwartenden Standardabweichung der Sensorwerte verwendet werden. In Abbildung 2.32 wird dies exemplarisch für einen Ausschnitt von Messwerten aus einem aufgezeichneten Versuch dargestellt. In dieser Abbildung zeigt sich, dass bereits die Standardabweichung als Entscheidungskriterium für einen Trigger relativ gute Ergebnisse erzielen könnte, jedoch auch zu einigen "Fehlern" führen würde. Dies hat sich auch in praktischen Tests auf dem Roboter bestätigt. Die Standardabweichung bezieht sich auf das "Rauschen" der Sensorwerte, und die Abweichungen die zu einem bestimmten Zeitpunkt im Verhältnis zu den gegebenen Steuerkommandos auftreten.

Da im allgemeinen angenommen wird, dass ein solches Rauschen normal verteilt ist, erfasst

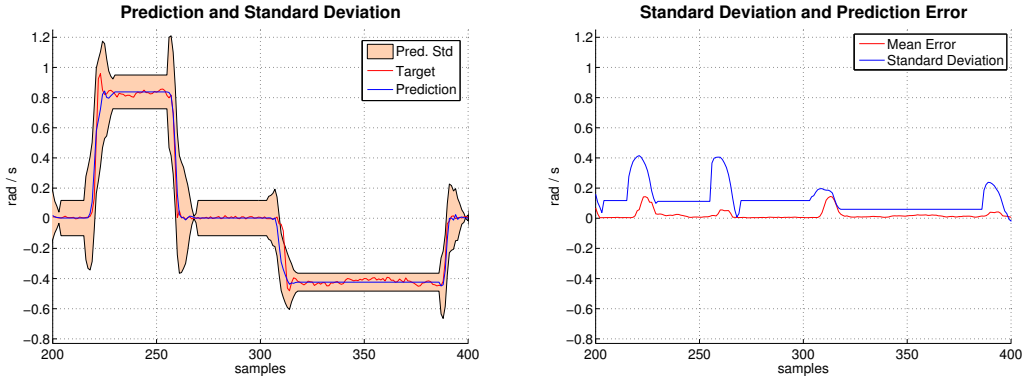


Abbildung 2.32 Diese Abbildung zeigt links einen Plot der Vorhersage, sowie der tatsächlich gemessenen Sensorwerte. Der Bereich um die Vorhersage ist die von einem zweiten Netz vorhergesagte Standardabweichung der Messwerte, mit der zu dem entsprechenden Zeitpunkt gerechnet werden muss. Auf der Rechten Seite zeigt der Plot die zugehörigen Kurven des Vorhersagefehlers, der über jeweils die letzten 5 Samples gemittelt wurde sowie noch einmal den Wert der Standardabweichung als Vergleich.

der Bereich der Standardabweichung etwa 68% der Fälle, daher wäre als Trigger Kriterium z.B. das 1.5 Fache der Standardabweichung möglicherweise sinnvoller. Es muss nun in praktischen Tests ermittelt werden, wie sich dieser Faktor verhalten muss, damit möglichst selten im Normalfall ein Fehlertrigger auslöst, aber mit ausreichender Sicherheit im Fehlerfall reagiert wird.

Spätere Anpassungen: Bei der späteren Evaluation der Vorhersage auf dem Robotersystem und der Integration weiterer Vorhersagen, wie beispielsweise die Vorhersage der Bewegung eines Objektes im Kamerabild auf Basis der eigenen Bewegung, viel auf, dass an dem System noch Anpassungen erforderlich waren. Für den Fehlertrigger ist nämlich eigentlich nicht relevant, wie die Standardabweichung des original Signals aussieht, sondern der Fehler der zum aktuellen Zeitpunkt von der Vorhersage zu erwarten ist. Es gilt zwar nach wie vor, dass der zu erwartende Vorhersagefehler groß ist, wenn in der aktuellen Situationen (z.B. Beschleunigen) auch eine größere Varianz des Signals zu erwarten ist, jedoch sind diese Werte nicht gleich. Daher hat es sich herausgestellt, dass eine Vorhersage des zu erwartenden Fehlers hier besser funktioniert und zu dem konzeptuell auch sinnvoller ist. Zu diesem Zweck wurde für die letzten Endes in AP5300 durchgeführten Tests und die Demo Szenarien anstelle der Vorhersage der Standardabweichung eine Vorhersage für den zu erwartenden Fehler verwendet. Die Vorgehensweise bleibt im Grunde sehr ähnlich, es wird lediglich in dem Fenster in dem vorher

die Standardabweichung berechnet wurde nun die maximale Abweichung als Zielwert für die Vorhersage der Fehlertoleranz verwendet. Zu diesem Zweck ist allerdings ein weiterer Datensatz erforderlich, also zunächst wir die Sensordatenvorhersage auf Trainingsdaten trainiert, um nun aber realistische Trainingsdaten für die Fehlervorhersage zu erhalten, muss die Sensordaten auf bisher unbekannte Daten angewendet werden, also auf einem zweiten Datensatz, dieser wird dann für das Training der Fehlervorhersage verwendet.

Da das MLP ein statistisches Lernverfahren ist, wird mit der neuen Vorgehensweise der im Mittel zu erwartende Fehler in jeder Situation (mit einer zeitlichen Toleranz entsprechend der Fenstergröße auf der der maximale Fehler ermittelt wurde) gelernt. Daher können aber reale Fehlerwerte um diesen Wert entsprechend nach oben und unten mit gleicher Wahrscheinlichkeit abweichen, damit dieser Wert sich als Entscheidungskriterium für den Fehlertrigger eignen muss dieser mittels Multiplikator wie bereits im vorigen Fall beschrieben angepasst werden. Dieser Multiplikator ist im wesentlichen eine Einstellung, mit welcher Wahrscheinlichkeit ein Fehlerwert auftreten darf, um noch zugelassen zu werden.

Die Ergebnisse der finalen Auswertung finden sich im Abschnitt zu Arbeitspaket AP5300. Zu weiteren Ergebnissen von AP 4400 siehe das Kapitel A.2 im Anhang.

2.1.6 AP 5300: Eval. Vorhersagesystem

Vergleich verschiedener Vorhersagesysteme

In der Beschreibung zu AP4400 wurden drei Vorhersageverfahren (MLP, Pt3 und NG) bereits ausführlich verglichen. Hier wurde bereits in experimenteller Auswertung die Möglichkeit zur Sensordatenvorhersage mittels verschiedener Methoden ermittelt und dargestellt. Im Abschnitt "Experimentelle Ergebnisse" im Kapitel zu "AP4400" wurden die Ergebnisse bereits ausführlich diskutiert, dieser Abschnitt ist entsprechend auch Bestandteil der Auswertung und somit des AP5300. Die gewonnenen Erkenntnisse wurde bereits in einer Veröffentlichung verwertet (siehe hierzu [55]).

Für das finale Vorhersagesystem wurde dann wie in AP4400 beschreiben die Sensordatenvorhersage mittels eines MLPs realisiert.

Für die Verwendung der Vorhersage zur Verhaltenssteuerung ist es erforderlich, neben der Vorhersage für die zu erwartenden Sensordaten auch eine Vorhersage über den zu erwartenden Fehler zu erzeugen. Wie in den vorigen AP Beschreibungen erwähnt, wird die Vorhersage der Sensorwerte dazu verwendet, die Abweichung der Sensordaten von den erwarteten (den vorhergesagten) zu analysieren. Je nach Situation, Untergrund und Umgebungsbedingungen kann die Vorhersage aber mehr oder weniger gut zu den gemessenen Sensordaten passen. Erwartungsgemäß fiel hier auf, dass die Abweichungen bei positiven oder negativen Beschleunigungen stark von denen bei konstanter Geschwindigkeit abweichen. Dies lässt sich damit erklären, dass die tatsächliche Reaktion des Roboters auf Bewegungskommandos wie z.B. zur Änderung der Drehgeschwindigkeit immer leicht variieren. Da aber bereits eine geringe zeitliche Abweichung der Vorhersage hier zu recht deutlich abweichenden Werten führt (siehe z.B. Abbildung 2.33), ist auch der Fehler der in diesen Situationen zu erwarten ist größer als während einer konstanten Bewegung, bei der die Sensordaten im Normalfall wesentlich besser zu der Vorhersage passen.

Aus diesem Grund wurde für das finale System zur Verhaltenssteuerung eine zweite Vorhersage trainiert, die zusätzlich zu den Sensorwerten den aktuell zu erwartenden Fehler schätzt, der zu der aktuellen Situation (z.B. Beschleunigung) passt. Dementsprechend wird in diesem Abschnitt die Vorhersage verschiedener Sonsorwerte sowie die Vorhersage der Fehler, die zur Bestimmung der Fehlertoleranz im Verhaltensframework verwendet wird, erläutert. Um hier eine stabile Fehlertoleranz zu erhalten ist jedoch das Lernen auf den Fehlern an sich noch nicht ausreichend, da hier immer noch zeitliche Abweichungen einen großen Einfluss haben, um diesen ausreichend abzuschwächen wurde immer der Maximalwert der Fehlerkurve über 15 Samples vor und nach dem aktuellen Wert verwendet. Somit ergibt sich eine Vorhersage des im Mittel zum aktuellen Zeitpunkt zu erwartenden maximalen Fehlerwertes. Da die vorhergesagten Fehlerwerte aber das Ergebnis eines statistischen Lernverfahrens sind, entsprechen diese dem im Mittel an dieser Stelle maximal zu erwartenden Fehler. Da dies aber bei Annahme einer normalverteilten Schwankung (Rauschen) zu einer Ablehnung von 50% der Fälle führen würde, muss dieser Wert noch einmal mit einem Multiplikator multipliziert werden, der die eigentliche Fehlertoleranz einstellt. In den Versuchen hat sich hierfür ein Multiplikator von 1.5 als gutes Maß herausgestellt.

Vorhersage der IMU Gyroskop-Werte Zunächst werden nun die Vorhersageergebnisse für die Vorhersage der IMU-Gyro Werte erläutert. Diese werden bei der Verhaltenssteuerung dazu verwendet, Störungen im Ablauf der Bewegung des Roboters zu erkennen und darauf zu reagieren. Zum einen wurde die Y-Rotation dazu verwendet unerwartete Kippbewegungen (Rampenszenario) auf eigentlich geradem Untergrund zu erkennen. Zum anderen wurde diese Vorhersage verwendet um die Rotation um die Z-Achse auf Basis von Bewegungskommandos zu evaluieren. Hierbei sind besonders diejenigen Bewegungskommandos von Interesse, die eine Drehung beinhalten, da diese einen direkten Einfluss auf die gemessene Drehung der Gyroskope der IMU haben sollte.

Zur Evaluierung des Vorhersagesystems wurden nun 3 Datensätze auf dem Roboter aufgenommen. Hierbei hat sich der Roboter mit drei verschiedenen Geschwindigkeiten (0.4, 0.8, 1.4 rad/s) jeweils für einige Sekunden links bzw. rechts herum gedreht. Der erste Datensatz enthält jeweils 10 Wiederholungen der jeweiligen Bewegung, der zweite jeweils 8 und der dritte Datensatz jeweils 2 Wiederholungen. Der erste Datensatz wurde nun dazu verwendet, die Vorhersage der IMU Gyroskobe um die x-,y- und z-Achse zu lernen. Die gelernte Vorhersage wurde zur Vorhersage der z-Rotationswerte auf dem zweiten Testdatensatz mit je 8 Wiederholungen verwendet. Anschließend wurde der Fehler der gemachten Vorhersage auf diesem Datensatz dazu verwendet um die Vorhersage für den erwarteten Fehler zu trainieren. Der dritte Datensatz dient dann zur Auswertung, bei dem sowohl die Sensordatenvorhersage als auch die Fehlervorhersage auf nicht zum Training verwendeten Daten getestet und bewertet wird.

Die Bewertung erfolgt hier auf zwei Arten, wie bereits im vorigen Abschnitt. Auch wird hier der RMSE-Fehler als durchschnittliche Fehlermetrik für die Vorhersage auf den Testdaten verwendet. Ein weiteres Kriterium zur Auswertung ist hier, ob sich die Vorhersage für die Verwendung im Verhaltensframework verwenden lässt. Zu diesem Zweck wird die verwendete Triggerlogik des Verhaltensframeworks (für den dynamischen Threshold) hier dargestellt um zu zeigen, ob sich das Vorhersagesystem im Ganzen zur Verhaltenssteuerung eignet.

Abbildung 2.33(a) zeigt die Sensordatenvorhersage der IMU-Gyro Z Werte (blaue Linie), die Vorhersage der Fehlertoleranz wird hier als hervorgehobener Bereich um den Wert der Sensordatenvorhersage dargestellt. Der Plot umfasst den gesamten Testdatensatz (je 2 Wiederholungen mit 3 verschiedenen Geschwindigkeiten). Die tatsächlich gemessenen Sensordaten sind hier durch die rote Kurve dargestellt. In Abbildung 2.33(c) wird ein Ausschnitt dieser Grafik vergrößert dargestellt, um exemplarisch die genauen Kurvenverläufe deutlicher darzustellen.

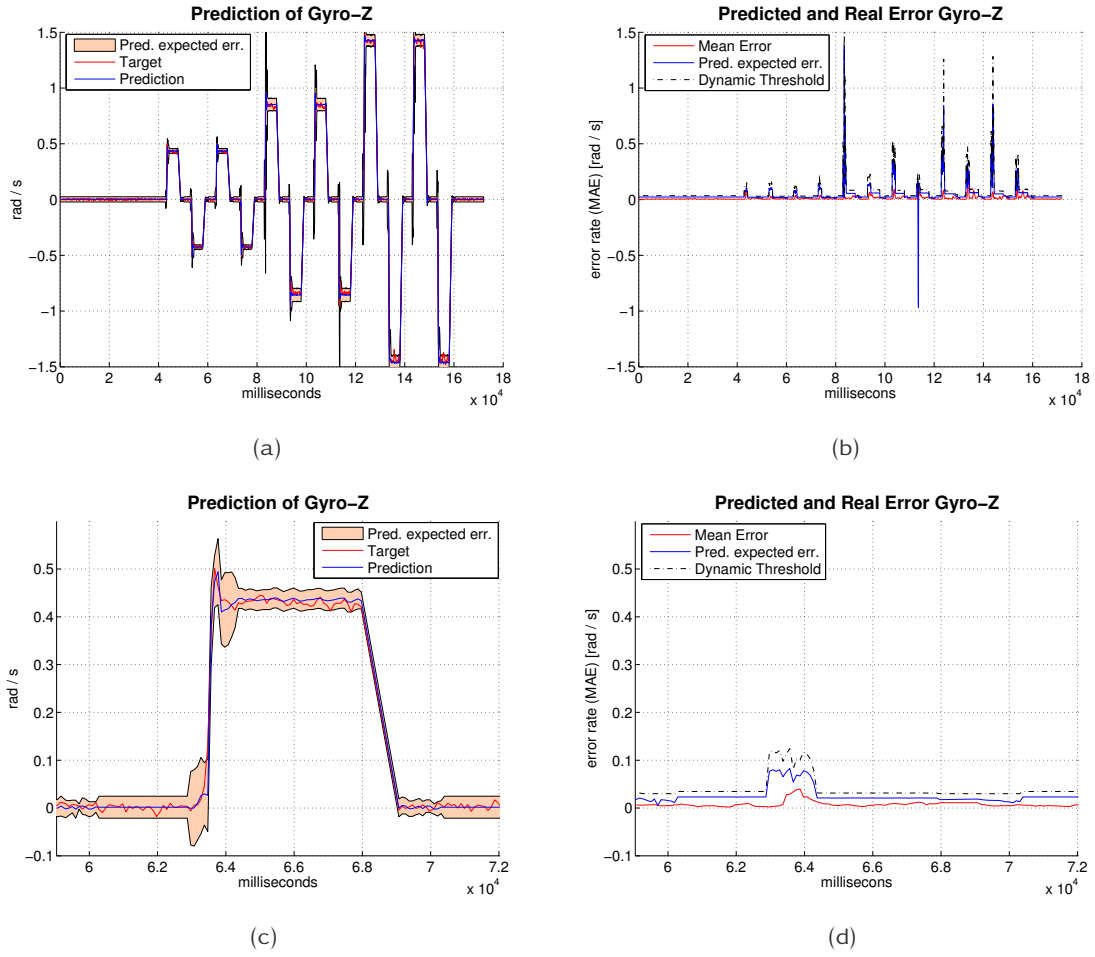


Abbildung 2.33 Diese Abbildung zeigt die Vorhersage für die von der IMU gemessenen Gyroskopwerte für die Drehung um die Z-Achse. Während Teil (a) die Werte und die Vorhersage zeigen, zeigt Teil (b) den Vergleich zwischen dem mittleren Vorhersagefehler der letzten 5 Werte und dem vorhergesagten zu erwartendem Fehler. Teil (c) und (d) zeigen entsprechend vergrößerte Ausschnitte.

In Abbildung 2.33(b) wird die zu Abbildung 2.33(a) gehörige Fehlerwertvorhersage dargestellt. Die rote Linie stellt hier den tatsächlichen Fehler über die letzten 5 Werte gemittelt dar (zur Unterdrückung von Ausreißern). Die blaue Kurve zeigt den vorhergesagten Fehlerwert und die schwarz gestrichelte Linie zeigt den dynamischen Threshold, nach der Triggerlogik berechnet. Hierbei ist zu sagen, dass die Fehlerkurve (rot) immer unterhalb der gestrichelten schwarzen Linie bleiben muss, um keine falsch positiven Auslösungen eines Fehlertriggers zu erzeugen, dies war für den hier dargestellten Testdatensatz durchgehend der Fall. In Abbildung 2.33(d) ist der vergrößerte Ausschnitt korrespondierend zu Abbildung 2.33(b) dargestellt.

Ergebnis: Der RMSE für die Sensordatenvorhersage beträgt für den Testdatensatz *0.025* und für die Fehlervorhersage *0.087* (beide in Radiant pro Sekunde). In Bezug auf die Verwendbarkeit im Vorhersagesystem ist auf Basis der Testdaten anzunehmen, dass in der Situation in der die Testdaten aufgenommen wurden (flacher Boden) nur selten oder gar nicht mit falsch positivem Triggern zu rechnen ist, da dies in den Testdaten nicht aufgetreten ist.

Vorhersage von Objektbewegung Es folgt ein Vergleich der Werte, für die Vorhersage der Bewegung eines Objektes im Kamerabild, basierend auf der eigenen Bewegung des Roboters. Vorhergesagt wird die zu erwartende Geschwindigkeit des Objektes im Kamerabild (Pixel pro Millisekunde). Abbildung 2.34 zeigt die Ergebnisse, die Plots sind hier, wie schon für die IMU Werte erklärt, zu interpretieren.

Ergebnis: Die Vorhersage der Änderung der x Koordinaten delta x (dx), also der Geschwindigkeit, hat auf den Testdaten einen RMSE Fehler von *0.020* und die Fehlervorhersage von *0.027*, beide in Pixel pro Millisekunde. Im Testdatensatz ist lediglich während einer Drehung (etwa bei Sample 400) eine fehlerhafte Positiverkennung aufgetreten, d.h. Überschreitung des mittleren Fehlers über den dynamischen Threshold.

Experimentelle Auswertung der Verhaltenssteuerung

Neben den in den vorigen Abschnitten erläuterten systematischen Analysen der Fehler bei den einzelnen Vorhersagen wurde zusätzlich auch das Verhalten des Roboters mit den auf den Vorhersagen basierenden Triggern getestet. Hierzu wurden einige der im AP5400[7] definierten Demoszenarios verwendet.

[7] Die Beschreibung zu diesem AP findet sich im Abschlussbericht des Projektpartners DFKI

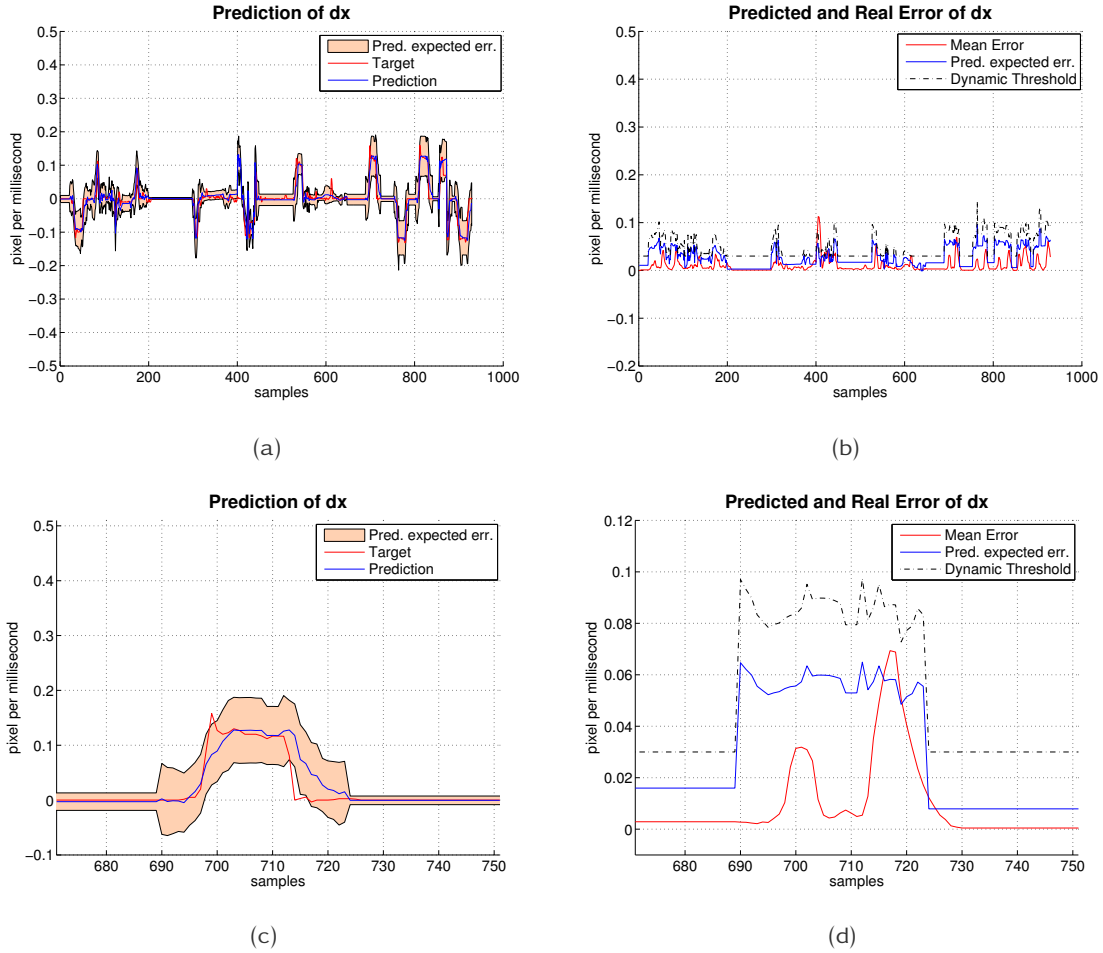


Abbildung 2.34 Diese Abbildung zeigt die Vorhersage für die gemessene Bewegung in X-Richtung von dem Objekt, delta x (dx). Während Teil (a) die Werte und die Vorhersage zeigen, zeigt Teil (b) den Vergleich zwischen dem mittleren Vorhersagefehler der letzten 5 Werte und dem vorhergesagten zu erwartendem Fehler. Teil (c) und (d) zeigen entsprechend vergrößerte Ausschnitte.

Hierzu zählt zum Beispiel die Vorhersage der Objektbewegung die in Abschnitt 2.1.6 erläutert wurde. Diese wurde verwendet um bei zwei Experimenten / Demos zu ermitteln ob sich ein im Bild der Kameras detektiertes Objekt bewegt oder nicht. Um die Vorhersage im praktischen Einsatz zu testen, wurde das "homing" Verhalten verwendet, bei dem der Roboter sich auf das erkannte Objekt ausrichtet und dann solange darauf zu fährt, bis das Objekt eine vorgegebene Größe (und somit korrespondierend einen bestimmten Abstand zum Objekt) erreicht hat. Hierbei wurde ein Trigger verwendet, der den dynamischen Fehler für *dx* verwendet um zu entscheiden, ob sich das Objekt wie vorhergesagt im Bild verhält (keine Eigenbewegung des Objektes) oder nicht. Als Reaktion stoppt der Trigger die aktuelle Ausführung. Das "homing" Experiment wurde nun 5 mal wiederholt und der Roboter konnte jedes mal bis zu dem Objekt fahren, es gab also in diesem Experiment keine falsch Positive Erkennung einer Objektbewegung.

Die Objektbewegungsvorhersage wurde ebenfalls für ein an das im vorigen Absatz beschriebenen angelehntes Experiment verwendet. Hierbei sollte der Roboter sich auf einem vorgegebenen Pfad bewegen und das Objekt wurde in diesem Fall auf einem Pioneer transportiert und kreuzt den Pfad des SeekurJr Roboters. Hierbei wurde ein Trigger verwendet, der den Roboter stoppt, sobald die Objektbewegung erkannt wird und die Bewegungsrichtung zur Bildmitte orientiert ist (d.h. der Pioneer mit dem Objekt die Bewegungsrichtung des SeekurJr kreuzt). In diesem Fall wurde der SeekurJr gestoppt und wartet bis sich das Objekt wieder entfernt hat. In dieser Trigger Konfiguration wurde nun 5 mal an einem stehenden Objekt vorbei gefahren, in diesen Fällen gab es *keine falsch positiven* Erkennungen und der Roboter passierte das Objekt jedes mal. Anschließend wurde der Roboter 5 mal mit dem sich zu seiner Bewegungsrichtung kreuzend bewegenden Objekt konfrontiert. In diesem Fall hat der SeekurJr, die Bewegung *in jedem Versuch* erkannt und gestoppt bis sich das Objekt aus dem Erfassungsbereich heraus bewegt hat. Zuletzt wurden noch ein paar Wiederholungen durchgeführt, bei denen das Objekt mehrfach während einer durchgehenden Bewegung des SeekurJr, dessen Pfad kreuzt. Bei diesem letzten Versuch gab es lediglich *eine falsch negative* Erkennung, bei der die Objektbewegung nicht korrekt erkannt wurde. Im letzten Fall lag dies allerdings daran, dass das Objekt bereits zu nahe am SeekurJr vorbei geführt wurde und es dadurch einen zu kleinen Bereich im Kamerabild abdeckte, was eine stabile Erkennung verhinderte.

Ein weiteres Experiment war das "Rampenszenario", bei dem der Roboter SeekurJr eine Rampe herunterfahren sollte. Zunächst sollte der Roboter kein Kippen um die IMU Y Achse erwarten, hierzu wurde ein Trigger verwendet der das "normale" Verhalten auf flachem Untergrund annimmt und daher auslöst, sobald eine größere Abweichung der Y Rotation zu der Vorhergesagten eintritt. Dieser Trigger stoppt nun den Roboter und löst für eine kurze Zeit

ein rückwärts Fahrverhalten aus. Im nächsten Schritt soll der Roboter das vor Ihm liegende Gebiet analysieren und ermitteln ob es passierbar ist oder nicht. Im Fall der Rampe soll der Roboter das Gelände als passierbar einstufen und nun mit einer erhöhten Erwartung an die Y-Rotation der IMU die Rampe erneut befahren und dieses mal passieren. Dieses Experiment soll das Zusammenspiel des LoB's und der Vorhersage demonstrieren. Der wichtige Aspekt hierbei ist die Adaption der Erwartung (Trigger). Der Roboter wurde nun 10 mal die Rampe herunter geschickt. In *6 dieser Versuche* verhielt sich der Roboter wie erwartet. In *3 der Fälle* hat der Roboter beim zweiten befahren der Rampe ebenfalls getriggert und konnte die Rampe somit nicht erfolgreich passieren. In *einem Fall* unternahm der Roboter keinen zweiten Versuch die Rampe zu befahren. Die Fehler bei denen der Trigger auch beim zweiten mal ausgelöst hat liegen in der Pfadplanung begründet, das Modul (Corridor Sorvoing) welches zur Planung der Bewegung verwendet wurde hat hier eine Trajektorie generiert, bei der der Roboter seitlich auf die Rampe gefahren ist und somit ein ganz anderes Kippverhalten erzeugt hat als bei dem erwarteten geraden Befahren der Rampe. Bei der Bewegungsplanung im Corridor Sorvoing wird ebenfalls ermittelt, ob ein Gelände passierbar ist oder nicht. In dem Versuch bei dem der Roboter die Rampe nicht erneut passiert hat, wurde der Weg vor dem Roboter fälschlicherweise für nicht passierbar gehalten.

Ein letztes Experiment zur Quantitativen Auswertung wurde aufbauend auf dem vorigen realisiert. Dieses wurde ebenfalls in der Projektdemo gezeigt. Hierbei wurde der Roboter erneut die Rampe hinunter geschickt, dieses mal jedoch wurde nach dem der Roboter aufgrund der unerwarteten Kippbewegung stoppt und zurück fährt eine spiegelnde Folie vor dem Roboter ausgebreitet. Diese Folie reflektiert nun die Laserstrahlen des Entfernungsmessers der für die Erstellung der internen Umgebungskarte des Roboters verwendet wird. Daher "denkt" der Roboter nun, dass sich vor Ihm ein "Loch" im Boden befindet und das Gelände somit nicht passierbar ist. In diesem Fall sollte er keinen weiteren Versuch unternehmen das Gelände an der Stelle zu passieren. Hierbei soll ein Trigger den Status im Corridor Sorvoing erkennen, dass hier mehrfach versucht wird eine Trajektorie durch das nicht passierbare Gebiet zu planen und die Planausführung daraufhin an dieser Stelle zu beenden. Dieses Experiment wurde 5 mal ausgeführt und hierbei hat der Roboter *jedes Mal* korrekt reagiert und die Planausführung abgebrochen.

2.2 Wichtigste Positionen des zahlenmäßigen Nachweises

Die von der AG Robotik für das Forschungsvorhaben VirGo[4] aufgewendeten Kosten fielen hauptsächlich in dem Bereich Personalkosten an. Der wesentliche Teil des Personalaufwandes wurde dabei für die Arbeitspakete 4100 bis 4400 benötigt. In diesen APs erfolgte die Spezifikation und Realisierung der zentralen Bestandteile "Levels-of-Behaviour"-Modell und Vorhersage und Selbstbewertung.

Bei den Beschaffungen nahm das Robotersystem "SeekurJr" von Adept/Mobile-Robots den größten Anteil ein. Wie zu Vorhabensbeginn geplant, war hier die Beschaffung einer robusten, Outdoor-fähigen Forschungsplattform notwendig.

Schließlich fielen Kosten zur Durchführung von Reisen zu wichtigen nationaln und internationalen wissenschaftlichen Konferenzen an. Dort erfolgten die Veröffentlichungen von Forschungsergebnissen und in Diskussionen konnten Expertenmeinungen in Form von Rückmeldungen und Hinweisen, beispielsweise zum Konzept der Vorhersage und der Selbstbewertung, gesammelt werden.

Insgesamt ist die Ausgabenplanung eingehalten worden. Eine detaillierte Aufstellung aller Kostenpositionen kann dem rechnerischen Verwendungsnachweis entnommen werden. Zu den Kosten beim Partner DFKI siehe dessen Bericht.

2.3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die Ziele und Aufgabenstellung des Vorhabens setzten sowohl einen großen Schritt in der Erbringung wissenschaftlicher Ergebnisse, wie auch die Durchführung sehr umfangreicher technischer Arbeiten voraus. Für die Bearbeitung der Aufgaben im Bereich wissenschaftlicher Grundlagenforschung waren Literaturarbeiten, die Entwicklung neuer Lösungsansätze, Durchführung von Experimenten, Datensammlungen und Analysen notwendig. Auf der technischen Seite standen insbesondere ausführliche Design- und Spezifikationsarbeiten, große Implementierungsaufgaben und lange Testläufe in Simulationen und auf dem realen Robotersystem. Um die geplanten Ziele erreichen zu können war daher ein entsprechend umfangreicher Personalaufwand notwendig.

Vergleicht man abschließend die aufgewendete Arbeit mit den wissenschaftlichen und technischen Ergebnissen des Forschungsvorhaben, so zeigt sich eine eindeutige Angemessenheit von Aufwand und Nutzen: Mit den Ergebnissen, die die AG Robotik nach dem Abschluss von VirGo[4] vorweisen und künftig nutzen kann, zeigen sich mehrere entscheidende Vorteile:

1. Die wissenschaftlichen Untersuchungen haben zu (publizierten) Ergebnissen geführt, die für das Thema eines autonomen, selbstevaluierenden robotischen Systems einen höchst interessanten Vorschlag ergeben und demonstriert haben.

2. Die Erkenntnisse über die untersuchten Lernverfahren lassen sich voraussichtlich auf andere Anwendungsfelder und Probleme übertragen und damit auch in künftigen Arbeiten der AG nutzen.

3. Die erfolgten Implementierungen (beispielsweise in pySPACE) sind leicht wiederverwendbar realisiert worden. Hier wird es bei kommenden Anwendungen deutliche Vereinfachungen geben.

Die Ergebnisse des Vorhabens sind somit sowohl in anderen, bereits laufenden Arbeiten als Werkzeuge und Hilfen bei anderen Themenschwerpunkten nutzbar, als auch im Rahmen von direkt auf den Ergebnissen aufbauenden und diese fortführenden Forschungsvorhaben.

2.4 Voraussichtlicher Nutzen, insbesondere der Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans

Mit den im Vorhaben erlangten Ergebnissen der Grundlagenforschungsaktivitäten konnte eine fundierte Basis bezüglich Sensorwertvorhersagen, Selbstevaluation sowie zur Realisierung von Fehlererkennungssystemen innerhalb von Systemen für weltraumrobotische Missionen, welche mit Sensoren ausgestattet sind, geschaffen werden. Mit dieser Basis hat die AG Robotik eine Kompetenz geschaffen, welche für künftige Forschungsaktivitäten von großem Wert ist.

Damit ergibt sich auch ein entscheidener Vorteil bei der Akquise neuer Forschungsvorhaben – sowohl für die AG Robotik alleine, als auch beim Aufbau von Kooperationen und neuen Verbundvorhaben. Hier wird (teilweise auch wieder gemeinsam mit dem DFKI Robotics Innovation

Center) an Einreichungen für nationale (DFG und andere) und europäische Ausschreibungen gearbeitet. Neben dem Anwendungsfeld der Raumfahrtrobotik ist hierbei auch die wissenschaftliche Fragestellung der Übertragbarkeit der in VirGo^4 erarbeiteten Verfahren auf andere Anwendungen und Probleme interessant.

Die AG Robotik kann als universitäre Arbeitsgruppe nicht selbst wirtschaftlich verwerten. Allerdings wird die AG Robotik bei der Akquise weiterer Projekte (auch gemeinsam mit dem DFKI Robotics Innovation Center) mit den in VirGo^4 gebildeten Kompetenzen bei Unternehmen Vorteile für die Initiierung neuer Foschungsvorhaben haben.

2.5 Während der Durchführung des Vorhabens bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Die Partner des Verbundvorhabens haben sich auf mehreren Konferenzen mit dem Thema und den Ergebnissen von VirGo^4 präsentiert und mit der jeweiligen Community ausgetauscht. Das Thema des mehrschichtigen Verhaltensmodells und die in VirGo^4 entwickelten Vorhersage- und Selbstevaluationsverfahren konnten erfolgreich präsentiert werden. Die Konzepte der Verhaltenssteuerung konnten ebenso bereits Einfluss auf die Modellierungen im Projekt BesMan nehmen.

Die Recherchen, die innerhalb des Projektes durchgeführt wurden, ergaben, dass das Thema Zeitreihenvorhersagen zur Realisierung von Fehlererkennungssystemen noch sehr wenig erforscht ist. Dies erklärt auch die gute Resonanz, die das Projektteam während des Austauschs mit anderen Institutionen erfahren hat.

Während der Durchführung des Vorhabens sind die Arbeiten von Armbrust et al. besonders aufgefallen [2, 3]. Es geht dabei um eine aus mehreren Komponenten bestehende Verhaltenssteuerung. Im Unterschied zum "Levels-of-Behaviour"-Modell sind dort jedoch nicht drei Schichten fest vorgegeben. Stattdessen wird auf einer hohen, deliberativen Schicht explizit die Kombination von verhaltensbasierter Steuerung und endlicher Zustandsautomaten untersucht. Eine explizite und tiefgreifende Kombination reaktiver und deliberativer Funktionen und Vorhersagen und Selbstbewertung finden nicht statt.

2.6 Veröffentlichungen

Im Rahmen des Forschungsvorhabens VirGo[4] wurden die folgenden Veröffentlichungen vorgenommen oder eingereicht (im Verbundvorhaben, also von beiden Partnern):

- Tim Köhler, Elmar Berghöfer, Christian Rauch, Frank Kirchner: Sensor Fault Detection and Compensation in Lunar/Planetary Robot Missions Using Time-Series Prediction Based on Machine Learning. In Acta Futura (2014), Issue 9: AI in Space Workshop at IJCAI 2013, pages 9-20, DOI 10.2420/ACT-BOK-AF, ESA Advanced Concepts Team, ESTEC, Noordwijk, The Netherlands, May/2014.

- *Eingereicht, noch nicht veröffentlicht:* Raúl Domínguez, Tim Köhler, Christian Rauch, Elmar Berghöfer, Frank Kirchner: Towards Autonomous Robot Long Distance Navigation by a Nature-Inspired Behavior Control Model Using Sensor Feedback Expectations. Special issue of the TPNC 2013 in Soft Computing, Springer.

- Elmar Berghöfer, Denis Schulze, Christian Rauch, Marko Tscherepanow, Tim Köhler, Sven Wachsmuth: ART-based fusion of multi-modal perception for robots. In Neurocomputing, Elsevier, volume o.A., pages 11-22, May/2013.

- Tim Köhler, Martin Schröer: Towards a "Holistic" Safety Monitoring in Intelligent Vehicle Control. In Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics, (ICINCO-2013), 29.7.-31.7.2013, Reykjavik, SciTePress Digital Library, volume 1, pages 583-588, Jul/2013. ISBN: 978-989-8565-70-9.

- Christian Rauch, Elmar Berghöfer, Tim Köhler, Frank Kirchner: Comparison of Sensor-Feedback Prediction Methods for Robust Behavior Execution. In KI 2013: From Research to Innovation and Practical Applications, (KI-13), 16.9.-20.9.2013, Koblenz, Springer, pages 200-211, Sep/2013. ISBN: 978-3-642-40941-7.

- Tim Köhler, Christian Rauch, Martin Schröer, Elmar Berghöfer, Frank Kirchner: Concept of a Biologically Inspired Robust Behaviour Control System. In Proceedings of International Conference on Intelligent Robotics and Applications 2012, (ICIRA-12), 03.10.-05.10.2012, Montreal, Quebec, Springer Berlin / Heidelberg, pages 486-495, Oct/2012. ISBN: 978-3-642-33514-3.

Im Rahmen von wissenschaftlichen Konferenzen wurden auch Ergebnisse des Vorhabens auf Postern oder in Vorträgen präsentiert und diskutiert, ohne dass jeweils zusätzlich ein Artikel dazu erstellt wurde:

- Raúl Domínguez, Tim Köhler, Christian Rauch, Elmar Berghöfer, Frank Kirchner: Towards Autonomous Robot Long Distance Navigation by a Nature-Inspired Behavior Control Model Using Sensor Feedback Expectations. Theory and Practice of Natural Computing (TPNC) 2013 *(Poster)*

- Sylvain Joyeux: Building Complex Systems with Single-Purpose Components. ICRA Workshop on Software Development in Robotics (SDIR), 2013 *(Vortrag)*

- Sylvain Joyeux and Thomas Roehr: ROS and Rock: mixing Orocos components and ROS nodes into a model-driven toolchain. ROSCon 2013 *(Vortrag)*

- Christian Rauch, Tim Köhler, Martin Schröer, Elmar Berghöfer, Frank Kirchner: A Concept of a Reliable Three-Layer Behaviour Control System for Cooperative Autonomous Robots. konferenz für Künstliche Intelligenz (KI-2012), 24.9.-27.9.2012, Saarbrücken, o.A., Sep/2012. *(Poster)*

Darüber hinaus erfolgten noch weitere Präsentationen von und Diskussionen zu Ergebnissen des Forschungsvorhabens im Rahmen von wissenschaftlichen Tagungen und Konferenzen, sowie auch von nicht-wissenschaftlichen Veranstaltungen. Siehe hierzu auch die Beschreibungen zum Arbeitspaket 1000 im Abschlussbericht des DFKI.

3 Zusammenfassung

Rückblickend auf das Verbundvorhaben "VirGo^4" läßt sich zusammenfassend feststellen, dass alle zuvor gestellten technischen und wissenschaftlichen Ziele erfüllt werden konnten. Im Folgenden werden die zwei in Kapitel 1.1.1 genannten Themenblöcke noch einmal aufgegriffen und beantwortet.

Verhaltenssteuerung

Es wurde eine biologisch motivierte Verhaltenssteuerung entwickelt, die es einem System ermöglicht, eine Selbsteinschätzung des momentanen Systemzustands vorzunehmen und darauf basierend eine angepasste Verhaltensauswahl, ggf. einen Verhaltenswechsel durchzuführen. Mit Hilfe der Verhaltensframeworkkomponenten konnte so das "Levels-of-Behaviour"-Modell realisiert werden. Details zu diesem Punkt finden sich in dem Text zu AP 4100/4200 bzw. im Abschlussbericht des Projektpartners DFKI (APs 3100, 3200: Framework-seitig). Auch sei hierzu auf die Beschreibung der Demonstrationen im Bericht des DFKI RIC verwiesen (AP 5400).

Vorhersagesystem

Das Vorhersagesystem und die Selbstevaluation wurden erfolgreich in den Arbeitspaketen 4300 und 4400 geplant bzw. realisiert (siehe die entsprechenden Ausführungen). Als zentrales Ergebnis können auch die erfolgreich durchgeführten Demonstrationen von Vorhersage und Selbstevaluation gesehen werden (AP 5300 in diesem Bericht und AP 5400 im Bericht des Vorhabenspartner DFKI RIC).

Literaturverzeichnis

[1] Arkin, Ronald C.: *Behavior-based Robotics*. 1998

[2] Armbrust, Christopher ; Kiekbusch, Lisa ; Ropertz, Thorsten ; Berns, Karsten: Verification of Behaviour Networks Using Finite-State Automata. In: Glimm, Birte (Hrsg.) ; Krüger, Antonio (Hrsg.): *KI 2012: Advances in Artificial Intelligence*. Saarbrücken, Germany : Springer, September 24-27 2012

[3] Armbrust, Christopher ; Schmidt, Daniel ; Berns, Karsten: Generating Behaviour Networks from Finite-State Machines. In: *Proceedings of the 7th German Conference on Robotics (ROBOTIK 2012)*, 2012

[4] Aycard, O ; Mari, JF: Learning to automatically detect features for mobile robots using second-order Hidden Markov Models. In: *Arxiv preprint cs/0501068* (2005). http://arxiv.org/abs/cs/0501068

[5] Bonato, Vanderlei ; Marques, Eduardo ; Constantinides, George a.: A Floating-point Extended Kalman Filter Implementation for Autonomous Mobile Robots. In: *Journal of Signal Processing Systems* 56 (2008), August, Nr. 1, 41–50. http://dx.doi.org/10.1007/s11265-008-0257-8. – ISSN 1939–8018

[6] Bristow, D.A. ; Tharayil, Marina ; Alleyne, A.G.: A Survey of Iterative Learning Control. In: *Control Systems Magazine, IEEE* 26 (2006), Nr. 3, 96–114. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1636313

[7] Britten, Kenneth H.: Mechanisms of self-motion perception. In: *Annual review of neuroscience* 31 (2008), Januar, 389–410. http://dx.doi.org/10.1146/annurev.neuro.29.051605.112953. – ISSN 0147–006X

[8] Browning, N. A. ; Grossberg, Stephen ; Mingolla, Ennio: Cortical dynamics of navigation and steering in natural scenes: Motion-based object segmentation, heading, and obstacle avoidance. In: *Neural networks : the official journal of the International Neural Network Society* 22 (2009), Dezember, Nr. 10, 1383–98. http://dx.doi.org/10.1016/j.neunet.2009.05.007. – ISSN 1879–2782

[9] Budiharto, Widodo ; Jazidie, Achmad ; Purwanto, Djoko: Indoor Navigation Using Adaptive Neuro Fuzzy Controller for Servant Robot. In: *2010 Second International Conference on Computer Engineering and Applications* (2010), 582–586. http://dx.doi.org/10.1109/ICCEA.2010.119. ISBN 978–1–4244–6079–3

[10] Busoniu, L. ; Babuska, R. ; De Schutter, B.: A Comprehensive Survey of Multiagent Reinforcement Learning. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38 (2008), März, Nr. 2, 156–172. http://dx.doi.org/10.1109/TSMCC.2007.913919. – ISSN 1094–6977

[11] Carreras, Marc ; Ridao, Pere ; Garcia, Rafael: Learning Reactive Robot Behaviors with a Neural-Q_Learning Approach. In: *Proceedings of* (2002). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.6367&rep=rep1&type=pdf

[12] Crespi, Valentino ; Galstyan, Aram ; Lerman, Kristina: Top-down vs bottom-up methodologies in multi-agent system design. In: *Autonomous Robots* 24 (2008), Januar, Nr. 3, 303–313. http://dx.doi.org/10.1007/s10514-007-9080-5. – ISSN 0929–5593

[13] Dempster, A.P. ; Laird, N.M. ; Rubin, D.B.: Maximum likelihood from incomplete data via EM algorithm. In: *J. Roy. Stat. Soc.* 39 (1977), S. 1–38

[14] Du, Xin ; Chen, Hua-hua ; Gu, Wei-kang: Neural network and genetic algorithm based global path planning in a static environment. In: *Journal of Zhejiang University SCIENCE* 6A (2005), Juni, Nr. 6, 549–554. http://dx.doi.org/10.1631/jzus.2005.A0549. – ISSN 1009–3095

[15] Durgin, Frank H. ; Gigone, Krista ; Scott, Rebecca: Perception of Visual Speed While Moving. In: *Journal of experimental psychology. Human perception and performance* 31 (2005), April, Nr. 2, 339–53. http://dx.doi.org/10.1037/0096-1523.31.2.339. – ISSN 0096–1523

[16] Egerstedt, Magnus ; Hu, Xiaoming: Formation Constrained Multi-Agent Control. In: *IEEE Transactions on Robotics and Automation* 17 (2001), Nr. 6, 947–951. http://dx.doi.org/10.1109/70.976029. – ISSN 1042296X

[17] Floreano, Dario ; Mattiussi, Claudio: *Bio-Inspired Artificial Intelligence*. Cambridge, Massachusetts : The MIT Press, 2008. – ISBN 978–0–262–06271–8

[18] Fox, M ; Ghallab, M ; Infantes, Guillaume ; Long, D: Robot Introspection through Learned Hidden Markov Models. In: *Artificial Intelligence* 170 (2006), Februar, Nr. 2, 59–113. http://dx.doi.org/10.1016/j.artint.2005.05.007. – ISSN 00043702

[19] Fritzke, Bernd: Incremental learning of local linear mappings. In: *ICANN* Bd. 95, Citeseer, 1995, 217–222

[20] Garforth, J ; Mchale, S ; Meehan, a: Executive attention, task selection and attention-based learning in a neurally controlled simulated robot. In: *Neurocomputing* 69 (2006), Oktober, Nr. 16-18, 1923–1945. http://dx.doi.org/10.1016/j.neucom.2005.11.018. – ISSN 09252312

[21] Gaudiano, P ; Zalama, E ; Coronado, J L.: An unsupervised neural network for low-level control of a wheeled mobile robot: noise resistance, stability, and hardware implementation. In: *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society* 26 (1996), Januar, Nr. 3, 485–96. http://dx.doi.org/10.1109/3477.499798. – ISSN 1083–4419

[22] Gouko, Manabu ; Ito, Koji: An Action Generation Model Using Time Series Prediction. In: *Neural Networks, 2007. IJCNN 2007.* (2007), 0–5. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4371025

[23] Gross, H.M. ; Stephan, V. ; Krabbes, M.: A neural field approach to topological reinforcement learning in continuous action spaces. In: *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on* Bd. 3, IEEE, 1998. – ISBN 0780348591, 1992–1997

[24] Gurney, Kevin ; Hussain, Amir ; Chambers, Jon ; Abdullah, Rudwan: Controlled and automatic processing in animals and machines with application to autonomous ve-

hicle control. In: *Artificial Neural Networks–ICANN 2009* (2009), 198–207. http://www.springerlink.com/index/mp84048h3216844x.pdf

[25] Ho ; Liu: *Simulated annealing based algorithm for smooth robot path planning with different kinematic constraints*. http://portal.acm.org/citation.cfm?id=1774361. Version: 2010

[26] Hoffmann, Heiko ; Pastor, Peter ; Park, Dae-Hyung ; Schaal, Stefan: Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. In: *2009 IEEE International Conference on Robotics and Automation* (2009), Mai, 2587–2592. http://dx.doi.org/10.1109/ROBOT.2009.5152423. ISBN 978–1–4244–2788–8

[27] Infantes, Guillaume ; Ingrand, F. ; Ghallab, M.: Learning Behaviors Models for Robot Execution Control. In: *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*, IOS Press, 2006, 678–682

[28] Iocchi, Luca ; Nardi, Daniele ; Piaggio, Maurizio ; Sgorbissa, Antonio: Distributed Coordination in Heterogeneous Multi-Robot Systems. In: *Autonomous Robots* 15 (2003), Nr. 2, 155–168. http://dx.doi.org/10.1023/A:1025589008533

[29] Jaradat, Mohammad Abdel K. ; Al-Rousan, Mohammad ; Quadan, Lara: Reinforcement based Mobile Robot Navigation in dynamic Environment. In: *Robotics and Computer-Integrated Manufacturing* 27 (2011), Februar, Nr. 1, 135–149. http://dx.doi.org/10.1016/j.rcim.2010.06.019. – ISSN 07365845

[30] Kaiser, Alexander ; Schenck, Wolfram ; Möller, Ralf: Mental Imagery in Artificial Agents. In: *Workshop New Challenges in Neural Computation 2010*, 2010, 25

[31] Kalman, R.E. ; Bucy, R.S.: New results in linear filtering and prediction theory. In: *Trans. ASME J. Basic Eng. D* 83 (1961), S. 95–107

[32] Kelley, Richard ; Tavakkoli, Alireza ; King, Christopher ; Nicolescu, Monica ; Nicolescu, Mircea ; Bebis, George: Understanding human intentions via hidden markov models in autonomous mobile robots. In: *Proceedings of the 3rd international conference on Human*

robot interaction - HRI '08 (2008), 367. http://dx.doi.org/10.1145/1349822.1349870. ISBN 9781605580173

[33] Kim, Juno ; Palmisano, Stephen: Visually mediated eye movements regulate the capture of optic flow in self-motion perception. In: *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale* 202 (2010), April, Nr. 2, 355–61. http://dx.doi.org/10.1007/s00221-009-2137-2. – ISSN 1432–1106

[34] Köhler, Tim ; Berghöfer, Elmar ; Rauch, Christian ; Kirchner, Frank: Sensor Fault Detection and Compensation in Lunar/Planetary Robot Missions Using Time-Series Prediction Based on Machine Learning. In: *Acta Futura* (2014), Nr. 9, S. 9–20. http://dx.doi.org/10.2420/ACT-BOK-AF. – ISSN 2309–1940. – DOI 10.2420/ACT-BOK-AF

[35] Kumar, Manish (Duke U. ; Garg, Devendra P. (Duke U.: Intelligent learning of fuzzy logic controllers via neural network and genetic algorithm. In: *Proceedings of 2004 JUSFA 2004 Japan – USA Symposium on Flexible Automation* (2004), S. 1–8

[36] Lebeltel, Olivier ; Bessière, P. ; Diard, Julien ; Mazer, Emmanuel: Bayesian robot programming. In: *Autonomous Robots* 16 (2004), Nr. 1, 49–79. http://www.springerlink.com/index/LH76585657N21244.pdf

[37] Levenberg, Kenneth: A Method for the Solution of Certain Problems in Least Squares. In: *Quarterly of Applied Mathematics* 2 (1944), S. 164–168

[38] Lin, Chuan-kai: A reinforcement learning adaptive fuzzy controller for robots. In: *Fuzzy Sets and Systems* 137 (2003), S. 339 – 352. ISBN 8867582968

[39] Liou, Jing-Jia ; Cheng, Kwang-Ting ; Kundu, Sandip ; Krstic, Angela: Fast Statistical Timing Analysis By Probabilistic Event Propagation. In: *Proc. 2001 Design Automation Conference* (2001), S. 661–666

[40] Lucidarme, Philippe ; Liégeois, A.: Learning Reactive Neurocontrollers using Simulated Annealing for Mobile Robots. In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* Bd. 1, IEEE, 2003, 674–679

[41] Marquardt, Donald: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. In: *Journal of the Society for Industrial & Applied Mathematics* 11 (1963), Nr. 2, S. 431–441

[42] Martinetz, T.M. ; Berkovich, S.G. ; Schulten, K.J.: "Neural-gas"network for vector quantization and its application to time-series prediction. In: *Neural Networks, IEEE Transactions on* 4 (1993), jul, Nr. 4, S. 558 –569. http://dx.doi.org/10.1109/72.238311. – ISSN 1045–9227

[43] Martinetz, T.M. ; Schulten, K.J.: A neural-gas network learns topologies. In: Kohonen, T. (Hrsg.) ; Mäkisara, K. (Hrsg.) ; Simula, O. (Hrsg.) ; Kangas, J. (Hrsg.): *Artificial Neural Networks*, North-Holland, Amsterdam, 1991, S. 397–402

[44] Michaud, Francois: Selecting behaviors using fuzzy logic. In: *Proceedings of 6th International Fuzzy Systems Conference* 1 (1997), Nr. July, 585–592. http://dx.doi.org/10.1109/FUZZY.1997.616432. ISBN 0–7803–3796–4

[45] Michaud, Francois: EMIB – Computational Architecture Based on Emotion and Motivation for Intentional Selection and Configuration of Behaviour-Producing Modules. In: *Cognitive Science Quarterly, Special Issue on Desires, Goals, Intentions, and Values: Computational Architectures* Vol. 3-4 (2002), 340–361. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.3327&rep=rep1&type=pdf

[46] Michaud, Francois ; Lachiver, Gerard ; Le Dinh, Chon T.: Architectural Methodology Based on Intentional Configuration of Behaviors. In: *Computational Intelligence* 17 (2001), Februar, Nr. 1, 132–156. http://dx.doi.org/10.1111/0824-7935.00136. – ISSN 0824–7935

[47] Mitchell, Tom M. (Carnegie Mellon U.: *Machine Learning*. Internatio. The McGraw-Hill Companies, Inc., 1997

[48] Moreno, David L. ; Regueiro, Carlos V. ; Iglesias, Roberto: Using Prior Knowledge to Improve Reinforcement Learning in Mobile Robotics. In: *Control* (1996)

[49] Morimoto, Jun ; Doya, Kenji: Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. In: *Robotics and Autonomous Systems* 36 (2001), 37–51. http://www.sciencedirect.com/science/article/pii/S0921889001001130

[50] Norman, Donald A. ; Shallice, Tim: Attention to action: Willed and automatic control of behaviour. In: Davidson, R J. (Hrsg.) ; Schwartz, G E. (Hrsg.) ; Shapiro, D (Hrsg.) ; Center

for Human Information Processing (Veranst.): *Consciousness and selfregulation* Bd. 4. Plenum Press, 1986 Plenum Press, 1986, Kapitel Vol. 4, S. 1–18

[51] Paris, Sébastien ; Pettré, Julien ; Donikian, Stéphane: Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach. In: *Computer Graphics Forum* 26 (2007), September, Nr. 3, 665–674. http://dx.doi.org/10.1111/j.1467-8659.2007.01090.x. – ISSN 0167–7055

[52] Parker, L. ; Chandra, Maureen ; Tang, Fang: Enabling autonomous sensor-sharing for tightly-coupled cooperative tasks. In: *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III* (2005), 119–130. http://www.springerlink.com/index/h325mh718w125123.pdf

[53] Peula, Jose M. ; Urdiales, Cristina ; Herrero, Ignacio ; Sánchez-Tato, Isabel ; Sandoval, Francisco: Pure reactive behavior learning using Case Based Reasoning for a vision based 4-legged robot. In: *Robotics and Autonomous Systems* 57 (2009), Juni, Nr. 6-7, 688–699. http://dx.doi.org/10.1016/j.robot.2008.11.003. – ISSN 09218890

[54] Ranasinghe, Nadeesha ; Shen, Wei-Min: Surprise-Based Learning for Developmental Robotics. In: *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)* (2008), August, 65–70. http://dx.doi.org/10.1109/LAB-RS.2008.18. ISBN 978–0–7695–3272–1

[55] Rauch, Christian ; Berghöfer, Elmar ; Köhler, Tim ; Kirchner, Frank: Comparison of Sensor-Feedback Prediction Methods for Robust Behavior Execution. In: Timm, IngoJ. (Hrsg.) ; Thimm, Matthias (Hrsg.): *KI 2013: Advances in Artificial Intelligence* Bd. 8077, Springer Berlin Heidelberg, 2013 (Lecture Notes in Computer Science). – ISBN 978–3–642–40941–7, S. 200–211

[56] Ren, Li-Hong ; Ding, Yong-Sheng ; Shen, Yi-Zhen ; Zhang, Xiang-Feng: Multi-agent-based bio-network for systems biology: protein-protein interaction network as an example. In: *Amino acids* 35 (2008), Oktober, Nr. 3, 565–72. http://dx.doi.org/10.1007/s00726-008-0081-2. – ISSN 1438–2199

[57] Reymond, Gilles ; Droulez, Jacques ; Kemeny, Andras: Visuovestibular perception of self-motion modeled as a dynamic optimization process. In: *Biological cybernetics* 87

(2002), Oktober, Nr. 4, 301–14. http://dx.doi.org/10.1007/s00422-002-0357-7. – ISSN 0340–1200

[58] Riedmiller, Martin ; Braun, Heinrich: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: *IEEE International Conference on Neural Networks*, IEEE Press, 1993, S. 586–591

[59] Ruan, Xiaogang ; Cai, Jianxian ; Chen, Jing: Learning to Control Two-Wheeled Self-Balancing Robot Using Reinforcement Learning Rules and Fuzzy Neural Networks. In: *2008 Fourth International Conference on Natural Computation* (2008), 395–398. http://dx.doi.org/10.1109/ICNC.2008.361. ISBN 978–0–7695–3304–9

[60] Rumelhart, David E. ; Hintont, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Nr. 6088, S. 533–536

[61] Rusu, P. ; Petriu, E.M. ; Whalen, T.E. ; Cornell, A. ; Spoelder, H.J.W.: Behavior-based neuro-fuzzy controller for mobile robot navigation. In: *IEEE Transactions on Instrumentation and Measurement* 52 (2003), August, Nr. 4, 1335–1340. http://dx.doi.org/10.1109/TIM.2003.816846. – ISSN 0018–9456

[62] Saegusa, Ryo ; Metta, Giorgio ; Sandini, Giulio ; Sakka, Sophie: Active motor babbling for sensorimotor learning. In: *2008 IEEE International Conference on Robotics and Biomimetics* (2009), Februar, 794–799. http://dx.doi.org/10.1109/ROBIO.2009.4913101. ISBN 978–1–4244–2678–2

[63] Saegusa, Ryo ; Nori, Francesco ; Sandini, Giulio ; Metta, Giorgio ; Sakka, Sophie: Sensory prediction for autonomous robots. In: *2007 7th IEEE-RAS International Conference on Humanoid Robots* (2007), November, 102–108. http://dx.doi.org/10.1109/ICHR.2007.4813855. ISBN 978–1–4244–1861–9

[64] Saffiotti, A.: The uses of fuzzy logic in autonomous robot navigation. In: *Soft Computing - A Fusion of Foundations, Methodologies and Applications* 1 (1997), Dezember, Nr. 4, 180–197. http://dx.doi.org/10.1007/s005000050020. – ISSN 1432–7643

[65] Schenck, Wolfram: Space Perception through Visuokinesthetic Prediction. In: *Anticipatory Behavior in Adaptive Learning Systems 2009* (2009), 247–266. http://www.springerlink.com/index/T7J5038687670175.pdf

[66] In: Schenck, Wolfram ; Möller, Ralf: *Space Perception by Visuokinesthetic Prediction*. Springer Verlag, 2008. – ISBN 978–3540404293, 247 – 266

[67] Schenck, Wolfram ; Sinder, Dennis ; Möller, Ralf: Combining neural networks and optimization techniques for visuokinesthetic prediction and motor planning. In: *ESANN*, ESANN, 2008, 23–25

[68] Schneider, Walter ; Shiffrin, Richard M.: Controlled and automatic human information processing: I. Detection, search, and attention. In: *Psychological Review* 84 (1977), S. 1–66

[69] Sebastian Thrun (Stanford University), Wolfram Burgard (University of Freiburg), Dieter Fox (University of W.: *Probabilistic Robotics*. 2005

[70] Sedighi, K.H. ; Ashenayi, K. ; Manikas, T.W. ; Wainwright, R.L.: Autonomous local path planning for a mobile robot using a genetic algorithm. In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)* (2004), 1338–1345. http://dx.doi.org/10.1109/CEC.2004.1331052. ISBN 0–7803–8515–2

[71] Shumway, R.H. ; Stoffer, D.S.: *Time Series Analysis and Its Application. With R Examples*. 2. New York : Springer, 2006

[72] Siciliano, Bruno (Hrsg.) ; Khatib, Oussama (Hrsg.): *Handbook of Robotics*. Springer, 2008 http://www.springerlink.com/content/978-3-540-23957-4. – ISBN 978–3–540–23957–4

[73] Simmons, R. ; Smith, T. ; Dias, M.B. ; Goldberg, D. ; Hershberger, D. ; Stentz, A. ; Zlot, R.: A Layered Architecture for Coordination of Mobile Robots. In: *Multi-Robot Systems: From Swarms to Intelligent Automata* (2002). http://dx.doi.org/10.1.1.57.7569

[74] Sridharan, Mohan ; Kuhlmann, Gregory ; Stone, Peter: Practical vision-based monte carlo localization on a legged robot. In: *ICRA 2005. Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005.*, IEEE, 2005. – ISBN 078038914X, 3366–3371

[75] Stroupe, Ashley W. ; Balch, Tucker: Value-based action selection for observation with robot teams using probabilistic techniques. In: *Robotics and Autonomous Systems* 50

(2005), Februar, Nr. 2-3, 85–97. http://dx.doi.org/10.1016/j.robot.2004.08.002. – ISSN 09218890

[76] Suh, Il H. ; Lee, Sanghoon ; Kwon, Woo Y. ; Cho, Young-Jo: Learning of Action Patterns and Reactive Behavior Plans via a Novel Two-Layered Ethology-Based Action Selection Mechanism. In: *Intelligent Robots and* (2005), 1232–1238. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1545148. ISBN 0780389123

[77] Thrun, Sebastian ; Fox, Dieter ; Burgard, Wolfram ; Dellaert, Frank: Robust Monte Carlo localization for mobile robots. In: *Artificial Intelligence* 128 (2001), Nr. 1-2, 99–141. http://www.sciencedirect.com/science/article/pii/S0004370201000698

[78] Wang, Y ; Desilva, C: A Machine-Learning Approach to Multi-Robot Coordination. In: *Engineering Applications of Artificial Intelligence* 21 (2008), April, Nr. 3, 470–484. http://dx.doi.org/10.1016/j.engappai.2007.05.006. – ISSN 09521976

[79] Werger, Barry B. ; Mataric, Maja J.: Broadcast of Local Eligibility: Behavior-Based Control for Strongly Cooperative Robot Teams. In: *Proceedings of the fourth international conference on Autonomous agents - AGENTS '00* (2000), 21–22. http://dx.doi.org/10.1145/336595.336621. ISBN 1581132301

[80] Werger, Barry B. ; Mataric, Maja J.: Broadcast of Local Eligibility for Multi-Target Observation. In: *Robotics* (2000). http://dx.doi.org/10.1.1.35.996

[81] Wolf, D.F. ; Sukhatme, G.S. ; Fox, D. ; Burgard, W.: Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (2005), Nr. April, 2026–2031. http://dx.doi.org/10.1109/ROBOT.2005.1570411. ISBN 0–7803–8914–X

[82] Xivry, Jean-Jacques O. ; Coppe, S. ; Lefèvre, P. ; Missal, Marcus: Biological motion drives perception and action. In: *Journal of Vision* 10 (2010), Nr. 2, 1–11. http://dx.doi.org/10.1167/10.2.6.Introduction

[83] Yun, SC ; Ganapathy, Veleppa: Improved genetic algorithms based optimum path planning for mobile robot. In: *Control Automation Robotics, Robotics and Vision* (2010), Nr. December, 7–10. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5707781. ISBN 9781424478156

Anhang A

A.1 Stand der Technik: Weitere Arbeiten

Neben den in Kapitel 2.1.1 vorgestellten Ergebnissen wurden noch folgende, für VirGo[4] relevante Arbeiten gefunden.

A.1.1 Verhalten

Nach Floreano und Mattiussi [17] werden Verhalten aus Sicht von Designern des Systems oft als funktionelle Ketten dargestellt, in denen der Agent etwas wahrnimmt, dann modelliert, plant und schließlich ausführt. Geht man davon aus, dass Agenten jedoch nicht bei jeder Wahrnehmung auch modellieren und planen müssen, um Aktionen auszuführen oder dass manche Wahrnehmungen gar nicht zu Aktionen führen, stimmt eine solche Darstellung von Verhalten nicht. Stattdessen wird vorgeschlagen, Verhalten als einfache Kette von Sensorinput, Verhaltensauswahl und Aktion wie in Abbildung A.1 zu sehen, darzustellen.

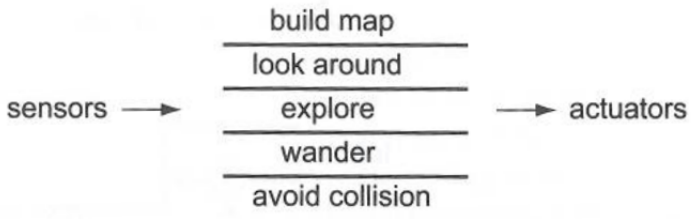


Abbildung A.1 Darstellung von Verhaltensauswahl nach Floreano [17].

Hier werden Kompetenzen des Agenten (wie z.B. build map und look around), die zur Ausführung einer Aktion notwendig sind, parallel modelliert und ausgeführt. Mit dieser Darstellung

wird das Ziel verfolgt, dass das Modell direkt auf Sensordaten reagiert und nicht von abstrakten Beschreibungen abhängig ist. Dabei sind die Verhalten im Sinne der Maslowschen Bedürfnispyramide angeordnet, d.h. von unten nach oben werden existenznotwendige Bedürfnisse, Sicherheit, soziale und Individualbedürfnisse modelliert. Die Verhalten der untersten Schicht werden automatisch angesprochen, treffen jedoch die Bedingungen für Verhalten höherer Schichten zu, können entweder die Verhalten beider Schichten parallel ausgeführt werden, oder aber Verhalten unterer Schichten wie in Abbildung A.2 zu sehen unterdrückt oder verändert werden, wobei obere Schichten nicht auf Verhalten unterer Schichten aufbauen.

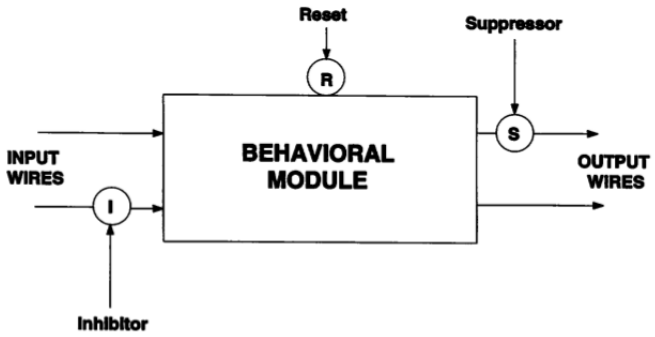


Abbildung A.2 Darstellung der Auswahl eines Verhaltens nach Arkin [1].

In Abbildung A.2 kann man erkennen, dass der Sensorinput entweder direkt zur Ausführung eines Verhaltens (grauer Kasten) führen kann, oder durch ein höheres Verhalten unterdrückt (*suppressed*) werden kann. Durch ein *Reset* kann ein höheres Layer das Verhalten verändern oder ausschalten und schaltet sich ein höheres Verhalten als ein *Inhibitor* dazu, wird die Ausführung der Aktion verhindert.

Eine Kombination/ Koordination von mehreren Verhalten nach der gerade vorgestellten Architektur könnte dann wie in Abbildung A.3 dargestellt werden.

Im untersten *Layer*, welches das Security Layer des Agenten darstellt, wird hier das *Run Away* Verhalten direkt angesprochen. Muss der Agent keinem Objekt ausweichen, wird das Verhalten jedoch nicht ausgeführt und stattdessen das *Forward* Verhalten angesprochen. Beide Verhalten der untersten Ebene sprechen direkt den Motor und die Bremsen an. In der zweiten Ebene, dem *Explore Layer*, liegen dann die erweiterten Verhalten *Wander* (zur Auskundschaftung der Umgebung) und *Go*. Wander ist abhängig von einer *Clock*, da es nur eine bestimmte Zeit lang ausgeführt wird. Außerdem kann wander die beiden Aktionen des untersten Layers beeinflussen (suppress). In der obersten Ebene steht eine Kollision direkt

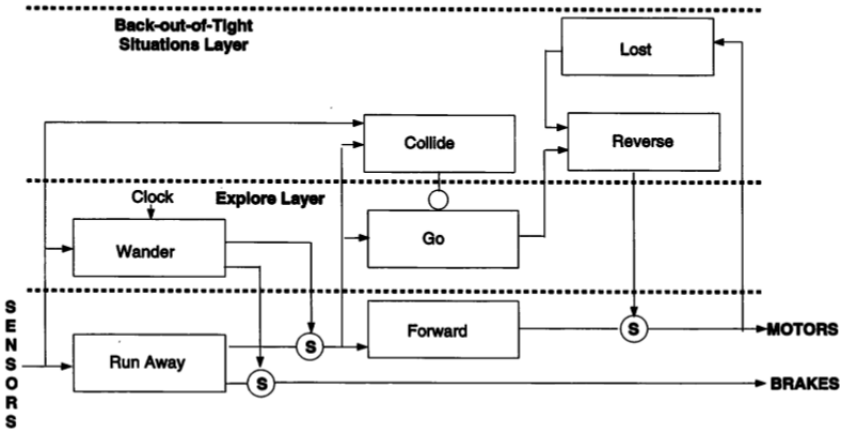


Abbildung A.3 Koordination von Verhalten nach Arkin [1].

bevor *Collide* oder der Agent hat sich verlaufen *Lost*. Im ersten Fall wird der Befehl zum Weiterlaufen unterdrückt (Verbindung zu *Go),* im zweiten Fall wird ein weiteres Verhalten angesprochen, *Reverse*. Floreano und Mattiussi [17] zeigen auf Seiten 413-415 ein weiteres Beispiel zur Anwendung dieser Architektur.

A.1.2 Lernen von Verhalten

Es gibt viele Ansätze des verhaltensbasierten Lernens, die teilweise Weiterentwicklungen von bestehenden Methoden der Statistik und des maschinellen Lernens sind, aber auch Kombinationen von Algorithmen darstellen können. Abbildung A.4 stellt eine Übersicht über einige für das Projekt interessante Verfahren dar.

Im Folgenden soll auf die in der Grafik erwähnten Verfahren näher eingegangen werden.

Naive Bayes Das statistische Verfahren Naive Bayes und andere Methoden, die auf dem Satz von Bayes beruhen, klassifizieren neue Instanzen durch Auswertung vorheriger Hypothesen/ Klassifizierungen und der gegebenen Datenbasis, wobei Unabhängigkeit der Variablen vorausgesetzt wird. Ein Unterschied zu anderen Methoden des maschinellen Lernens ist, dass die Datenbank nicht nach geeigneten Hypothesen durchsucht wird sondern eine Kalkulation

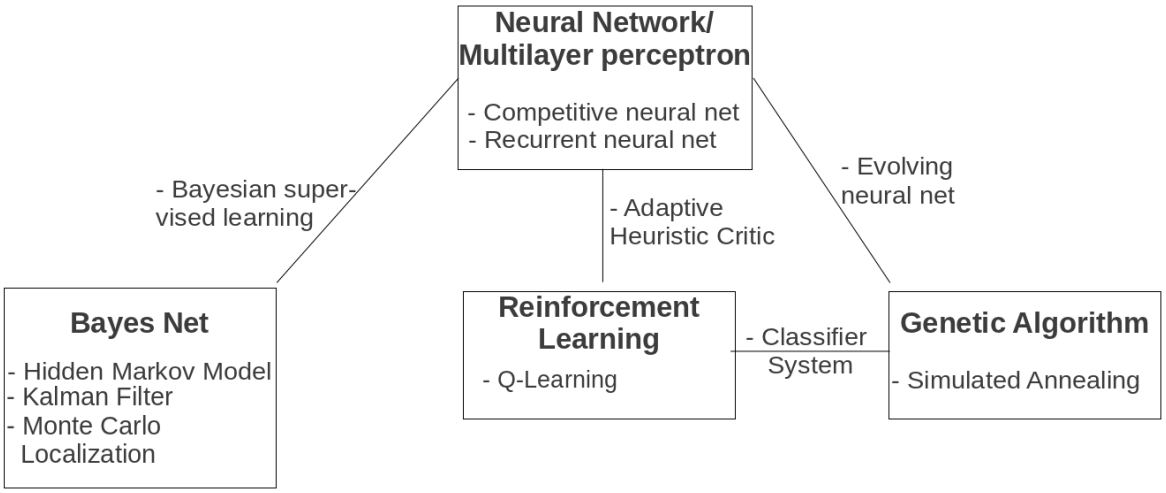


Abbildung A.4 Übersicht über Lernverfahren des maschinellen Lernens, die für das Projekt eingesetzt werden könnten.

auf Basis von Häufigkeiten verschiedener Datenkombinationen vorgenommen wird [47]. Auch gehen statistische Methoden nicht von einer einzigen, optimalen Lösung aus, sondern von vielen Lösungen mit unterschiedlichen Eintrittswahrscheinlichkeiten [69].

Ein Nachteil von Methoden, die auf dem Satz von Bayes beruhen, ist dass die Eintrittswahrscheinlichkeiten bekannt sein müssen oder geschätzt werden müssen.
Im Bayes'schen Netz wird das Bayes'sche Theorem in Form eines gerichteten Graphen aus Knoten und Kanten dargestellt. Die Knoten des Netzes beinhalten die Zufallsvariablen mit ihren bedingten Wahrscheinlichkeiten, die Kanten beschreiben die Abhängigkeiten der Variablen.
Ein Beispiel für die Anwendung des Bayes' Theorem wird von Lebeltel et al. [36] vorgestellt. Hier lernt ein Roboter mit Hilfe von Bayes'scher Inferenz reaktives Verhalten und die Kombination von Aktionen zur Durchführung eines Ziels, Situationen zu erkennen und Objekte zu klassifizieren. Außerdem wird eine *Sensor Fusion* von 8 Lichtquellen durchgeführt. Das genannte Verfahren ist in der Lage, unvollständige und unsichere Informationen zu verarbeiten.

Eine Anwendung von *Dynamic Bayesian Network* (DBN) zum inkrementellen Lernen von Verhalten findet sich in [27]. Durch Expectation Maximisation (EM) werden die Wahrscheinlichkeiten von Wechseln zwischen *belief states* ermittelt. Über die inkrementelle Anwendung von DBNs ist es möglich, bereits gelernte Modelle mit neuen Erkenntnissen zu erweitern und so in

dynamischen Umgebungen zu lernen. Zur Entscheidungsfindung in diesem Modell wird ein Vertrauensfaktor für jeden Wechsel von belief states eingeführt. In einer Lernphase wird dieser Faktor durch verkettet aufgerufene belief states aktualisiert, wobei häufige Übergänge von zwei Zuständen ein höheres Vertrauen erhalten, als Zustände, die nie oder kaum von anderen Zuständen aus aufgerufen werden. Bei der späteren Bevorzugung von Wechseln mit hohen Vertrauenswerten bei Entscheidungen kann dieses aufgebaute Modell genutzt werden. Werden risikoreiche Entscheidungen mit einem geringen Vertrauenswert bevorzugt, so werden Zustände erreicht, die während des Lernprozesses noch nicht durchlaufen wurden. In diesem Fall werden die Umgebung und die eigenen Möglichkeiten des Modells untersucht und erweitert.

Hidden Markov Model Das Hidden Markov Model baut auf einer Markov-Kette mit Zuständen und Übergangswahrscheinlichkeiten auf. Die Zustände dieser Markov-Kette werden zusammen mit den jeweiligen Beobachtungen der Zustände in Knoten eines Bayes'schen Netzes repräsentiert, denen dann eine Wahrscheinlichkeitverteilung zugefügt wird. Das Hidden Markov Model 1. Ordnung ist nur abhängig vom Zeitpunkt *t-1*. Will man eine Abhängigkeit der Variablen von mehr als nur dem vorangehenden Zeitpunkt darstellen, können in Hidden Markov Models n-ter Ordnung Verbindungen im Netz hinzugefügt werden.
Aycard et al. [4] benutzen Hidden Markov Models zur automatischen Erkennung von Objekten (*feature detection*) wie T-Kreuzungen und offenstehende Türen im Innenbereich aber auch kleine/große Steine auf beiden Seiten und Steigungen im Außenbereich, was z.B. für die autonome Navigation genutzt werden kann. Ein Vorteil des vorgestellten Verfahrens ist, dass es sehr gut mit *noise* umgehen kann.
In der Arbeit von Kelley et al.[32] wird versucht, Absichten von anderen Agenten auf Grundlage von Erfahrungen und durch das Hineinversetzen in die Lage des Anderen vorherzusagen, bevor die Aktion durchgeführt ist. Diese Technik basiert auf der Verarbeitung von visuellen Reizen und dem Gebrauch von Hidden Markov Models, welche sich als sehr effizient für die Vorhersage von Aktionen von Agenten erweisen. Auch der vorgestellte Gebrauch von abstrakten activity structures ist leicht verständlich, jedoch bergen Hidden Markov Models den Nachteil, dass man wegen des zugrundeliegenden stochastischen Modells den Weg der Lösungsfindung nicht nachvollziehen kann.
Eine weitere Anwendung von Hidden Markov Models zeigt sich in der Arbeit von Wolf et al.[81], die den Algorithmus zur Erstellung von 3D Landkarten mit Hilfe eines SICK Laserscanners benutzen. In A.5 kann man das Ergebnis erkennen, das eine erfolgreiche Klassifikation der Terrains in wegsames und unwegsames Gelände darstellt.

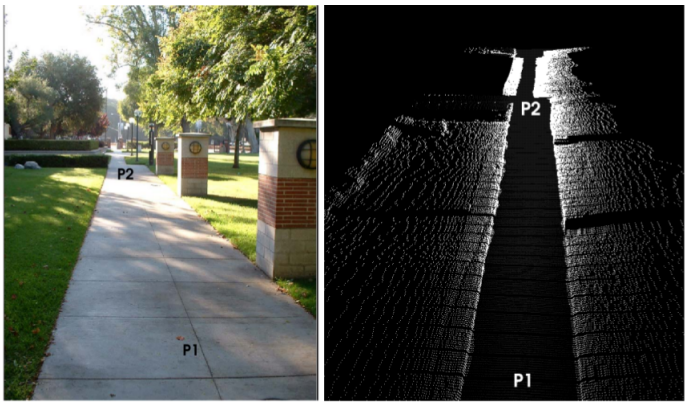


Abbildung A.5 In Wolf [81] erstellte Landkarte mit Hilfe eines SICK Laserscanners. P1 und P2 stellen dabei Referenzpunkte dar, die den gleichen Bildabschnitt kennzeichnen.

Durch die Überwachung der inneren Zustände eines auf HMM basierenden Modells in [18] lassen sich Verhaltensweisen von Systemen in Abhängigkeit der Sensordaten vorhersagen. Dazu wird automatisch ein nur auf Sensordaten basierendes HMM für ein zu lösendes Problem erlernt. Durch den Ansatz des Lernens basierend auf Sensordaten ist das Modell flexibel und für dynamische unsichere Umgebungen geeignet. Die Überwachung der inneren Zustände des HMM ermöglicht es, die durch Sensordaten hervorgerufenen abstrakten Verhalten mit den Sensordaten in Zusammenhang zu bringen. Durch das Erkennen dieser Zusammenhänge ist es dem Roboter möglich, während der Ausführung einer Aufgabe die durch Sensordaten hervorgerufene Verhalten vorherzusehen.

Kalman Filter Nach Bonato et al. [5] setzt sich der Kalman Filter aus zwei Teilen zusammen: der Vorhersage und der Aktualisierung. Bei der Vorhersage wird auf Grundlage des Satzes von Bayes z.B. die Position eines Agenten anhand seiner vorherigen Position und einer Bewegungsvariablen bestimmt. Die Aktualisierung bezieht sich auf neue Sensordaten, die die interne Karte des Agenten verändern und damit die Position des Agenten bestimmen können. Diese beiden Schritte werden zu jedem neuen Zeitpunkt erneut ausgeführt, wobei Linearität vorausgesetzt wird. Eine Methode, die Nichtlinearität der Übergänge einbeziehen kann ist der *Extended Kalman Filter (EKF)* [69, 5].
Kalman Filter sind effizient und einfach in der Anwendung [69], sie können jedoch nicht mehrere Hypothesen (z.B. mehrere Vorhersagen zur Lokalisation) verarbeiten. Auch werden zur Berechnung lineare Funktionen benutzt, die nicht immer gute Schätzer darstellen.
Der EKF wird überwiegend in der Lokalisierung verwendet, wobei er bei globalen Lokalisie-

rungsproblemen [1] nicht effizient arbeitet [69]. Zur Lokalisierung werden die Bildinformationen drastisch reduziert. Ein weiterer Nachteil des EKF ist, dass negative Sensordaten (wie z.B. ein fehlendes Objekt) nicht verarbeitet werden können.

Monte-Carlo Localization Bei der Monte-Carlo Lokalization wird das Wahrscheinlichkeitsmodell durch Partikelfilter (wobei ein Partikel ein Beispiel/ eine Hypothese ist) dargestellt, denen Wichtigkeiten zugeordnet werden. Dabei hängt die Genauigkeit des Modells von der Zahl der genutzten Partikel ab (je höher die Zahl der Partikel, desto höher die Genauigkeit) [69]. Im Gegensatz zum Kalman Filter kann die Monte-Carlo Localization Methode auch globale Lokalisierungsprobleme lösen und negative Sensordaten verarbeiten.
Mit Hilfe der Monte-Carlo Lokalisation wird versucht, die Position eines Agenten in Relation zu seiner Umgebung anzugeben [77]. Thrun et al. [77] zeigen, dass die Monte-Carlo Lokalization effizient auf globale Lokalisierungsprobleme und auch das *Kidnapped Robot Problem* angewandt werden kann. Sridharan et al. [74] stellen eine Methode zur Stabilisierung des Agenten vor in der der Agent an seiner Position bleibt auch wenn er mit einem Hindernis kollidiert, indem sie eine Historie der gesichteten *landmarks* abspeichern.
Grundsätzlich haben alle vorgestellten statistischen Verfahren Probleme mit dynamischen Umgebungen.

Neural Net Neuronale Netze sind in Software implementierte Computermodelle, die das Verhalten des biologischen Nervensystems nachbilden sollen. Dabei sind mehrere Einheiten des Systems, die Neuronen, miteinander verbunden. Einige Einheiten werden durch Umweltreize angesprochen (*input units*), manche Einheiten beeinflussen direkt ihre Umwelt (*output units*) und andere besitzen keine Verbindung zur Umwelt sondern nur zu anderen Einheiten (*hidden units*). Die Verbindungen sind dabei gewichtet, jeder Knoten des input layers ist mit den Knoten des hidden layers verbunden und jeder Knoten des hidden layers mit den Knoten des output layers. Ein gerichteter Graph, bei dem alle Einheiten einer Schicht nur die Aktivierungen der Einheiten der jeweils vorhergehenden Schicht verwenden, wird als *feedforward neural net* bezeichnet. Netze mit mehr als einer Schicht werden *multilayer perceptron* genannt [17].
Inkrementellen Vorhersagen von zukünftigen Sensordaten aus aktuellen Daten werden häufig mittels *Forward Models (FM)* realisiert, welche wiederum auf Neuronalen Netzen/*Multilayer Perceptrons (MLP)* basieren. [63] beschreibt ein Vorhersagemodell, welches eine auf MLP-basierte Lernmethode verwendet und eine Methode zur Generierung von Behavior nutzt.

[1] Bei der globalen Lokalisierung kennt der Agent seine Position im Raum nicht und weiß gleichzeitig nichts über seine Umgebung. Im Gegensatz dazu kennt der Agent bei der lokalen Lokalisierung seine Startposition.

Auch neuronale Netze sind robust, können mit noise gut umgehen und können im Gegensatz zu statistischen Verfahren auch Funktionen unbekannter Form modellieren. Außerdem können benötigte Knoten bei Bedarf (je nach angewandtem Verfahren) automatisch hinzugefügt werden. Ein Nachteil des neuronalen Netzes ist, dass (je nach eingesetztem Lernverfahren) die Gewichte des Netzes zum lokalen Minimum konvergieren können. Daneben besteht das Problem des *Overfitting*, bei dem das Netz zu spezielle Regeln aus den Trainingsdaten lernt und schlechter auf sich ändernde Bedingungen reagieren kann.
Gaudiano et al. [21] stellen ein neuronales Netz vor, das mit Unsicherheit in der Lokalisierung von Robotern umgehen kann. In Simulationen konnte der Agent erfolgreich Ziele erreichen auch wenn Sensordaten nicht mehr geliefert wurden (*blindes System*). Das Ziel wurde bei Ausrutschen ohne Unterbrechung weiter verfolgt und an Änderungen der Umgebung konnte sich der Agent durch ständiges Feedback an die Sensoren und On-Line-Umsetzung des Verfahrens anpassen.

Recurrent Neural Net Bei einem recurrent neural net sind beliebig gerichtete Verbindungen zwischen Knoten des gleichen layers oder zwischen unterschiedlichen Schichten möglich. Dadurch kann das Netz ein dynamisches Verhalten zeigen (auch bei statischen Eingaben) bzw. modellieren.

Competitive Neural Net Ein in [22] verwendetes "competitive neural net" genanntes Netz baut auf einem feedforward neural net auf, wobei das output layer jedoch als zweidimensionales grid dargestellt werden kann und die input units normalisiert sind. Diese neuronale Netz organisiert sich selbst: es werden nicht das gewünschte und aktuelle Ergebnis miteinander verglichen, sondern die Gewichte der input und output units, wonach die Gewichte der output unit angepasst werden [2].
Gouko et al. [22] stellen eine adaptive Variante des Lernens vor, in der Sensordaten zusammen mit einer Vorhersage auf Grundlage von vergangenen Situationen für die Entscheidungsfindung verarbeitet werden. Damit wird das Ziel verfolgt, sich an ändernde Umgebungen anpassen zu können, da das System bei einer wiederkehrenden Situation neue Entscheidungen fällen kann, sich aber auch an die Empfehlung auf Grundlage vergangener Situationen halten kann. Für die Generierung von Aktionen werden ein *competitive neural network* und ein *recurrent neural network* eingesetzt.

[2] siehe `http://www.benbest.com/computer/nn.html`

Bayesian Supervised Learning Eine Kombination von neuronalen Netzen mit dem Bayes' Theorem ist das bayesian supervised learning (oder auch *Maximize Likelihood in a neural net* [47] , in dem die Gewichte eines neuronalen Netzes als Parameter der Wahrscheinlichkeiten im Satz von Bayes genutzt werden. Durch den Gebrauch mehrerer Funktionen und Wahrscheinlichkeiten wird Unsicherheit in die Berechnung einbezogen. Die so erzeugten Ausdrücke sind jedoch schwer zu erstellen und sehr komplex [17].

Genetic Algorithm Genetische Algorithmen agieren auf einer Population von "Individuen", deren "DNA" durch ihre Parameterkonstellation in einem Bit-String dargestellt wird. Die Güte einzelner Individuen wird mittels einer *fitness function* gemessen. Anhand der fitness function wird entschieden, welche Individuen in die nächste Generation übernommen werden und welche Individuen sich paaren(*crossover*), um Nachkommen zu erzeugen. Zur Vermeidung lokaler Minima werden außerdem zufällig einige DNAs von Individuen verändert (*mutation*).

Simulated Annealing Beim Simulated Annealing (der simulierten Abkühlung) wird versucht, mittels Nachbildung eines Abkühlungsprozesses globale Minima einer Funktion zu finden. Ho und Liu [25] wenden Simulated Annealing zur Bestimmung des kürzesten Weges an.

In [40] werden reaktive Verhalten automatisch über ein Neuronales Netz anhand der Sensorimotor-Information generiert. Dabei wird Simulated Annealing zur Optimierung der Neuronengewichtungen eingesetzt.

Reinforcement Learning Reinforcement Learning ist ein Überbegriff für eine Reihe von Methoden des maschinellen Lernens, bei denen ein Agent den Nutzen von Aktionen bewertet. Die Architektur des Agenten beim Reinforcement Learning besteht aus einem *Actor* und einem *Critic*, die beide Sensorinformationen aus der Umwelt erhalten. Der Actor wählt Verhalten aus dem Pool vorhandener Behavior aus und der Critic verarbeitet zusätzlich Informationen zur Belohnung/ dem Nutzen (*reward*, eine numerische Bewertung der Güte einer Aktion) einer Aktion. Der Reward wird dann indirekt an den Actor zur eventuellen Anpassung der Auswahlkriterien weitergegeben, da eine Maximierung der Belohnung angestrebt wird. Dadurch entsteht eine Kette von Zuständen, Aktionen und Belohnungen [47]. Die Architektur des Reinforcement Learnings kann mit Hilfe verschiedener bereits genannter Verfahren durch Einführung eines Critic umgesetzt werden.

Adaptive Heuristic Critic Die Struktur des reinforcement learnings kann auf viele der bereits genannten Methoden angewandt werden. Beim adaptive heuristic critic learning werden der Prozess des Lernens einer Aktion und der Prozess des Lernens der Nutzenfunktion getrennt. Zum Beispiel können jeweils der Actor und der Critic durch ein neuronales Netz umgesetzt werden [17].

Lin et al. [38] benutzen Adaptive heuristic critic zur Kontrolle/Verhinderung von Ausrutschen, wobei der Agent keine vorherigen Informationen zum Verhalten bei Ausrutschen mitbekommt.

Q-Learning Q-Learning beschreibt eine Nutzenfunktion für die Bewertung von Aktionen und Zuständen im Reinforcement Learning, actor und critic werden also nicht voneinander getrennt.
Morimoto et al. [49] kombinieren Q-Learning mit einer *continuous actor-critic method* in ihrem hierarchical reinforcement learning, in der es zwei getrennte Ebenen gibt. Die obere Ebene lernt eine Abfolge von Unterzielen/ Aufgaben zur Erzielung des Hauptziels und ist in Q-Learning implementiert, die untere Ebene lernen, wie die Unterziele erreicht werden können. Der Agent in ihrer Simulation lernt selbstständig aufzustehen.

In [11] werden die Q-Funktionen durch mehrschichtige Neuronale Netze abgebildet, wobei jede dieser Funktionen ein reaktives und adaptives Verhalten repräsentiert. Die Auswahl der Verhalten und das Zusammenführen der Verhaltensaktionen wird von einem *Hybrid-Coordinator* durchgeführt.

Shortest Path Q-Learning (SPQL) wird in [76] angewendet, um verschiedene Relationen in einem zweischichtigen *Action Selection Mechanism* (ASM) zu erlernen. Dieses Zweischichtenmodell besteht aus einem *reactive behavior plan layer* und einem *action pattern layer*. Über das reactive behavior plan layer wird eine Aufgabe anhand der Motivation eines Systems ausgewählt, welche sich wiederum aus den Sensordaten ergibt. Der der Aufgabe zugehörige *reactive behavior plan* wird als Sequenz von Handlungsmustern, welche wiederum einfache Motoranweisungen beinhalten, ausgeführt. In den beiden Ebenen werden jeweils mit SPQL die Zusammenhänge zwischen Wahrnehmung und Handlungsmustern (*reactive behavior plan layer*) zur Verfolgung der eigenen Motive, und der Zusammenhang von Sensordaten und Motoranweisungen (*reactive behavior plan layer*) erlernt.

Learning Classifier Systems Learning classifier systems sind eine Kombination von reinforcement learning mit genetischen Algorithmen. Die einfachste Version stellt die *population* in der Form von einfachen Regeln dar:

IF state THEN action

Diese Regeln werden zufällig ausgeführt und die *fitness* je nach positiver/ ausbleibender Belohnung der Umgebung angepasst. In erweiterten Versionen des Algorithmus können die Regeln ihre Belohnung vorhersagen [17].

Evolving Neural Net In evolved neural nets werden genetische Algorithmen auf Neuronale Netze angewandt. Dabei kann der genetische Algorithmus auf verschiedene Teile des Netzes, z.B. die Gewichte der Verbindungen, die Architektur des Netzes (d.h. die Anzahl der Knoten und der Verbindungen) oder aber das eigentliche Lernen des Netzes angewandt werden.
Floreano et al. [17] geben eine Übersicht über neuroevolutionäre Methoden und deren Anwendung, wobei sich z.B. ein mobiler Roboter an dynamische Umgebungen anpassen kann und Agenten nützliche Verhalten erlernen können.
Neuronale Netze mit genetisch codierten Gewichten lernen schneller und besser als Netze mit zufällig verteilten Gewichten. Ein Vorteil der Kombination von neuronalen Netzen und genetischen Algorithmen ist, dass verschiedene Aspekte des neuronalen Netzes genetisch codiert und gleichzeitig weiterentwickelt werden können [17]. Außerdem ist das Training von anderen Kombinationen von neuronalen Netzen, wie z.B. dem Adaptive Heuristic Critic, schwieriger [17].

Case-Based Reasoning Ein Ansatz zum Erlernen reaktiver Verhalten durch das Demonstrieren möglicher Kombinationen von Sensorinformationen und Motoraktionen wird in [53] vorgestellt. Hierbei wird ein vierbeiniger Roboter manuell durch eine reale Umgebung gesteuert und dieser erlernt währenddessen, wie ein Mensch auf bestimmte Situationen reagieren würde. Die Zusammenhänge von Sensordaten und Motoranweisungen werden durch *Case-Based Reasoning* hergestellt.

Kombination Verhaltenssteuerung und Vorhersagemodell

Die Verhaltenssteuerung auf Grundlage von Fuzzy-Logik kann mit Vorhersagetechniken kombiniert werden, um dem Aufwand der Erstellung von Regeln zur Verhaltensgenerierung zu entgehen und adaptives Verhalten an dynamische Umgebungen zu ermöglichen.

Fuzzy-Logik und Reinforcement Learning Ein Vorteil von *Reinforcement Learning* stellt die Tatsache dar, dass das Verfahren keine definierten Inputs benötigt, wobei es sich aber nicht an ändernde Umgebungen anpassen kann [48]. Neue Verfahren wie das *Supervised Reinforcement Learning* [48] basieren auf einer Erfahrungsdatenbank oder besserer Aktionsgenerierung [23] und können Verhalten schneller erlernen. Die statistische Natur von Reinforcement Learning lässt Agenten jedoch unter Umständen vorgeschlagene korrekte Wege als Lösung ablehnen. Die Tatsache, dass Agenten Aktionen erst üben müssen, um auf Erfahrungen zurückgreifen zu können, macht das Einbeziehen von Unsicherheit schwierig.
Lin [38] stellt einen *reinforcement learning adaptive fuzzy controller* vor, der in der Lage ist, das *cart-pole balancing problem* zu lernen, in dem es darum geht, einen kippbaren, stehenden Stab auf einer mobilen Plattform zu stabilisieren.

Fuzzy-Logik und neuronale Netze Kumar [35] kombiniert Fuzzy-Logik mit neuronalen Netzen zur effizienten Generierung von Regeln und macht dabei die Regeln auch interpretierbar. Da beide Verfahren auf berechenbaren Funktionen beruhen, können Eingabeparameter der Fuzzy-Logik auch für neuronale Netze genutzt werden und bei Anwendung von *competitive neural networks* generiert sich das Netz selbst.

Die Kombination von Fuzzy-Logik und neuronalen Netzen ist unter anderem auch unter dem Begriff ANFIS (Adaptive Neuro-Fuzzy Inference System) bekannt [9]. Weiterführende Arbeiten stellen Netze vor, die Topologien lernen können und sich selbst generieren (Growing neural gas [19]).
Das Modell eines neuronalen Netzes zur Pfadfindung wird von der Startposition zur Zielposition des Agenten berechnet, wobei entweder die Position und die Distanz oder aber die Bewegungsrichtung (bei selbstgenerierenden Netzen) Parameter darstellen. Die Berechnung solch eines Modells ist oft sehr zeitintensiv [14]. Abbildung A.6 zeigt ein neuronales Netz, das zur Pfadfindung in einem Raum mit mehreren Hindernissen genutzt werden kann.

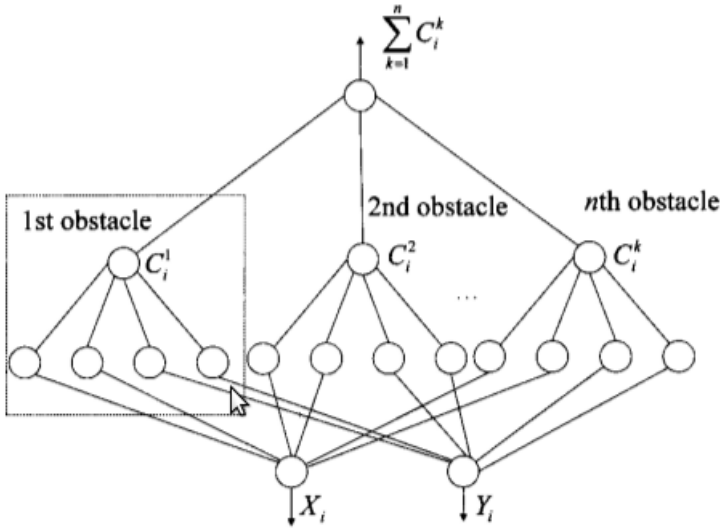


Abbildung A.6 Ein neuronales Netz zur Umfahrung mehrerer Hindernisse. Dabei wird von einem beliebigen Startpunkt im Koordinatensystem (X,Y) die gewichtete Distanz zu den Hindernissen errechnet und nur solche Pfade ausgewählt, die eine Kollision vermeiden.

Ruan et al. [59] kombinieren fuzzy neural networks mit reinforcement learning und demonstrieren, dass ein Agent damit die Balance behält.
Der große Vorteil von neuronalen Netzen liegt darin, dass sie auch nach der initialen Trainingphase weiterlernen und sich damit sehr gut an ändernde Bedingungen anpassen können.

Fuzzy-Logik und genetische Algorithmen Auch genetische Algorithmen[3] haben sich vor allem wegen den Möglichkeiten zur Optimierung in der autonomen Pfadfindung von Robotern in Zusammenhang mit Fuzzy-Logik etabliert ([35], [83]), wobei die frühen Versionen genetischer Algorithmen auch unausführbare Wege generieren, was sich negativ auf die Laufzeit des Algorithmus auswirkt. Neuere Versionen genetischer Algorithmen fügen Funktionen zur Hindernisvermeidung hinzu und vermeiden die Generierung von unausführbaren Wegen zur besseren Performanz und führen damit zum effizienten Ausweichen von Hindernissen und Ausführen eines gegebenen Ziels.

[3] Ein genetischer Algorithmus ist eine allg. verwendbare, globale Heuristik zur Lösung von Entscheidungsproblemen, wobei Teillösungen generiert werden, deren Überlebensfähigkeit bewertet und die dann mittels einer fitness function gekreuzt und mutiert werden um eine immer bessere Lösung für das gegebene Problem zu finden.

Genetische Algorithmen bieten eine robuste und unter Umständen weniger zeitintensive Alternative zu der Anwendung von neuronalen Netzen [14]. Durch Optimierung des genetischen Algorithmus kann die Performanz sogar noch gesteigert werden[70]. Schon in Abbildung A.7 kann man erkennen, warum die Berechnung genetischer Algorithmen potentiell weniger zeitintensiv ist. Die Tatsache, dass zweidimensionale Koordinaten hier in eindimensionalen Tabellen gespeichert werden, auf die eine einzelne *fitness function* angewandt wird, ist in der Regel wesentlich effizienter als die Netzerstellung und Berechnung in neuronalen Netzen.

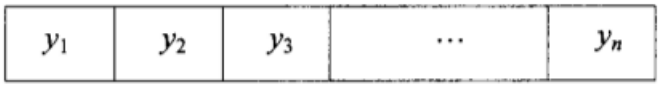


Abbildung A.7 Speicherung der Koordinaten von Hindernissen für die Anwendung einer fitness function des genetischen Algorithmus.

A.2 Real. Vorhersage und Selbstbewertung: Weitere Arbeiten

Softwareframework zur Evaluation von Vorhersagemethoden

Zur Evaluation und zum direkten Vergleich der vorgeschlagenen Vorhersage- und Klassifikationsmethoden wird das am DFKI in Python entwickelte Softwareframework "Signal Processing And Classification Environment" (pySPACE) verwendet.

pySPACE Bei pySPACE handelt es sich um ein modulares Softwareframework zur digitalen Datenverarbeitung. Dabei lassen sich Module in beliebiger Reihenfolge hintereinander schalten, austauschen und modifizieren: So lassen sich einfach flexible Datenverarbeitungsflüsse realisieren. Es existiert bereits eine breite Auswahl an Modulen aus den Bereichen der digitalen Signalverarbeitung (z.B. Frequenzfilter, räumliche Filter) und des maschinellen Lernens (z.B. Support Vector Maschinen, Lineare/Quadratische Diskriminanzanalyse), sowie Module zur automatischen Parameteroptimierung und solche zur Datenvisualisierung. Zudem bietet das Framework die Möglichkeit, über definierten Interfaces neue Methoden zu implementieren. Zusätzlich können auch bestehende Algorithmen, die beispielsweise in C/C++ Bibliotheken realisiert sind, über entsprechende Python-Wrapper in das Framework eingebunden und dort verwendet werden. Das Framework lässt es zudem zu, die Verarbeitungsflüsse

systematisch parallel auszuführen. Dadurch lassen sich bequem beliebig viele Kombinationen aus unterschiedlichen Methoden, Parameterkonfigurationen und Datensätzen auswerten. Im Folgenden können die Resultate, insbesondere Klassifikationsergebnisse, übersichtlich visualisiert werden.

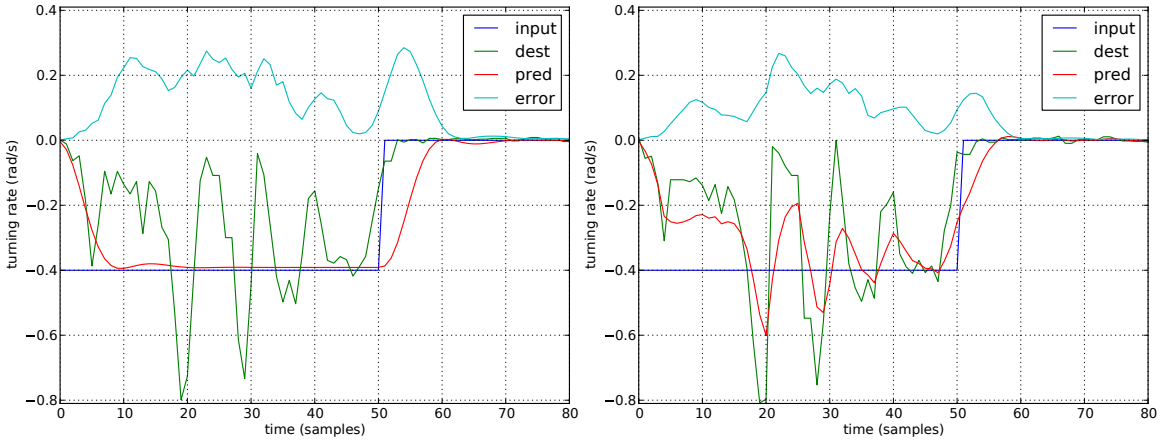
Das pySPACE Framework kann nun dazu genutzt werden, aus den verschiedenen Lernverfahren dasjenige, welches am besten für eine Vorhersage von bestimmten Sensordaten geeignet ist, auszuwählen und es gleichzeitig mit den gesammelten Trainingsdaten auf das Problem zu optimieren. Hierbei können die Parameter der jeweiligen Lernverfahren durch pySPACE verändert und somit optimal gewählt werden, indem die Performance des parametrisierten Verfahrens auf den Testdaten automatisch ausgewertet wird. Ein fertig trainiertes Verfahren mit seinen optimalen Parametern kann dann in eine Orogen-Komponente übertragen und im Robotersystem in der Rock-Umgebung zur Vorhersage von Sensordaten verwendet werden. Die für das MLP gewählte Lösung wurde bereits integriert und basiert auf der C++ Bibliothek "openANN[4]".

Verbesserung der Vorhersage des analytischen Models durch den Kalman Filter Das analytische PT_3 Model und der Kalman Filter, wurden in einem weiteren Schritt kombiniert um den Vorhersagefehler in verrauschten Umgebungen zu minimieren. Abbildung A.8 zeigt die Vorhersage für das Drehen auf dem Kraterboden gegen einen Stein, dessen Auswirkung auf das Drehen zwischen Sample 5 und 15 zu sehen ist.

Abbildung A.8(a) zeigt wie sich die Vorhersage ("pred", rot) unbeeinflusst von den tatsächlichen Werten ("dest", grün) und nur nach gelernten Modell verhält. In Abbildung A.8(b) ist die Vorhersage für das durch den Kalman Filter aktualisierte Vorhersagemodell abgebildet. Dabei werden die internen Zustandsvariablen des gelernten analytischen Modells durch tatsächliche Messungen angepasst. Die Wahl des Prozess- und des Messrauschens beeinflusst dabei, in wie weit der Vorhersage (Prozess) und der Messung vertraut wird. Es ist zu beobachten, dass sich die Vorhersage der tatsächlichen Messung annähert.

Der Vorhersagefehler wurde wieder über den MAE der letzten 5 Vorhersagen errechnet. Der Vergleich der Vorhersagen und des Vorhersagefehlers ohne (a) und mit (b) Aktualisierung des internen Zustands durch den Kalman Filter zeigt, dass sich die Vorhersage den verrauschten Sensordaten anpassen kann und der Vorhersagefehler dadurch verringert wird. Wird das

[4] https://github.com/OpenANN/OpenANN, Stand: Dezember 2013



(a) Vorhersage ohne Kalman Filter (b) Vorhersage mit Kalman Filter

Abbildung A.8 Vergleich der Vorhersage des analytischen PT_3 Models für das Drehen gegen den Stein ohne (a) und mit (b) Kalman Filter. Als Diagonalelemente für die Kovarianz Matrizen wurden gewählt: 1e-5 für den initialen Schätzfehler, 1e-3 für das Prozessrauschen und 5e1 für das Messrauschen.

Prozessrauschen weiter erhöht, so hat die Messung stärkeren Einfluss auf die Aktualisierung des Modells. Die Vorhersage würde sich dann stärker den gemessenen Werten anpassen und der Vorhersagefehler ließe sich weiter verringern.

Anwenden ließe sich der Kalman Filter in Umgebungen mit schwer vorhersagbaren Sensordaten wie beispielsweise dem Gyroskop auf dem Krater. Das Vorhersagemodell würde weiterhin in der Umgebung gelernt werden (siehe Abbildung A.8(a)) jedoch könnte man als weiteren Schritt das Modell in der Anwendung durch Messungen aktualisieren. Die Parameter des Kalman Filter müssten dann im Vorfeld durch Messungen in repräsentativen Umgebungen festgelegt werden oder zur Laufzeit, z.B. durch Messung des Rauschen des Sensors, dessen Werte vorhergesagt werden sollen. Mit dem Kalman Filter steht dann eine Möglichkeit bereit, fehlerhafte Vorhersagen herauszufiltern. Als Alternative zum Kalman Filter wurde im Rahmen des AP4400 zusätzlich ein Frequenzfilter implementiert, der als Tiefpass-Filter ebenfalls verrauschte Sensordaten filtern soll. Beide Filter-Methoden müssen im weiteren Untersuchungen verglichen werden.

Experimentelle Ergebnisse

Hier werden nun einige weitere Ergebnisse der Methoden NG und MLP auf den in den vorigen Abschnitten beschriebenen Testdaten detaillierter beschrieben.

Ergebnisse für das NG Für das Training wurden (wie oben beschrieben) mehrere Wiederholungsläufe einer Roboterbewegung durchgeführt und Motorkommandos und Sensorwerte aufgezeichnet. Abbildung A.9 zeigt beispielhaft diese Daten für eine Drehung auf der Stelle.

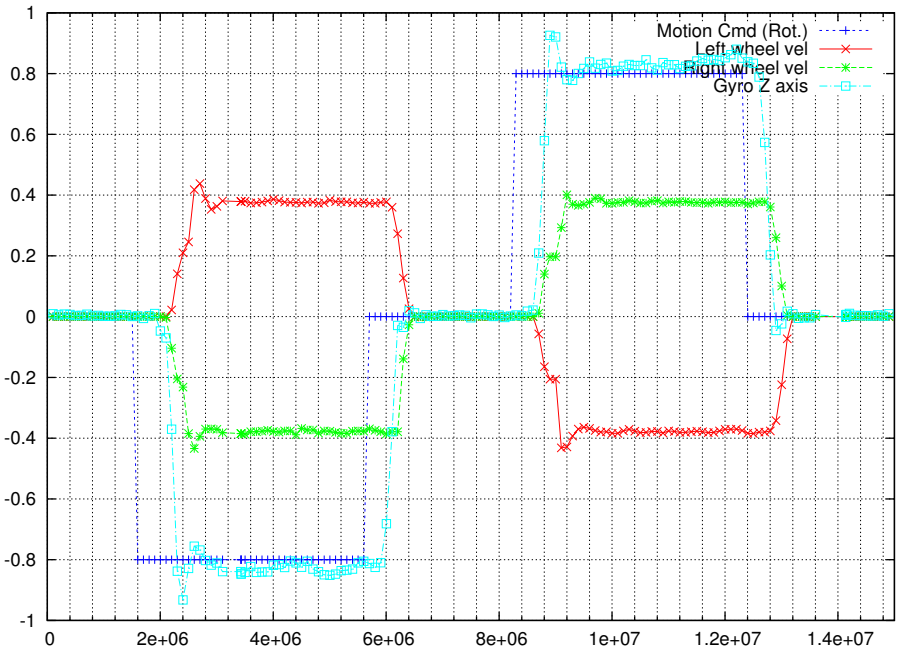


Abbildung A.9 Motorkommandos, Radgeschwindigkeiten und Drehgeschwindigkeit (mit Hilfe des Gyroskops gemessen) um die Z-Achse.

Das Ergebnis des Lernens lässt sich an Abbildung A.10 gut erkennen. Initial sind die Center-Vektoren zufällig verteilt (grün im Diagramm). Am Ende des Lernens decken die Vektoren (blau) sehr gut die Traingsdaten (rot) ab.

In Abbildung A.11 ist das Lernergebnis für die linke und die rechte Radgeschwindigkeit zu sehen. Neben den Foci im Stand (Geschwindigkeiten 0/0), während der Rechtsdrehung und während der Linksdrehung sind auch gut gelernte Daten im Übergangsbereich zu erkennen. Während erstere für die spätere Vorhersage des stabilen Zustands (Steady-State) benötigt

werden, bilden letztere Center-Vektoren das dynamische Verhalten ab.

Der letzte Plot in Abbildung A.12 zeigt, dass auch beliebige komplexere, nicht-gausssche Verteilungen abgebildet werden. Zu erkennen sind wieder die drei "Steady-State"-Punkte sowie die dynamischen Übergänge.

Ergebnisse für das MLP Die Fähigkeit des MLP die den Sensordaten zugrunde liegende Dynamik (y) zu lernen wurde anhand der in diesem Abschnitt eingangs beschriebenen Szenarien getestet. Wie bereits für die vorangegangenen Methoden beschrieben, wurde auch hier der aufgezeichnete Datensatz für die Drehung auf dem Hallen- und dem Kraterboden zum Training eines MLP's verwendet. Das Ergebnis der Vorhersage wurde dann mit den tatsächlichen Werten eines Testdatensatzes verglichen.

Für die hier dargestellten Analysen wurde zunächst ohne systematische Optimierung der Netzarchitektur und der Parameter gearbeitet. Diese wird später in der Framework-Architektur pySPACE noch einmal genauer analysiert. Hier wurde zunächst ein fester Parametersatz

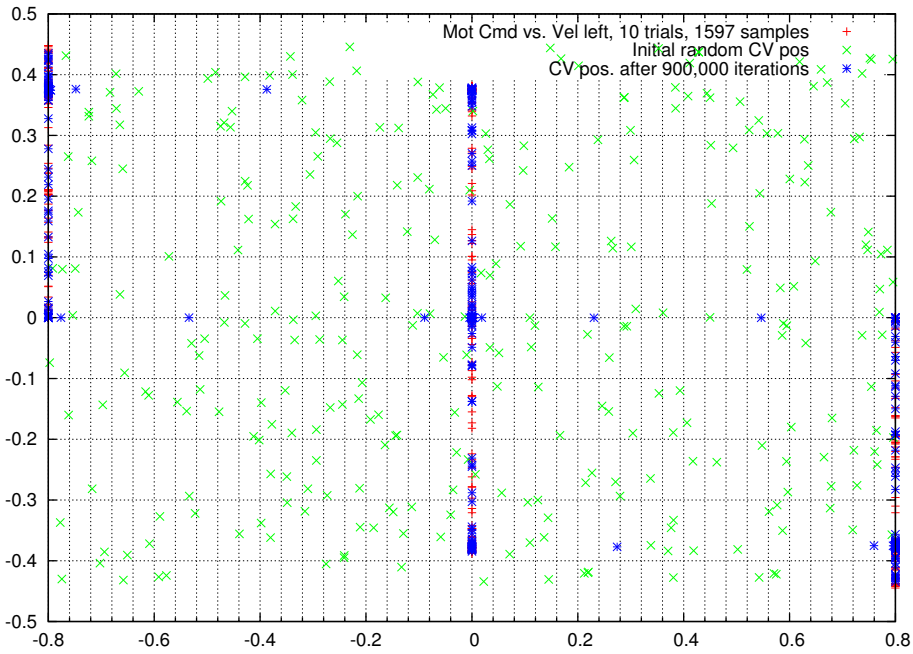


Abbildung A.10 Initiale CV, CV am Lernende und Trainingsdaten für die linke Radgeschwindigkeit über dem Motorkommando.

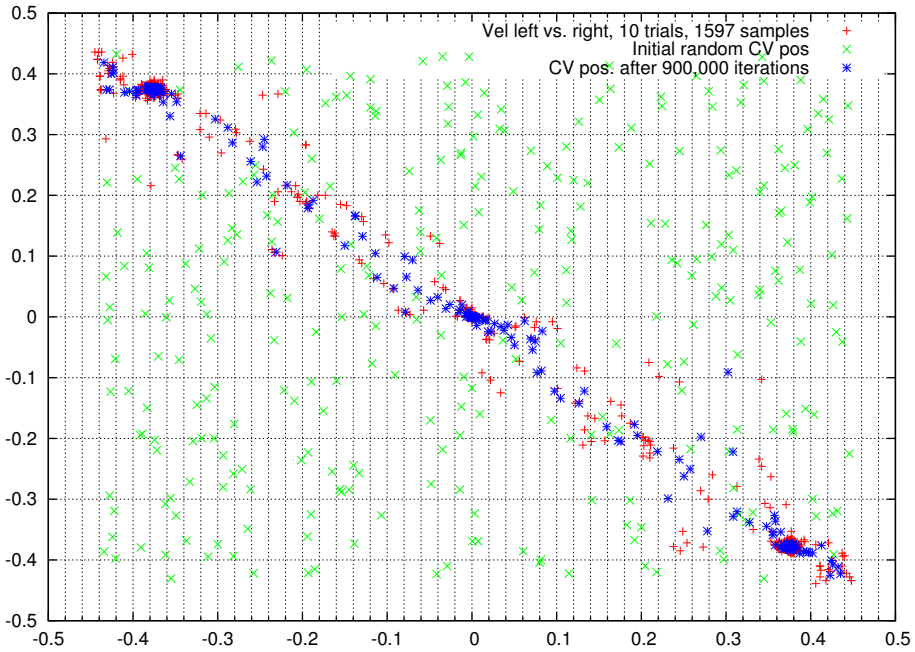


Abbildung A.11 Initiale CV, CV am Lernende und Trainingsdaten für die linke und die rechte Radgeschwindigkeit.

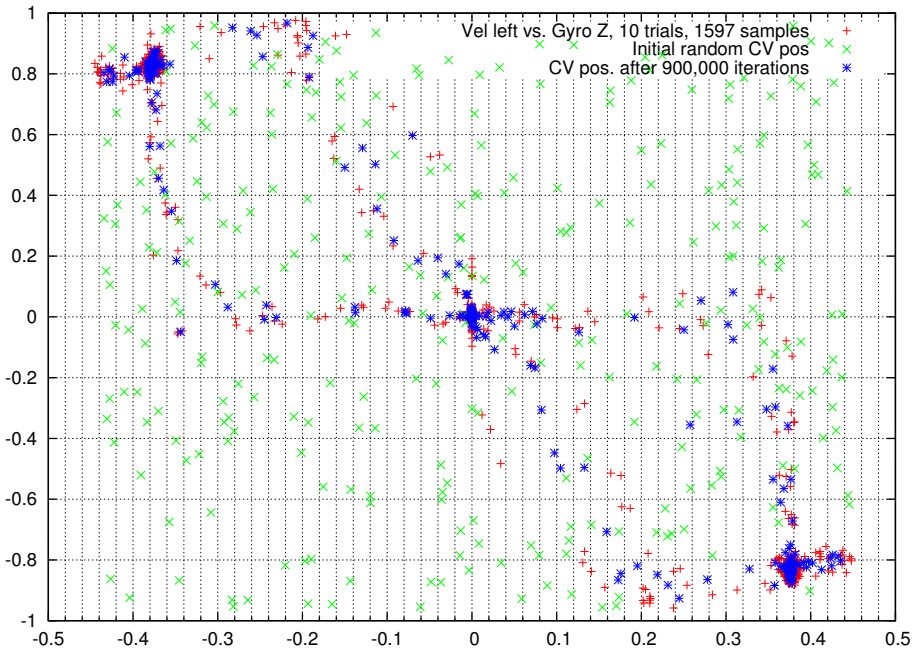


Abbildung A.12 Initiale CV, CV am Lernende und Trainingsdaten für die linke Radgeschwindigkeit und der Gyro-Z-Drehgeschwindigkeit.

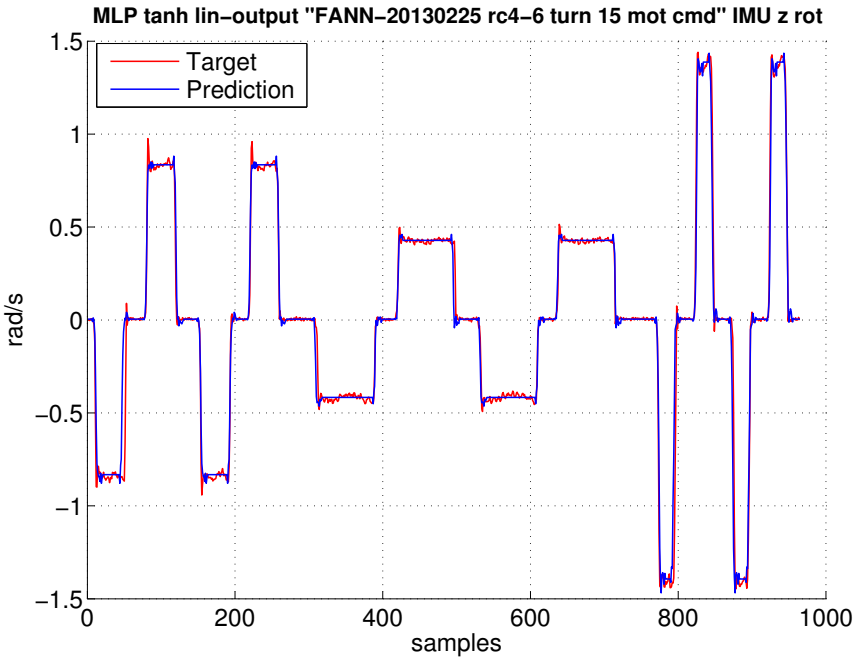


Abbildung A.13 Die Grafik zeigt die Vorhersage für verschiedene Drehgeschwindigkeiten (±0.8, ±0.4 und ±1.4 rad/s).

und eine feste Architektur verwendet. Das MLP hatte hier zwei hidden layer. Das erste mit 12 und das zweite mit 4 Neuronen. Die Aktivierungsfunktion der hidden layer ist tanh und die Ausgabeschicht verwendet eine lineare Aktivierungsfunktion. Insgesamt wurde das Netz jeweils mit 10.000 Epochen trainiert, allerdings zeigten die Ergebnisse, dass es bereits nach ca. 2.500 Epochen keine signifikanten Änderungen mehr gab.

Die Abbildung A.13 zeigt je zwei Beispiele für verschiedene Drehgeschwindigkeiten auf dem Boden der Halle, den vorhergesagten Sensorverlauf für t+1 und den tatsächlich zum Zeitpunkt t+1 gemessenen. Das MLP hat die Vorhersage für die Sensoren auf einem größeren Datensatz mit je 8 Beispielen pro Geschwindigkeit gelernt. Wobei die jeweils zwei Wiederholungen die hier zum testen verwendet wurden nicht in den Trainingsdaten enthalten waren. Die dargestellte Vorhersage basiert wie für den Datensatz beschrieben auf den letzten 15 Motor Kommandos. Man sieht hier, dass die vom Netz gelernte Dynamik (blau) sehr gut zu der tatsächlich eintretenden (rot) passt und die tatsächlich gemessenen Werte auch nur relativ leicht um den Grundwert rauschen.

Im Gegensatz zu der ersten Darstellung zeigt die Abbildung A.14 die Vorhersage der verschie-

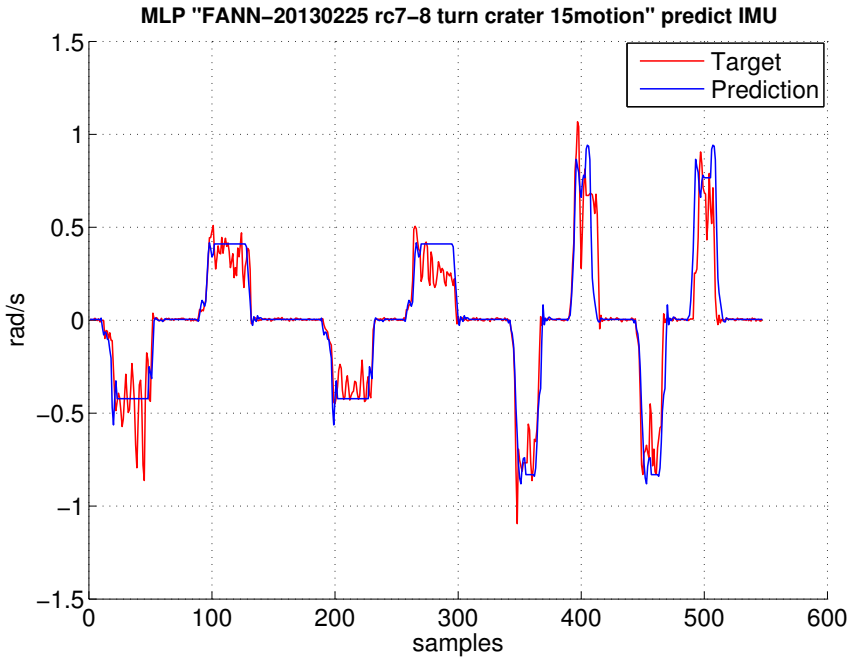


Abbildung A.14 Die Grafik zeigt die Vorhersage für die Geschwindigkeiten (±0.4 und ±1.4 rad/s) auf dem Kraterboden.

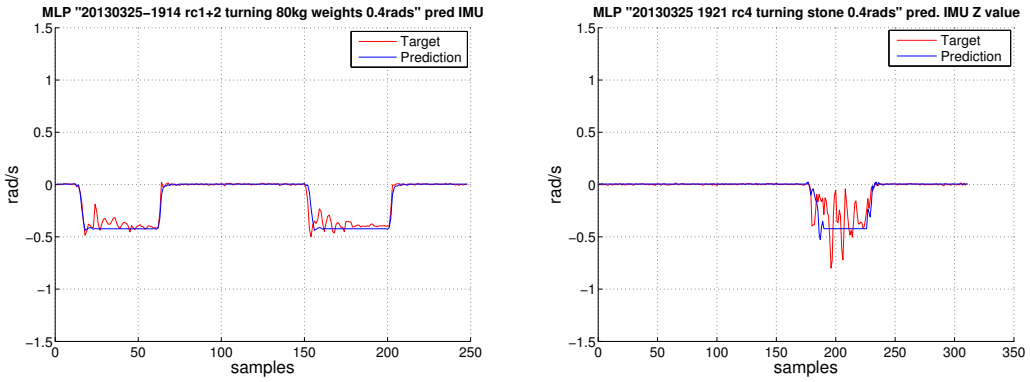


Abbildung A.15 Diese Grafik zeigt links (a) zwei Versuche bei dem sich der Roboter mit -0.4 rad/s auf dem Hallenboden gegen ein 80 kg schweres Hindernis dreht. Im rechten Teil der Abbildung (b) sieht man die Drehung gegen einen verankerten Stein auf dem Kraterboden.

denen Geschwindigkeiten auf dem Kraterboden. Der Trainingsdatensatz besteht hier ebenfalls aus Daten die auf dem Kraterboden aufgenommen wurden. Obwohl die Daten hier offenbar deutlich stärker verrauscht (gestört) sind konnte das Netz hier die Dynamik (blau) wieder recht gut lernen. Allerdings zeigt sich, dass die tatsächlich gemessenen Werte (rot) des Sensors hier deutlich stärker um den Grundwert rauschen.

In Abbildung A.15 wird die Vorhersage (blau) und die gemessenen Werte (rot) für den Fehlerfall dargestellt. In Teil (a) der Abbildung der Fall bei dem sich der Roboter gegen ein Hindernis auf dem Hallenboden dreht und in Teil (b) der Fall bei dem sich der Roboter gegen den Stein auf dem Kraterboden dreht. Die Berührung mit dem Stein erfolgt in beiden Fällen relativ kurz nach Beginn der Drehung.

Im Vergleich zu Abbildung A.13 und Teil (a) von Abbildung A.15 lässt sich erkennen, dass die Störung am Anfang der Drehung hier relativ deutlich im Unterschied zum Normalfall steht. Die stärkeren Ausschläge entgegen der intendierten Drehrichtung lassen sich hier zu Beginn der Drehung recht gut erkennen, bis sich der Roboter von dem Hindernis wieder "befreit" hat. Der Vergleich von Teil (b) der Abbildung mit Abbildung A.14 zeigt, dass hier im Fehlerfall zu Beginn der Berührung mit dem Stein ebenfalls ein Unterschied erkennbar ist. Denn zunächst bleibt die rote (Target) Linie deutlich "gleichmäßiger" unter dem erwarteten Werte zurück. Allerdings ist für die Erkennung hier eine größere Rauschtolleranz notwendig, da auch im Normalfall die werte kurzfristig stark von der Erwartung abweichen. Zu erkennen ist, dass

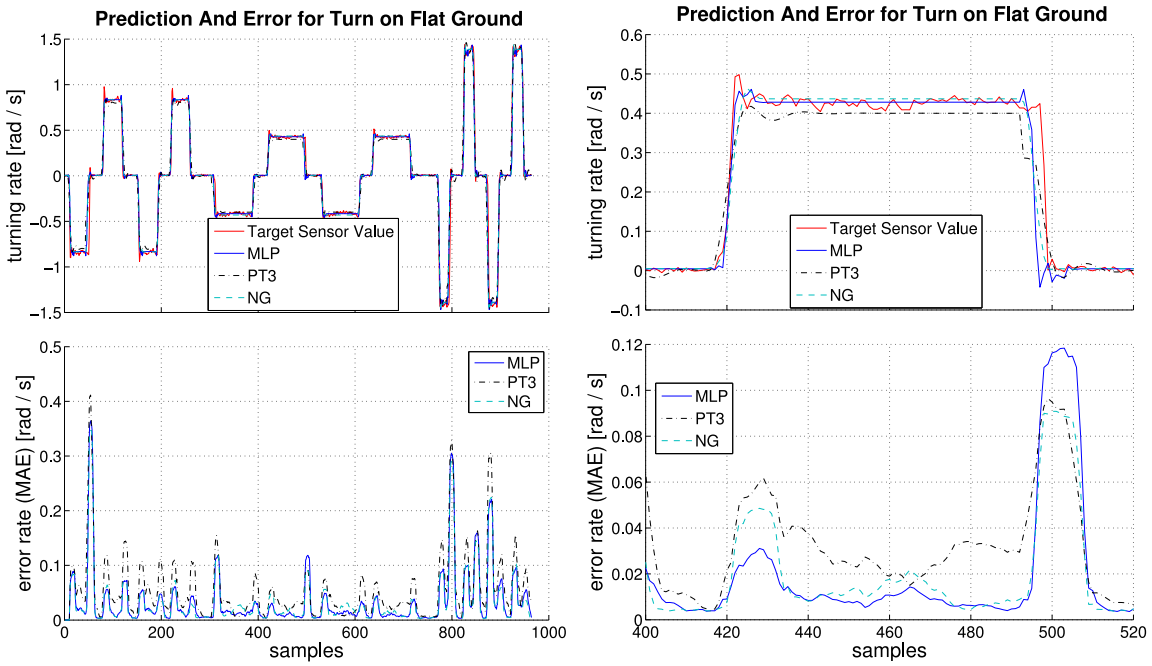
während der Störung auch die Varianz im Rauschen abzunehmen scheint. Hier muss noch detaillierter geprüft werden ob diese Kriterien zur robusten Detektion des Fehlers geeignet sind.

Vergleich der Vorhersagemethoden zur Fehlererkennung Im Kontext des Projektes soll die Vorhersage dazu eingesetzt werden, Fehler in den Abläufen von Aktionen zu erkennen um auf Störungen reagieren zu können. Externe Störungen, z.B. durch andere Agenten oder durch Umwelteinflüsse, können vom Roboter nur erkannt werden, indem eine Erwartung (also eine Vorhersage) dessen erzeugt wird, was der Roboter in naher Zukunft an Sensorrückflüssen erwartet. Beispielsweise wird erwartet, dass eine Drehung um die Z-Achse in der IMU gemessen wird, wenn der Roboter seine Motoren so ansteuert, dass er sich rechts oder links herum dreht. Eine Störung von außen kann nun Eintreten, wenn der Roboter durch ein zuvor "übersehenes" Hindernis in der Ausführung seiner Tätigkeit behindert wird. Im Falle einer Drehung beispielsweise soll der Roboter dieses Problem erkennen und zunächst die Drehung stoppen und/oder sich wieder von dem Hindernis entfernen.

Wie bereits beschrieben, haben wir einen Datensatz aufgenommen, bei dem sich der Roboter in verschiedenen Geschwindigkeiten auf flachem Untergrund gedreht hat. Hierbei wurden alle drei zuvor vorgestellten Lernverfahren (PT3, MLP, NG) auf den Trainingsdaten optimiert und die gelieferte Vorhersage für die Z-Drehung die von der IMU gemessen wird auf den Testdaten getestet. Zum Vergleich sind in Abbildung A.16 die tatsächlich gemessenen Werte (rote Kurve) zusammen mit den von den einzelnen Verfahren vorhergesagten Werten dargestellt. Im unteren Teil der Abbildung ist für jede Methode der über die letzten 10 Vorhersagen gemittelte absolute Fehler aufgetragen. Die Teilabbildung A.16(b) stellt hierbei einen Ausschnitt der Teilabbildung A.16(a) dar. Die gleiche Analyse wurde auch für die auf dem Kraterboden aufgezeichneten Trainings- und Testdaten durchgeführt. Erwartungsgemäß war hierbei das Grundrauschen in den Sensordaten aufgrund des unebenen Untergrundes deutlich stärker.

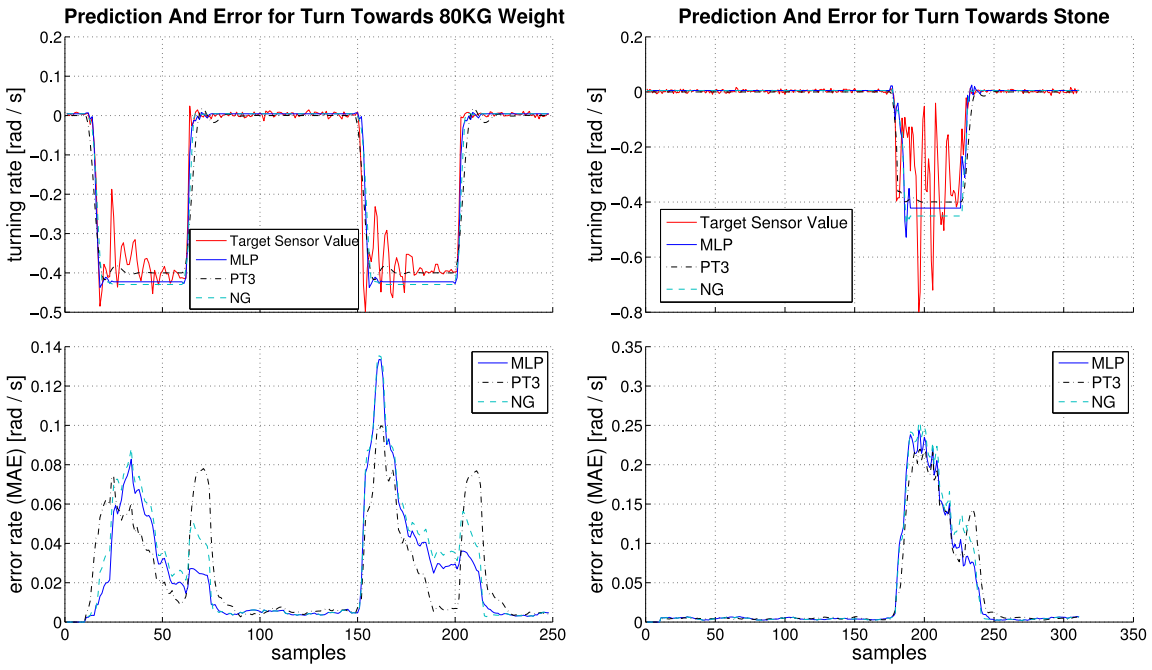
Um nun untersuchen zu können, inwieweit sich die Fehlerfälle, also Störungen von außen, auf die Sensordaten auswirken, wurden zusätzlich noch einige Datensätze aufgenommen bei denen eine Störquelle enthalten war. Hierbei wurde der Roboter auf dem Kraterboden durch einen am Kratermodell befestigten Stein und auf dem flachen Hallenboden durch ein ca. 80 kg schweres Gewicht in seiner Drehung behindert. Die zugehörigen Vorhersagen und Fehler werden analog zum vorher beschriebenen Normalfall in Abbildung A.17 dargestellt.

Insgesamt lässt sich vor allem in Abbildung A.16 erkennen, dass alle drei Vorhersagemetho-



(a) (b)

Abbildung A.16 Ergebnisse der verschiedenen Vorhersagemethoden für die Vorhersage der Drehung um die Z-Achse die von der IMU gemessen wird. Im Rechten Teil der Abbildung wird ein vergrößerter Ausschnitt der linken Grafik dargestellt, der eine Drehung des Roboters nach rechts mit einer Drehgeschwindigkeit von 0.4 rad/s zeigt. Der untere Teil der Grafiken zeigt jeweils den über die letzten 10 Vorhersagen gemittelten absoluten Fehler. (Grafik aus Veröffentlichung [55])



(a) (b)

Abbildung A.17 In dieser Abbildung werden die beiden Fehlerfälle dargestellt. Links wird der Roboter in seiner Drehung auf dem flachen Untergrund durch ein 80 kg schweres Gewicht behindert und Rechts bei der Drehung auf dem Kraterboden durch den am Krater befestigten Stein.(Grafik aus Veröffentlichung [55])

den brauchbare Ergebnisse erzielen, wobei die beiden neuronalen Lernmethoden gegenüber der Analytische (PT3) etwas besser abschneiden. Für alle drei Methoden lässt sich in Teilabbildung A.16(a) sowie in dem vergrößerten Ausschnitt in Teilabbildung A.16(b) erkennen, dass die Fehler in den stabilen Phasen einer Drehung oder beim Stillstand relativ gering sind, da die eintretenden Sensorwerte recht gut vorhergesagt werden können. Bei den hier durchgeführten Experimenten stand der Roboter still und hat dann für eine gewisse Zeit auf eine vorgegebene Drehgeschwindigkeit beschleunigt und anschließen wieder relativ abrupt gestoppt. Dies führt zu Beginn und zum Ende einer jeden Drehbewegung zu einer relativ stark ansteigenden beziehungsweise abfallenden Flanke in der von der IMU gemessenen Drehung um die Z-Achse. Aufgrund dieser relativ steilen Flanken führt bereits eine kleine Abweichung der Vorhersage vom tatsächlichen Zeitpunkt des Beginns der Drehung bzw. deren Ende zu einem relativ großen Fehler. Dies führt zu den in Abbildung A.16 erkennbaren Ausschlägen der Fehlerkurven an den jeweiligen ansteigenden und abfallenden Flanken der Drehbewegungen. Zu beachten ist hier, dass die Vorhersagen der Signalverläufe selbst relativ gut zu den gemessenen Signalverläufen passen. Die Verzögerung zwischen Motorkommando und Sensorantwort variiert allerdings ein wenig und kann in der aktuellen Konfiguration nicht vorhergesagt werden. Daher soll in Zukunft versucht werden z.B. auf Basis der Änderungen in den Motorkommandos auch die zu erwartende Varianz des Signals zu lernen und somit eine fehlerhafte Erkennung von Störungen zu verhindern.

In Abbildung A.17 ist zu erkennen, dass alle Verfahren für die tatsächlichen Fehlerfälle, bei denen der Roboter von einem Objekt in der Ausführung seiner Drehung behindert wird, einen in der Regel höheren Ausschlag als im Normalfall erzeugen. Vor allem aber lässt sich erkennen, dass der Ausschlag der Fehlerkurve hier viel länger anhält und somit die Spitzen in den Fehlerkurven breiter sind. Dieses Ergebnis legt also nahe, dass auch mit der im vorigen Absatz vorgeschlagenen "Dämpfung" der Fehlerausschläge mittels der Varianz eine Erkennung dieser Fehlerzustände möglich ist, da die Fehlerkurven bei den Flanken im Normalfall deutlich "schärfer" sind. Die hier dargestellten Ergebnisse wurden im Projekt bereits in eine Veröffentlichung umgesetzt, siehe hierzu [55].

Kombination mehrerer Sensormodalitäten Da sich unerwartete Situationen, wie etwa das Fahren gegen ein übersehenes Hindernis, für den Roboter häufig in mehreren Sensormodalitäten zeigen, wurde die Vorhersagen mehrerer Sensorantworten Untersucht. Hierzu wurden die Methoden MLP und NG verwendet.

Das Testszenario ist wieder eine Drehung des Roboters auf der Stelle. Untersuchte Modali-

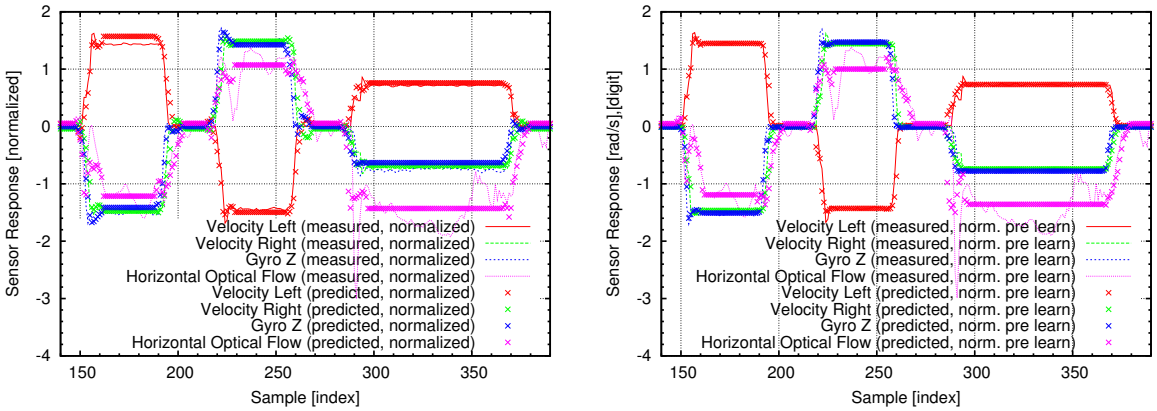


Abbildung A.18 Beispielverläufe von gemessenen und vorhergesagten Sensorsignalen. Für die Daten im linken Diagramm wurde ein mit einem MLP realisiertes Modell verwendet. Im Diagramm rechts sind Vorhersagen mit einem mit NG gelernten Modell zu sehen. Die gemessenen Sensorwerte sind in beiden Diagrammen identisch. Diese Ergebnisse sind veröffentlicht in [34].

täten sind jetzt die Radgeschwindigkeiten rechts und links, die Rotation des Gyroskops um die Z-Achse und der horizontale optische Fluss. In Abbildung A.18 zeigt exemplarisch die Sensorantworten und die jeweiligen Vorhersagen basierend auf einem mittels MLP und einem mit NG gelernten Modell.

Die Vorhersagefehler für die vier Modalitäten lassen sich in Abbildung A.19 vergleichen. Für beide Verfahren, MLP und NG, zeigen sich in den Histogrammen prinzipiell die gewünschten hohen Spitzen rund um einen Vorhersagefehler von 0. Negativ fallen in beiden Varianten jedoch die Vorhersagefehler für den optischen Fluss auf. Dies begründet sich in der starken Variation dieses Signals auch während einer gleichmäßigen Drehung (vergleiche Abbildung A.18). Die Fehler bei den anderen Modalitäten, insbesondere den Radgeschwindigkeiten, sind deutlich niedriger.

Für beide Verfahren, MLP und NG, wurden verschiedene Varianten getestet. Beim MLP wurde zum einen ein einzelnes MLP für alle vier Sensormodalitäten gelernt und zum anderen ein eigenes MLP nur für den optischen Fluss. Siehe hierzu auch die Beschreibungen zum MLP in Kapitel 2.1.5. Der Grund hierfür ist das spezielle Verhalten dieses Sensors. Wie in Tabelle A.1 zu sehen ist, verbessert sich die Vorhersage im Falle eines separaten MLP leicht. Zu beachten ist, dass die Messdaten für den optischen Fluss für die Ergebnisse in der ersten Tabellenzeile

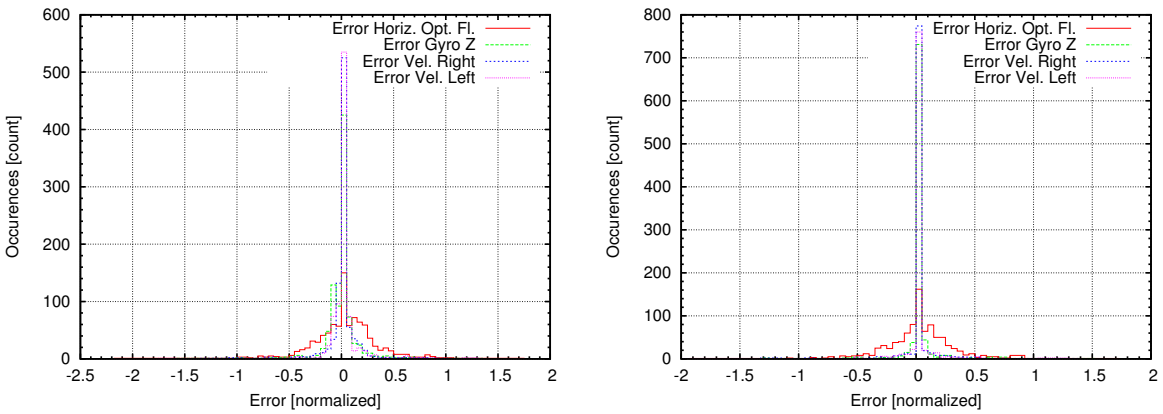


Abbildung A.19 Histogramme der Vorhersagefehler für die vier Sensormodalitäten. Gezeigt sind wieder Ergebnisse beider Verfahren (MLP links, NG rechts). Diese Ergebnisse sind veröffentlicht in [34].

Methode	*Bedingung*	*RMSE Mod. 1*	*RMSE Mod. 2*	*RMSE Mod. 3*	*RMSE Mod. 4*
1 – MLP	einzelnes MLP	0.173	0.160	0.185	0.333
2 – MLP	separates MLP optischer Fluss				0.326
3 – NG	keine Vorverarbeitung	0.225	0.217	0.272	0.482
4 – NG	normalisierte Training-Daten	0.165	0.145	0.172	0.331
5 – NG	norm. Dat., Recall mit Messwert.	0.144	0.114	0.150	0.279

Tabelle A.1 Vergleich der Vorhersagefehler. Die Fehlerwerte der Varianten, die rohe Messwerte verwenden (Zeilen 1 bis 3) wurden mit der Standardabweichung der Trainingsdaten skaliert. Für die anderen beiden Konfigurationen (Zeilen 4 und 5) wurden schon die Testdaten selbst genauso skaliert. Dadurch sind die gezeigten Vorhersagefehler sowohl für alle fünf Varianten als auch für alle vier Sensormodalitäten vergleichbar. Die vier Sensormodalitäten ("Mod.") sind (1) Radgeschwindigkeit links, (2) Radgeschwindigkeit rechts, (3) Gyroskop Y-Achse, (4) horizontaler optischer Fluss. Diese Ergebnisse sind veröffentlicht in [34].

(ein gemeinsames MLP für alle Modalitäten) vorher auf einen Mittelwert von 0.0 und eine Standardabweichung von 1.0 normiert wurden (siehe MLP-Beschreibungen oben).

Bei NG wurden in einer Variante die gemessenen Sensorwerte ohne Vorverarbeitung für das Training verwendet (dritte Zeile in der Tabelle). In einer zweiten Variante wurden zunächst alle Modalitäten auf einen Mittelwert von 0.0 und Standardabweichung 1.0 normiert (Zeile 4). Wie zu sehen ist, verbessert dies bei den verwendeten Testdaten die Ergebnisse deutlich.

In einer dritten NG-Variante wurden im Recall auch die jeweiligen gemessenen Sensorwerte zur Auswahl des besten Center Vector (CV) (und der damit verbundenen Vorhersage) verwendet. Hierdurch reduziert sich der Fehler naturgemäß (siehe Zeile 5), da solche CV ausgewählt werden, die möglichst dicht an den gemessenen Werten liegen. Die Distanz zwischen Vorhersage (CV) und gemessenem Wert ist dadurch gering.

Alle vorgestellten Ergebnisse zur Vorhersage mehrerer Modalitäten wurden in [34] veröffentlicht.

Berichtsblatt

1. ISBN oder ISSN	2. Berichtsart (Schlussbericht oder Veröffentlichung) Schlussbericht

3a. Titel des Berichts

Verbundvorhaben: VirGo4 - Vorhersagesysteme in reaktiven Gruppen autonomer Roboter (Abschlussbericht der Universität Bremen)

3b. Titel der Publikation

4a. Autoren des Berichts (Name, Vorname(n))

Berghöfer, Elmar; Dominguez, Raul; Duda, Alexander; Feess, David; Goldhoorn, Malgorzata; Joyeux, Sylvain; Kording, Michaela; Planthaber, Steffen; Rauch, Christian; Rehrmann, Felix; Schröer, Martin; Tiedemann, Tim

4b. Autoren der Publikation (Name, Vorname(n))

5. Abschlussdatum des Vorhabens
30.6.2014

6. Veröffentlichungsdatum

7. Form der Publikation

8. Durchführende Institution(en) (Name, Adresse)

Universität Bremen, Fachbereich 03
AG Robotik
Robert-Hooke-Straße 1
28359 Bremen

9. Ber.-Nr. durchführende Institution

10. Förderkennzeichen
50RA1114

11a. Seitenzahl Bericht
122

11b. Seitenzahl Publikation

12. Literaturangaben
83

13. Fördernde Institution (Name, Adresse)

Raumfahrtmanagement des
Deutschen Zentrums für Luft- und Raumfahrt e.V.
Königswinterer Str. 522-524
53227 Bonn

14. Tabellen
1

15. Abbildungen
55

16. Zusätzliche Angaben
--

17. Vorgelegt bei (Titel, Ort, Datum)
--

18. Kurzfassung

Um die Entwicklung von Software für Robotersysteme zu vereinheitlichen, deren Zuverlässigkeit zu erhöhen und Kooperation zwischen mehreren Systemen zu unterstützen, verfolgte VirGo4 im Wesentlichen zwei Ziele:
1. Eine plattformunabhängige Entwicklungsmethodik umgesetzt in einem Verhaltensframework zur Erleichterung der Integration heterogener Softwarekomponenten in ein konsistentes, modulares Gesamtsystem. Die Ressourcen des Robotersystems können transparent bereitgestellt werden, ohne beim Nutzer Kenntnisse über die Kommunikation oder die interne Implementierung der Ressourcen zu erfordern.
2. Eine Architektur zur Verhaltenssteuerung autonomer Roboter wurde entwickelt, die die Steuerung von Robotern in zukünftigen Weltraummissionen erleichtern und robuster gestalten kann. Zur Erkennung von unerwarteten Ereignissen, d.h. auch von dem System unbekannten Störungen, werden gelernte Vorhersagen der Auswirkungen von Aktionen des Roboters mit Messungen von Umwelt oder dem Zustand des Systems selbst verglichen. Ein Roboter kann diese Information verwenden, um sein Verhalten wenn notwendig zu adaptieren.
Diese biologisch motivierte Selbstüberwachung wurde bisher noch nicht im Kontext der Weltraumrobotik realisiert. Im Vorhaben verwendet wurden dazu Methoden des maschinellen Lernens (MLP, Neural Gas). Zur Realisierung des Software-Frameworks (Punkt 1 oben) kam das Robotik-Framework Rock (basierend auf Orocos) zum Einsatz.
Bei Projektabschluss wurden die gestellten Ziele erfüllt. In 8 Demonstrationen (7 auf einem mobilen Outdoor-Roboter, eine simuliert) konnten die Ergebnisse erfolgreich präsentiert werden. Die während der Laufzeit des Vorhabens gemachten Erfahrungen zeigten sowohl auf Seiten des Verhaltensframeworks als auch bei der biologisch inspirierten, auf gelernten Daten basierenden Selbstüberwachung große Vorteile. Insgesamt wurden so Verfahren und Software-Komponenten geschaffen, die eine Prädiktion und Selbstevaluation beinhaltende, mehrschichtige Verhaltenssteuerung von mobilen Robotern ermöglichen.

19.Schlagwörter

Vorhersage, Selbstevaluation, Verhaltenssteuerung, Software-Framework, Weltraumrobotik

20. Verlag	21. Preis

Document Control Sheet

1. ISBN or ISSN	2. Type of Report Final Report

3a. Report Title

Verbundvorhaben: VirGo4 - Vorhersagesysteme in reaktiven Gruppen autonomer Roboter (Abschlussbericht der Universität Bremen)

3b. Title of Publication

4a. Author(s) of the Report (Family Name, First Name(s))

Berghöfer, Elmar; Dominguez, Raul; Duda, Alexander; Feess, David; Goldhoorn, Malgorzata; Joyeux, Sylvain; Kording, Michaela; Planthaber, Steffen; Rauch, Christian; Rehrmann, Felix; Schröer, Martin; Tiedemann, Tim

4b. Author(s) of the Publication (Family Name, First Name(s))

5.End of Project
June 30th, 2014

6. Publication Date

7. Form of Publication

8. Performing Organization(s) (Name, Address)

Universität Bremen, Fachbereich 03
AG Robotik
Robert-Hooke-Straße 1
28359 Bremen

9. Originator's Report No.

10. Reference No.
50RA1114

11a. No. of Pages Report
122

11b. No. of Pages Publication

12. No. of References
83

13. Sponsoring Agency (Name, Address)

Raumfahrtmanagement des
Deutschen Zentrums für Luft- und Raumfahrt e.V.
Königswinterer Str. 522-524
53227 Bonn

14. No. of Tables
1

15. No. of Figures
55

16. Supplementary Notes
--

17. Presented at (Title, Place, Date)
--

18. Abstract

To unify the development of software for robotic systems, to increase their reliability and to support cooperation among multiple systems, VirGo4 followed mainly two objectives:
1. A platform-independent development methodology realized in a behaviour control framework to ease the integration of heterogeneous software components in a consistent modular system. The resources of the robot can be used transparently without the need to know about the communication details or the internal implementation of the resource.
2. An architecture for a behaviour-based control of autonomous robots was developed that can ease the control of robots in future space missions and that can make it more robust. To detect unexpected events (potentially including faults that were not thought of at design time), learned predictions of action consequences are compared with incoming sensor or internal state data. The robot can use these comparison results to adapt its behaviour, if required. This biologically-inspired self evaluation has not yet been realized in the context of robotic space missions. To accomplish this, machine learning methods were used in the project (Neural Gas, multi layer perceptron, MLP). To realize the software framework (point 1 above), the robotic framework Rock (based on Orocos) had been used.
Within the project all objectives could be fulfilled. In 8 demonstrations (7 on an outdoor robot, one in simulation) the results could be successfully shown. The experiences gained during the project show strong advantages of both, the behaviour control framework as well as the biologically inspired self evaluation. In summary, methods and software components were created that enable a multi-layer behaviour control of mobile robots which include prediction and self-evaluation.

19.Keywords

Prediction, self-evaluation, behaviour control system, software framework, space robotics

20. Publisher	21. Price