+++ neuer name +++
hochschule für angewandte
wissenschaften hamburg

university of applied sciences
gegr. 1970 fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIK

Diplomarbeit

Jan Philip Porschke

Genetische Programmierung von verhaltensbasierten Agenten

Studiengang Softwaretechnik
Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Klauck
Abgegeben am 16. Januar 2002

BERLINER TOR 3 · D - 20099 HAMBURG

Genetische Programmierung von verhaltensbasierten Agenten

Stichworte

Agenten, mobile Roboter, genetische Programmierung, maschinelles Lernen

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der genetischen Programmierung von verhaltensbasierten Agenten. Sie dokumentiert einen Laborversuch, in dem ein genetisches Verfahren das Verhalten eines mobilen Roboters hinsichtlich einer gegebenen Aufgabenstellung optimieren soll. Der Roboter kommuniziert dabei über eine Infrarotschnittstelle mit dem auf einem externen Rechner laufenden genetischen Optimierer.

Das im Rahmen der Diplomarbeit entstandene Programm ist für das Robot–Lab der Hochschule für Angewandte Wissenschaften Hamburg gedacht. Diese Arbeit dokumentiert neben der Versuchsdurchführung das Design und die Implementierung des beschriebenem Agentensystems. Die Analyse der Versuchsdurchläufe soll klären, ob die oben beschriebene Aufgabe mit dem beschriebenem System gelöst werden kann.

Genetic Programming of Behaviour Based Agents

Keywords

Agents, Mobile Robots, Genetic Programming, Machine-based Learning

Abstract

This thesis deals with the use of genetic algorithms to optimize the behaviour of an embodied agent (mobile robot). The thesis decribes an experiment in which a gentic method is used to optimize the behaviour of a mobile robot in repect to a specific task. During the experiment the robot exchanges data via an infrared-interface with a genetic optimizer running on an external computer.

The application developed in conjunction with this thesis is intend for the robot-lab of the university of applied siences Hamburg. Apart from describing the setup of the experiment the thesis contains a desciption of the design and the implementation of the agent system. The analysis of the test-runs is used to answer the question if the aforementioned task can be successfully completed by using this system.

Inhaltsverzeichnis

Einleitung **6**

1 Agenten **9**
1.1 Der Agenten Begriff 9
1.2 Taxonomie der Agenten 12
1.3 Agenten Architekturen 13
1.3.1 Modellbasierte Entscheidungsfindung 14
1.3.2 Reaktive Entscheidungsfindung 14

2 Maschinelles Lernen **16**
2.1 Lernen 16
2.2 Wissensrepräsentation 17
2.2.1 Symbolisch-kognitivistische Repräsentation 17
2.2.2 Konnektionistische Repräsentation 18
2.2.3 Interaktionistische Repräsentation 18
2.3 Lernverfahren 19
2.3.1 Konnektionistisches Lernen 19
2.3.2 Genetische Programmierung 19

3 Die Problemstellung **28**

4 Das Design **30**
4.1 Die virtuelle Maschine 31
4.1.1 Die Regelbasis 32
4.1.2 Der Regelinterpreter 36
4.1.3 Die Fitness 37
4.1.4 Die Vorgangssteuerung 39
4.1.5 Sequenzdiagramm der Selbstinspektion 40
4.2 Die Evolutionsmaschine 41
4.2.1 Das Genom 41
4.2.2 Die Populationsparameter 42

4.2.3 Das Abbruchkriterium 43
4.2.4 Die Transformationverfahren 44
4.2.5 Das Selektionsverfahren 46
4.3 Das Kommunikationsmodul 48
4.3.1 Die Architektur des Kommunikationsmoduls 48

5 Die Implementierung 50
5.1 Der Roboter 50
5.1.1 Lego Mindstorm RIS 50
5.1.2 Die Hardware Konfiguration 51
5.2 Das Versuchsareal 52
5.3 Die virtuelle Maschine 53
5.3.1 Das Betriebsystem 53
5.3.2 Das BehaviourPattern Objekt 53
5.3.3 Das Interpreter Objekt 55
5.3.4 Das Fitness Objekt 60
5.3.5 Das FitnessDeterminationProcess Objekt 60
5.4 Die Evolutionsmaschine 62
5.4.1 Die GAlib 62
5.4.2 Das RCXInterpreterWord Objekt 64
5.4.3 Das GATreeGenom Objekt 66
5.4.4 Das GAPopulation Objekt 68
5.4.5 Das GASteadyStateGA Objekt 69
5.4.6 Die Fitness Funktion 70
5.5 Das Kommunikationsmodul 71
5.5.1 Das Lnp Protokoll 71
5.5.2 Das SMWL Protokoll 72
5.5.3 Die IRCommunicator Objekthierarchie 74
5.5.4 Der RCX Communicator 74
5.5.5 Der PC Communicator 76
5.6 Die Debugger Objekthierarchie 78
5.6.1 Das Debugger Objekt 79
5.6.2 Das RCXDebugger Objekt 79
5.6.3 Das PCDebugger Objekt 80

6 Die Versuchsdurchführung 81
6.1 Die untersuchten Eigenschaften 81
6.1.1 Die Checkliste 82
6.1.2 Der Stammbaum des Genoms 84
6.2 1. Versuch 85
6.2.1 Die Versuchsparameter 85

6.2.2 Versuchsdurchlauf 1 85
6.2.3 Versuchsdurchlauf 2 90
6.2.4 Schlussfolgerungen 92
6.3 2. Versuch 92
6.3.1 Die Versuchsparameter 93
6.3.2 Checkliste 93
6.4 3. Versuch 95
6.4.1 Die Versuchsparameter 96
6.4.2 Checkliste 96
6.4.3 Schlussfolgerungen 98
6.5 4. Versuch 98
6.5.1 Die Versuchsparameter 99
6.5.2 Checkliste 99
6.5.3 Schlussfolgerungen 101

7 Zusammenfassung & Aussichten 102
7.1 Zusammenfassung 102
7.2 Kritische Würdigung & Aussichten 103

Literaturverzeichnis 107

A Versuchs Logs 109

B Sourcecode 111

Einleitung

Kaum eine Technologie hat seit der Industrialisierung in den 30er Jahren eine so grundlegende Veränderung des täglichen Lebens bewirkt, wie der Einsatz von Computern. Ganze Wirtschaftszweige haben ihre Produktionsabläufe restrukturiert und durch massiven Einsatz von Computern und Robotern ihre Produktivität gesteigert. Der Umgang mit gängigen Textverarbeitungsprogrammen ist oft grundlegende Anforderung für eine Einstellung.

Schon jetzt ist die Beherrschung komplexer Programme jedoch nur mit erheblicher Einarbeitung möglich. Zudem erfordern viele Arbeitsabläufe trotz intensiver Unterstützung durch den Computer menschliche Steuerung.

Schnell drängt sich bei diesen Aussichten der Wunsch nach einer Art niederem Stellvertreter auf, der unsere Anweisungen selbstständig in die richtigen Arbeitsschritte umsetzt. Solche Agenten könnten uns beim täglichen Umgang mit dem Heimcomputer unterstützen oder für uns den Wochenendeinkauf erledigen. Sie könnten zu fernen Planeten reisen und in für den Menschen unwirtlichen Umgebungen im Sinne ihrer Auftraggeber handeln.

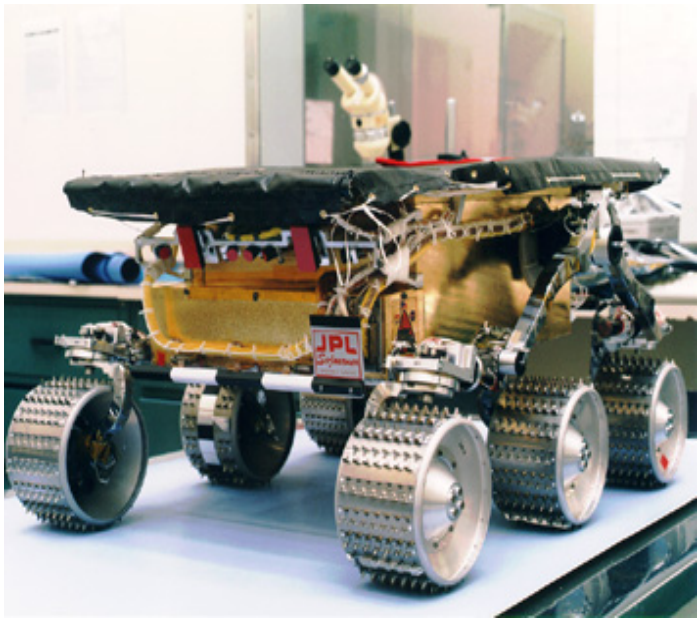


Abbildung 1: N.A.S.A. Roboter Sojourner

Einen der wohl populärste Vertreter der Agententechnologie entwickelte die N.A.S.A., die 1997 einen mobilen Roboter zum Mars sendete, um mit ihm Bodenanalysen durchzuführen. Die unbekannte Struktur der Marsoberfläche und der nicht genau vorhersehbare Ort der Landung schienen ebenso problematisch wie die Übertragungszeit der Funksignale. Bis zu 4 Minuten benötigte das verwendete Funksystem um, ein Bild der Umwelt vom Mars zur Erde zu senden [1]. Leicht lässt sich erkennen, dass bei dieser Latenz das direkte Steuern eines Roboters keinen Sinn macht. Zur Lösung dieses Problems wurde der Sojourner (siehe Abbildung 1) von einem Programm gesteuert, das in der Lage war, vorgegebene Etappenziele auf dem Weg zu einem Versuchsort selbstständig zu erreichen.

Autonome Roboter untersuchen schon heute die Innenwände von Kanalsystemen [2] und messen sich im Roboterfußball [3]. Die Programmierung solcher Maschinen ist jedoch eine hohe Kunst, da sie fast alle Bereiche der Informatik in sich vereint. Zusätzlich erfordern dynamische oder unbekannte Umgebungen die Fähigkeit des Agenten, sich an neue Situationen anzupassen.

Das maschinelle Erlernen von Verhaltensregeln zur Steuerung autonomer Roboter ist zentraler Untersuchungsgegenstand des in diesem Papier dokumentierten Versuchs. Der Versuch soll im speziellen die Frage beantworten, ob ein mobiler Roboter grundsätzlich in der Lage ist, sein Verhalten selbstständig so anzupassen, das er eine ihm gestellte Aufgabe zufriedenstellend lösen kann.

Zur Abgrenzung der Begrifflichkeiten werden in den ersten beiden Kapiteln die Bereiche der Agententechnologie und des maschinellen Lernens besprochen. Das Kapitel 3 beschreibt die exemplarische Problemstellung, die vom Roboter gelöst werden soll und skizziert dabei eine erste Architektur des Agenten.

Das 4. Kapitel beschreibt das Design des im Laborversuch verwendeten Agentensystem. Der Versuchsaufbau und seine Teilkomponenten werden beschrieben und ihre Auswahl begründet. Für den Versuch besonders relevante Module, wie die Lernkomponente, deren Erstellung und Konfiguration Hauptgesichtspunkt des Versuchs ist, werden dabei im Detail besprochen.

Kapitel 5 dokumentiert die Implementierung des im 4. Kapitel festgelegten Design. Es wird an dieser Stelle ebenfalls auf verwendete Bibliotheken und melchatronische Bauteile hingewiesen, deren detaillierte Dokumentation im Anhang zu finden ist.

Die zu untersuchenden Eigenschaften und Parameter des Lernvorgangs werden im 6. Kapitel beschrieben und in 4 Versuchsdurchläufen analysiert. Die gewonnenen Versuchsergebnisse werden hinsichtlich der Versuchskonfigurationen

analysiert und miteinander verglichen. Abhängigkeiten zwischen Versuchsparametern und der Erfolgsquote des entsprechenden Lernvorgangs werden analysiert und wenn möglich zu generellen Erkenntnissen abstrahiert.

Das 7. Kapitel bewertet die Versuchsdurchführung unter dem Aspekt der Gesamtfragestellung. Die Übertragbarkeit der gewonnenen Erkenntnisse auf andere Bereiche der Agentenorientierung wird bewertet und mögliche Erweiterungen der Systemarchitektur werden besprochen.

Kapitel 1

Agenten

1.1 Der Agenten Begriff

Was sind Agenten? Die eigentliche Übersetzung dieses Wortes lautet Stellvertreter. Der berühmte Geheimagent 007 agierte z.B. immer in der Sache Englands. Der Agent einer Sängerin immer im Sinne der Künstlerin. Was aber bedeutet dieser Begriff, wenn er in Verbindung mit Software genannt wird? Das Handbuch der Künstlichen Intelligenz definiert diesen Begriff wie folgt:

> ”Im Bereich der Software und der Robotik ist der Agent ein populärer Begriff zur Bezeichnung von intelligenten Programmen. Unter dem Namen agenten-orientierte Techniken entstehen neue Methoden und Strukturen, bei denen die Künstliche Intelligenz wesentliche Beiträge zu liefern hat. Dabei werden Ideen von intelligenten Akteuren, kognitiven Systemen und kooperierenden Agenten in realen und virtuellen Umgebungen aufgegriffen.” [5]

Abbildung 1.1: Geheimagent 007

Diese Definition ist im Kontext der Informatik sicher angemessener als der Vergleich mit einem Geheimagenten. Eine genaue Abgrenzung herkömmlicher Systemparadigmen von der Agententechnologie fehlt jedoch auch hier. Das Handbuch der künstlichen Intelligenz [5] klassifiziert Agenten auf Grund der verschiedenen Eigenschaften, die ein Agent besitzen kann:

- **Andauernde Verfügbarkeit/Aktivität:** Agenten sind über einen längeren Zeitraum hinweg ansprechbar und nehmen "Aufträge" von Nutzern oder anderen Agenten entgegen. Sie können eigenständig aktiv werden.

- **Interaktion mit einer Umwelt:** Agenten nehmen Informationen aus ihrer Umwelt auf. Sie agieren in ihrer Umwelt, um sie (auftragsgemäß) zu beeinflussen.

- **Situiertheit:** Auch hier geht es um die Einbettung von Agenten in einer Umwelt. Es wird der Aspekt betont, dass ein komplexes Agentenverhalten auch als Resultat von direkten Reaktionen auf Umwelteinflüsse erzeugbar ist ("emergentes Verhalten").

- **Eigenständigkeit (Autonomie) im Handeln:** Agenten unterliegen keiner (unmittelbaren) Steuerung und Kontrolle durch den Nutzer. Sie handeln eigenständig, aber "im Sinne ihrer Auftraggeber". Die Auswahl/Planung ihrer Handlungen kann (aber muss nicht) nach sehr komplexen Methoden erfolgen.

- **Reaktivität:** Im engeren Sinne wird darunter das unmittelbare Reagieren auf Umweltereignisse ("Stimulus-Responsus") verstanden. Im allgemeinen betrifft es die Interaktion mit der Umwelt insgesamt.

- **Zielgerichtetheit:** Agenten verfolgen Ziele bzw. Aufträge, die angepasstes Handeln über lange Zeiträume hinweg erfordern können.

- **Pro-Aktivität:** Der Begriff ist verwandt mit Zielgerichtetheit (und wird teilweise gleichwertig benutzt). Eine spezielle Betonung liegt dabei auf der "Eigeninitiative" des Agenten.

- **Deliberatives Verhalten:** "Deliberation" bezeichnet die explizit modellierte Auswahl von Zielen bzw. Absichten. Auch dieser Begriff wird als Gegensatz zu reaktivem Verhalten gebraucht.

- **Intelligenz:** Der Begriff kann im Sinne analoger Erklärungen des Begriffs "Künstliche Intelligenz" so interpretiert werden: Agenten vollführen Handlungen, die beim Menschen als intelligent gelten würden.

- **Rationalität:** Agenten treffen sinnvoll erscheinende Entscheidungen in angemessener Zeit auch unter dem Aspekt beschränkt verfügbarer eigener Ressourcen.

- **Lernfähigkeit:** Agenten können ihr Verhalten an die Umwelt anpassen, indem sie

  zum Beispiel Fähigkeiten oder Entscheidungsprozesse geeignet variieren.

- **Mobilität:** Agenten können eigenständig auf andere Plattformen (Rechner) migrieren. Dabei wird das Agenten-Programm mit seinen aktuellen Daten übertragen und setzt seine Arbeit dort fort mit der Möglichkeit des Zugriffs auf lokale Ressourcen.

- **Kooperation:** Agenten arbeiten zusammen mit Menschen und anderen Agenten.

- **Wohlwollen (benevolenz):** Agenten führen Aufträge nach Möglichkeit im Sinne der Auftraggebers aus, wobei auch andere Agenten Auftraggeber sein können.

- **Soziales Verhalten:** Agenten halten sich bei ihrer Arbeit mit anderen Agenten (und Menschen) an vorgegebene Regeln und Normen. Sie bilden Gruppen oder Koalitionen. Vorbild sind soziale Strukturen der Menschen.

- **Emotionales Verhalten:** Emotionen sollen einerseits das Verhalten steuern (sie treten damit neben andere Begriffe wie Ziele, Absichten usw.), und sie sollen die Interaktion mit dem Menschen verbessern.

- **Glaubwürdigkeit:** Die Erscheinungsform (z.B. Gestik animierter Agenten) soll sich in glaubhafter Übereinstimmung mit ihren Aktionen befinden. Agenten sollen als Individuen mit eigenen Zielen, Bedürfnissen und Emotionen erscheinen.

Würde ein Programm dann als Agent gelten, wenn es mindestens eine dieser Eigenschaften besitzt, könnte ein Großteil heutiger Informationssysteme als Agentensysteme angesehen werden. Tatsächlich stellt der Agentenbegriff eher die Vereinigung und Weiterführung bestehender Ansätze zum Design von Programmen und ihren Schnittstellen dar, als eine scharf abzugrenzende Programmklasse.

Die in bekannten Textverarbeitungsprogrammen verwendeten ”Personal Assistens” können beispielsweise mit einem intelligenten Workflowmanager verglichen werden, der eine generelle Absicht des Benutzers in eine Abfolge von Befehlen an das System übersetzt. Auch das automatische Vervollständigen von in einer

Website eingegebenen Benutzernamen stellt unter diesem Aspekt eine Agententypische Leistung dar. Der Agent (in diesem Fall der Internet Browser) erkennt die Absicht des Benutzers, seinen Namen einzugeben und bietet ihm selbstständig die Möglichkeit zur automatischen Vervollständigung an. Das agententypische Verhalten kommt also immer dann zum Zuge, wenn es darum geht, verschiedene Handlungsalternativen gegeneinander abzuwiegen, um diese in eigenständige Handlungen umzusetzen.

1.2 Taxonomie der Agenten

Wie können wir uns solche Agenten vorstellen? Können wir sie anhand der oben genannten Kriterien in wenige Klassen einteilen? Die unten stehende Taxonomie versucht eine grobe Hierarchisierung der Agentenarten:

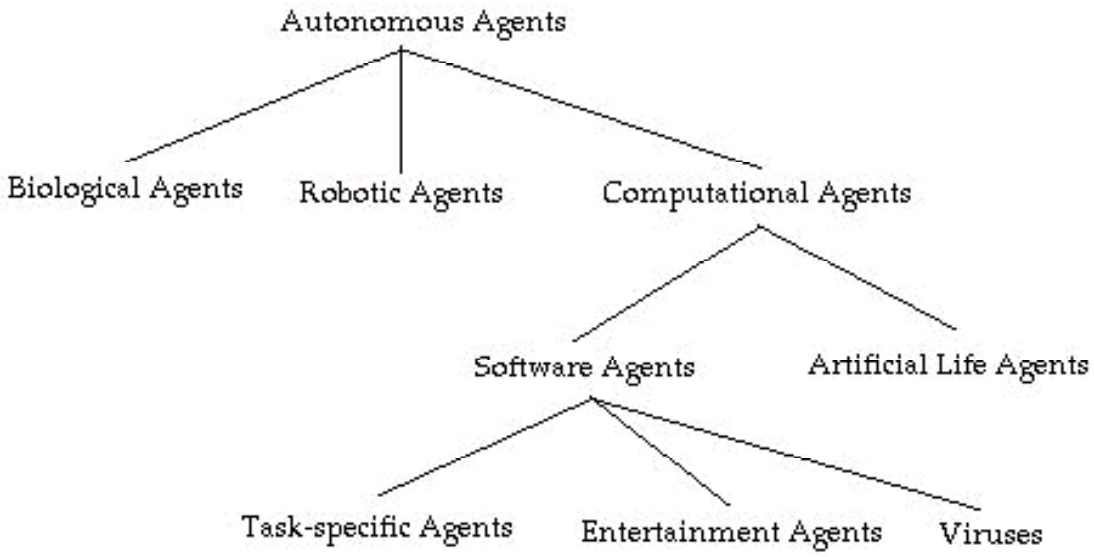


Abbildung 1.2: Taxonomie der Agenten

- **Biological Agents** Außen vor stehen die biologischen Agenten, deren Fähigkeiten in den meisten Fällen weit über die ihrer elektronischen Kollegen hinausreichen. Selbst der dümmste Wachhund schlägt, was seine allgemeine Fähigkeit zu Überleben angeht, den modernsten Roboter.

  Medizinische Substanzen wie Antibiotika können im entfernteren Sinne ebenfalls als biologische Agenten gesehen werden, da sie den vom Menschen bezweckten Heilprozess durch autonomes Handeln beschleunigen

oder ermöglichen. Die biologischen Kampfstoffe können als bösartige, biologische Agenten gesehen werden, die wie die elektronischen Viren zwar im Sinne ihres Auftraggebers handeln, jedoch ein moralisch zweifelhaftes Ziel verfolgen.

- **Robotic Agents** Die Roboter grenzen sich von den anderen elektronischen Agenten durch ihre Verkörperung ab. Sie verhalten sich im Gegensatz zu den Softwareagenten nicht in künstlichen, formalen Welten sondern in realen Umgebungen. Dies hat unmittelbaren Einfluss auf Architektur und Programmierung des Agentensystems.

- **Software Agents** Das Spektrum der Softwareagenten ist breit gefächert. Softwareagenten können von einem einfachen Suchmechanismus bis hin zu selbstmodifizierenden Viren reichen. Die Entertainment Agents besitzen beispielsweise keine formalisierbaren Ziele im herkömmlichen Sinne, da sie auf die Unterhaltung eines Menschen abzielen, wohingegen aufgabenspezifische Agenten in vielen Fällen ein unmittelbares Feedback bezüglich ihrer Handlungen beziehen können. Viren sind aufgrund ihrer Bösartigkeit zwar ein zweifelhaftes Mitglied der Agentenfamilie, handeln jedoch auch im Sinne ihrer Auftraggeber, denen sie wiederum wohlwollend gegenüber stehen.

1.3 Agenten Architekturen

Die Eigenständigkeit des Handelns ist eine Anforderung, die an jeden Agenten gestellt werden kann. Das dazu notwendige Verarbeiten von Wahrnehmungen und das daraus abgeleitete Handeln wird bei klassischen Agentensystemen in 3 Phasen der Informationsverarbeitung durchgeführt:

- **Phase der Informationsaufnahme** Der Agent nimmt Informationen aus der Umwelt auf. Er nimmt Nachrichten und Aufträge entgegen oder analysiert seine Umwelt durch Sensoren. Er beobachtet die Wirkung seiner Handlung.

- **Phase der Wissensverarbeitung und Entscheidung** Der Agent aktualisiert sein Wissen mithilfe der eingegangenen Informationen. Er analysiert und bewertet die aktuelle Situation, die neuen Aufträge und den Fortschritt bereits begonnener Handlungen. Er trifft Entscheidungen über seine unmittelbaren und künftigen Aktionen.

- **Phase der Aktionsausführung** Die anstehenden Aktionen werden ausgeführt.

Der Vorgang der Entscheidungsfindung kann auf verschiedene Weisen umgesetzt werden. Grundlegend unterscheidet die Agentenorientierung zwischen zwei Arten der Entscheidungsfindung: Dem modellbasierten und dem nicht modellbasierten Verhalten.

1.3.1 Modellbasierte Entscheidungsfindung

Die Schlussfolgerung von Aktionen durch die Betrachtung der Welt und ihres Zustandes ist das Kernstück intelligenter Handlungen eines Agenten. Die modellbasierte Entscheidungsfindung betrachtet neben den eingehenden Sensordaten stets auch ein internes Weltbild, welches sich aus bereits gemachten Erfahrungen oder vorgegebenen Informationen bildet.

Diese Informationen können von einer Karte der Umgebung bis hin zu komplexen Regelsystemen reichen. Diese Eigenschaft ermöglicht es modellbasierten Systemen, komplexe Pläne zur Lösung vorgegebener Probleme zu entwickeln und anzuwenden. In verknappter Form stellt sich die Informationsverarbeitung eines Modellbasierten Agenten wie folgt dar:

$$
\begin{aligned}
&repeat \\
&\qquad W := sense(S); \\
&\qquad Z_{neu} := thought(Z_{alt}, W); \\
&\qquad A := act(Z_{neu}); \\
&forever
\end{aligned}
$$

Die Funktion *thought(...)* repräsentiert hier die Durchführung von in einigen Fällen äußerst komplexen Berechnungen. Die vollständige Probagierung von hinzugewonnenen Fakten in einem Regelsystem, oder die Erstellung eines Planes auf Basis eines Situationskalküls kann schnell zu einer komplexen Aufgabe werden. Um die Rationalität eines solchen Agenten zu sichern, muss das System jedoch zu jedem Zeitpunkt in angemessener Zeit zu einem Ergebnis kommen. Die Komplexität des verwendeten Planungsalgorithmus muss also die möglichen Echtzeitanforderungen der Aufgabenstellung beachten. Diese komplexe Form der Entscheidungsfindung ist nicht immer notwendig und angemessen.

1.3.2 Reaktive Entscheidungsfindung

Reflexartiges Handeln kann in vielen Situation sinnvoller sein als zeitaufwändiges Überlegen. Gerade wenn ein schnelles Reagieren auf wenige Sensordaten

erfordert ist, macht die modellbasierte Entscheidungsfindung keinen Sinn. Die ”Stimulus-Response-Architektur” verzichtet auf komplexe Beweisverfahren oder Wahrscheinlichkeitsberechnungen. Sie begrenzt die Wissensverarbeitung auf ein einfaches Matching Verfahren, welches Mustern von Sensordaten eindeutig Aktionen zuordnet. Die Funktion *thought(...)* wird in unserer verkürzten Ansicht durch die Matchingprozedur *match(...)* ersetzt, welche die Adressierung einer Aktion in einer Look-up Tabelle repräsentiert:

$$
\begin{aligned}
&repeat \\
&\qquad W := sense(S); \\
&\qquad Z_{neu} := match(W); \\
&\qquad A := act(Z_{neu}); \\
&forever
\end{aligned}
$$

Die Funktion *match(...)* sucht den Eintrag an der Adresse W in der Look-up Tabelle und setzt diesen als neuen Zustand. Der alte Zustand wird dabei nicht mit in die Bestimmung des Folgezustands einbezogen.

Kapitel 2

Maschinelles Lernen

2.1 Lernen

Das Wort Lernen mag nicht bei allen Menschen mit positiven Erinnerungen assoziiert sein. Nicht viele werden aber bestreiten, dass Lernen ein notwendiger Vorgang zum Überleben eines Menschen in seiner Umwelt ist. Was aber bedeutet es, wenn eine Maschine lernt? Können wir den Lernvorgang einer Maschine feststellen oder messen? Stefan Wrobel definiert das maschinelle Lernen wie folgt:

> Eine Lernaufgabe wird definiert durch eine Beschreibung der dem lernenden System zur Verfügung stehenden Eingaben (ihrer Art, Verteilung, Eingabezeitpunkte, Darstellung und sonstiger Eigenschaften), der vom lernenden System erwarteten Ausgaben (ihrer Art, Funktion, Ausgabezeitpunke, Darstellung und sonstigen Eigenschaften) und den Randbedingungen des Lernsystems selbst (z.B. maximale Laufzeiten oder Speicherverbrauch). Ein System lernt erfolgreich genau dann, wenn es in der Lage ist, bei Eingaben, die den Spezifikationen entsprechen, unter den geforderten Randbedingungen Ausgaben mit den gewünschten Eigenschaften zu erzeugen. [6]

Wie aber kann ein Programm eine solche Lernaufgabe lösen? Üblicherweise versorgt der Mensch das Programm mit Informationen, während er es erstellt und immer dann, wenn eine Information nicht verfügbar ist. Es lernt also im gewissen Sinne vom Menschen. Dies ist sicherlich ein erprobter und oft sinnvoller Ansatz. Wollen wir aber von lernenden Agenten sprechen, widerspricht dies der Forderung nach der Autonomie des Handelns. Der Agent muss in der Lage sein, eigene Rückschlüsse aus seiner Wahrnehmungen zu ziehen, um sein Verhalten entsprechend anzupassen. Dies entspricht aus informationstechnischer Sicht einer Manipulation der vom Agenten verwendeten Wissensbasis.

Wie können wir uns diese Manipulationsverfahren vorstellen? Eine generalisierbare Eigenschaft dieser Manipualtionsverfahren ist der Vorgang der Selbstinspektion. Das Verfahren sammelt empirische Daten bezüglich seines Handelns, um es so bewerten und vergleichen zu können. Um die Qualität des Verhaltens eines formalen Systems zu optimieren, ist ein arithmetischer Vergleich der Qualität von verschiedenen Verhaltensmustern erforderlich. Die richtige Berechnung dieses numerischen Qualitätswertes entscheidet über den Erfolg des Lernvorgangs. Die speziellen Arbeitsweisen dieser Verfahren hängen von der gewählten Repräsentationsform des Wissens ab.

2.2 Wissensrepräsentation

Der Lernvorgang kann aus informationstechnischer Sicht als eine systematische Suche nach dem optimalem Verhalten im durch die Wissensrepräsentation aufgespannten Suchraum verstanden werden. Ein lernendes Programm (z.B. ein Agent) hat dann gelernt, wenn es seine Wissensbasis und damit sein Verhalten so modifiziert hat, das es die ihm gestellte Aufgabe besser erfüllt als zuvor. Die Künstliche Intelligenz unterscheidet in drei Hauptformen der Wissensrepräsentationen:

- symbolisch-kognitivistisch
- subsymbolisch - konnektionistisch
- interaktionistisch

Prof. Dr. Dilger von der TU Chemniz definiert die verschiedenen Wissenspräsentation in seinem Vorlesungsskript [7] wie folgt:

2.2.1 Symbolisch-kognitivistische Repräsentation

Die symbolisch-kognitivistische Position lässt sich in drei Grundsätzen zusammenfassen:

- Die Repräsentationen sind unabhängig vom zu Grunde liegenden physikalischen Träger.
- Mentale Zustände sind repräsentational oder intentional, d.h. sie beziehen sich auf etwas für die Agenten Externes.

- Repräsentationen bestehen aus Symbolen, die in Computer-basierten Sätzen oder irgendwelchen anderen Strukturen angeordnet sind. Schlussfolgern bedeutet das Manipulieren dieser Strukturen, um neue Strukturen aus ihnen zu erhalten.

Die symbolischen Expertensysteme speichern ihr Wissen z.B. in einer Menge von Regeln (Regelbasis). Die Regeln werden durch aussagenlogische Formeln (Prädikate) beschrieben. Jedes der Prädikate enthält eine Implikationsregel. Die Variablen des Prädikats repräsentieren dabei eine Menge von Fakten, die Aussagen über Zustände der Umwelt repräsentieren. Durch die Anwendung von Regeln auf eine gegebene Faktenbasis und deren Verkettung entsteht wiederum neues Wissen (die Faktenbasis wird manipuliert). Das Expertensystem lernt.

2.2.2 Konnektionistische Repräsentation

Nach der konnektionistischen Position ist Wissen in einem Netz wie einem zellulären Automaten in Form numerischer Werte verteilt, die den Verbindungen zwischen den Zellen zugeordnet sind. Schlussfolgern bedeutet Propagieren numerischer Werte in diesem Netz und Veränderung der Werte an den Verbindungen. Die Wissensrepräsentation nach der konnektionistischen Position heißt auch subsymbolisch, im Kontrast zur symbolischen Repräsentation. Obwohl das menschliche Hirn mit einem ähnlichen Verfahren sein Wissen organisiert, ist subsymbolisches Wissen für den Menschen nur schwer zugänglich.

2.2.3 Interaktionistische Repräsentation

Die interaktionistische Wissensrepräsentation sieht ein Pool von Agenten vor, bei denen jeder ein bestimmtes Konzept oder eine Informationseinheit repräsentiert. Das Verhalten des Agentensystem resultiert aus der Interaktion, in der die verschiedenen Agenten stehen. Da hier von einem Multiagenten System ausgegangen wird, sei diese Form der Repräsentation vernachlässigt. Für nähere Informationen sei hier auf die Diplomarbeit von Alexander Oreschenko [4] verwiesen.

2.3 Lernverfahren

2.3.1 Konnektionistisches Lernen

Das konnektionistische Lernen ist eng mit den subsymbolischen Neuronalen Netzen verknüpft. Das Programm sammelt empirische Daten über Zusammenhänge in seiner Umwelt, um auf deren Basis später Entscheidungen treffen zu können:

Eine Reihe von Eingabemustern, denen ein gewünschtes Ausgabemuster zugeordnet ist, werden dem Verfahren vorgegeben. Das System generiert auf jedes Eingabemuster ein Ausgabemuster und vergleicht es mit dem vorgegebenen Ausgabemuster. Entsprechend der Differenz zwischen Soll- und Ist- Muster modifiziert das Verfahren die mit den Kanten des Neuronalen Netzes assoziierten Gewichtungswerte, um die Qualität des Ausgabemusters zu verbessern.

Nachteil dieser Technik ist die subsymbolische Repräsentation des gelernten Wissens, welches wie oben erwähnt ein Nachvollziehen des Entscheidungsweges schwer oder unmöglich macht. Da an einen Agenten aber durchaus die Forderung nach Transparenz seiner Entscheidungswege gestellt werden kann, scheint der konnektionistische Ansatz einen Nachteil zu haben. Die Transparenz des Entscheidungsvorgangs kann in der künftigen Entwicklung der Agenten zum vertrauensbildenden Element in der Mensch - Agenten Kommunikation werden und ist unter diesem Gesichtspunkt von entscheidender Natur.

2.3.2 Genetische Programmierung

Das Verfahren der genetischen Programmierung arbeitet wie die Lernverfahren der Neuronalen Netze auf der Basis empirischer Analysen. Die genetischen Verfahren sind jedoch in der Lage, mit jeder auf Symbole abbildbaren Wissensbasis zu arbeiten.[1] Werden genetische Verfahren auf symbolisch repräsentiertes Wissen angewendet, hat dies den Vorteil, dass die erlernten Verhaltensregeln durch logische Sätze repräsentiert werden, die von einem Menschen verstanden werden können.

Der Ansatz der genetischen Verfahren ist es, verschiedene Symbolketten (Genome) zu generieren und diese durch einen Interpreter in ein Folge von Aktionen umzusetzen (Verhalten/Phänotyp). Die Aktionsfolgen werden hinsichtlich einer bestimmten Aufgabenstellung bewertet (Fitness) und unter Berücksichtigung

[1] Es ist also durchaus denkbar ein genetisches Verfahren auf ein Neuronales Netz anzuwenden, da es trotz seiner konnektionistischen Wissensrepräsentation auf eine endliche Menge von Symbolen abgebildet werden kann.

ihrer Qualität zu neuen überführt (Kreuzung/Mutation/Direkt Migration). Dabei werden eher die qualitativ hochwertigen Symbolketten zur Überführung gewählt, als die mit einer niedrigen Qualität (Selektion). Dieser Vorgang wird solange wiederholt, bis ein bestimmter Grad an Qualität erreicht oder eine bestimmte Anzahl von Überführungsvorgängen durchgeführt wurde.

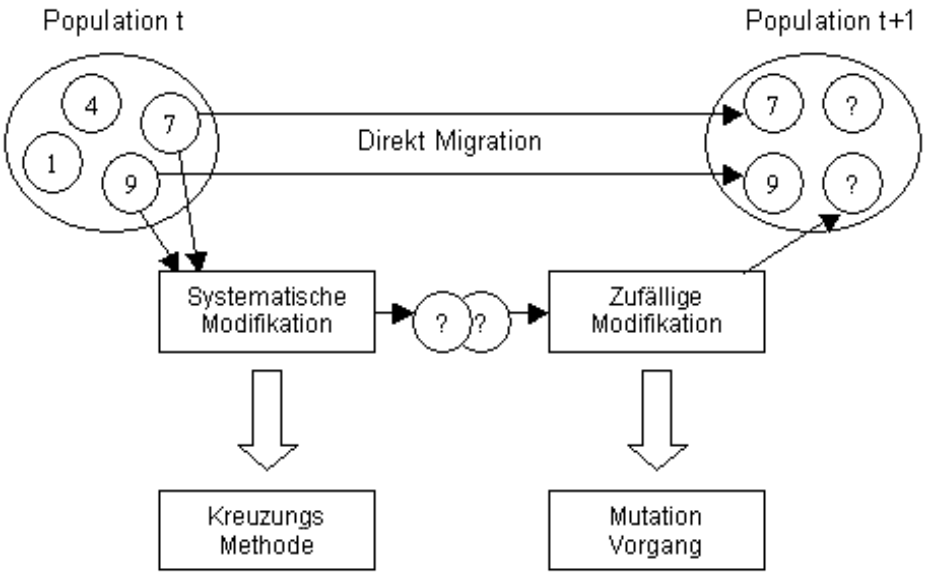


Abbildung 2.1: Die Populationsüberführung und ihre Teilprozesse

Die Strukturierung des Lernvorgangs ist stark am darwinistischen Evolutionsmodell orientiert. Zur Beschreibung der verwendeten Datenstrukturen und der dynamischen Prozesse des Verfahrens werden oft Metaphern aus der Genetik verwendet. Die folgende Übersicht zeigt die wichtigsten Begriffe der biologischen Evolution in Relation zu ihren Entsprechungen in der Informatik:

Biologie	Informatik
Problem der bestmöglichen Lebensbewältigung	formalisiertes, zu lösendes Problem
Grad der Überlebensfähigkeit eines Individuums	problemspezifische, zu optimierende Fitness-Funktion
Individuum (Phänotyp / Genotyp)	Lösungsvorschlag / dessen abstrakte Formulierung
Population	Pool von Lösungsvorschlägen
Mutation	zufällige Änderungen an Individuen
Kreuzung	Kombination des Informationsgehaltes von Individuen
Selektion	Verwerfen "schlechter" Lösungsvorschläge

Evolution in Biologie und Informatik

Die Modellbildung

Wie eingangs beschrieben wird der Lernvorgang durch wiederholtes Selektieren und Überführen von Populationen implementiert. Die Ur - Population wird dabei durch zufällig generierte Genome gebildet, die die ersten Lösungsansätze des genetischen Optimierers bilden. Der Prozess endet genau dann, wenn ein definiertes Abbruchkriterium zutrifft oder eine festgelegte Anzahl von Populationen generiert wurde.

Wollen wir eine Lernaufgabe mit Hilfe eines genetischen Verfahrens lösen, so gilt es, diese zu Formalisieren und auf den genetischen Optimierer in Form einer entsprechenden Konfiguration abzubilden. Dieser Vorgang entspricht der Abbildung problemspezifischer Aspekte der Realität auf ein formales Modell. Die folgende Übersicht beinhaltet die wichtigsten Konfigurationsschritte und Parameter, die bei diesem Prozess eine Rolle spielen:

- Ein **Genom** wird definiert. Die Sprache des Interpreters wird auf die Variablen des Genomdatentyps abgebildet. Dies beinhaltet neben der Definition der vom Genom gehaltenen Variablen auch einen Mechanismus für die zufällige Generierung von Programmen, die für die Erstellung einer Urpopulation von Nöten sind.

- Eine **Fitness Funktion** wird definiert. Um eine qualitätsorientierte Selektion durchzuführen wird jedes Genom durch die sogenannte Fitness Funktion bewertet. Sie interpretiert den Inhalt des Genoms und bewertet dessen Phänotyp hinsichtlich der Problemstellung. Die Fitnessfunktion muss einen

vergleichbaren, numerisch repräsentierten Wert zurückliefern. Sie muss die Bewertung eines Genoms vollständig, richtig und allgemeingültig beschreiben, um den Lernerfolg zu sichern, da sie die eigentliche Formalisierung der Lernaufgabe repräsentiert.

- Die **Kreuzungsmethode** definiert den Transformationsvorgang, der die erfolgreichsten Genome einer Population zu einer neuen Population überführt. Sie kreuzt den Informationsgehalt zweier Genome und generiert aus diesen neue Individuen, deren strukturelle Richtigkeit dabei nicht verletzt wird. Sie stellt den systematischen Teil der Transformation bei der Überführung der Populationen dar. Aus algorithmischer Sicht repräsentiert sie mit dem Selektionsverfahren das gerichtete Fortschreiten im Suchraum auf der Suche nach der optimalen Lösung. Sie setzt in Verbindung mit dem Selektionsprozess die durch die Fitnessfunktion generierten Informationen in eine heuristische Suche um.

- Die **Mutationsmethode** definiert die Methode, die auf die durch Kreuzung entstandenen "Kinder" zweier Genome angewendet wird. Die Mutationsmethode verändert zufällig ausgewählte Informationseinheiten des Genoms, ohne dessen strukturelle Richtigkeit zu verletzen. Die Mutation stellt eine Risikostreuung des Evolutionsprozesses da, der verhindern soll, dass sich der Suchprozess in einem lokalen Qualitätsmaximum "verrennt" (vergleiche Hill Climbing Algorithmen). Gerade bei nicht linearen Problemen, bei denen die strukturelle Ähnlichkeit keine unmittelbare Korrelation zur Qualität darstellt, ist diese zufällige Modifikation ein wichtiger Prozess.

- Die **Populationgröße** wird festgelegt. Sie legt die Größe der Urpopulation fest und wieviele Genome nach dem Selektionsvorgang neu berechnet werden sollen. Je grösser die Population, desto grösser ist das Spektrum der durch Mutation und Kreuzung generierten Genome. Ebenfalls über die Populationsgrösse konfigurierbar ist der Selektionsdruck , dem die Individuen unterliegen. Ist die Grösse der Folgepopulation kleiner als die der Ausgangspopulation, so haben weniger Individuen die Möglichkeit ihr Genominformationen in die nächste Generation einzubringen. Die "Konkurrenz" steigt und das "Überleben" wird schwieriger. Das Verfahren wird stärker gerichtet. Es setzt sich eher nur noch die scheinbare Elite durch, was wiederum einer Verstärkung der Hill Climbing Eigenschaft entspricht.

- Eine **Selektionsstrategie** wird definiert. Sie determiniert, welche Genome einer Population in die nächste Generation überführt werden. Die Selektionsstrategie ist eine Stellgröße, mit der wichtige Eigenschaften des Suchprozesses gesteuert werden können. Wird z.B. das beste Individuum ohne

Transformation direkt in die nächste Generation übernommen, so wird die Qualität des besten Individuums monoton steigen. Die Konfigurationsmöglichkeiten dieses Parameters werden im Bezug auf die Aufgabenstellung eingehend im Kapitel 4.2 auf Seite 41 beschrieben.

- Ein **Abbruchkriterium** wird definiert. Das Abbruchkriterium definiert eine Bedingung für den Abbruch des Optimierungungsvorgang terminiert. Der Abbruch wird durch das Eintreten eines bestimmtem Ereignisses festgelegt. Dieses Ereignis kann durch den Qualitätsgrad des besten Individuums oder durch eine festgelegte Anzahl von Generationen, die vom Optimierer generiert werden sollen, definiert.

Genetische Verfahren und Agenten

Die genetischen Verfahren bieten mit der Fitnessfunktion eine komfortable Schnittstelle zum Agentenparadigma. So entspricht die Bewertung des Genoms durch die Fitnessfunktion der Überprüfung des Verhaltens eines Agenten hinsichtlich seines Ziels. Wie könnte eine solche Verfahrenssymbiose aussehen? Stellen wir uns das folgende Szenario vor:

Eine Roboterpopulation wird sich auf das Ausweichen von Erdhügeln einstellen, wenn alle Roboter die diese Erdhügel nicht beachteten, nicht in die nächste Generation eingehen würden. In einem Minenfeld würden sicherlich sehr schnell nur noch die Programme die Population dominieren, die in der Lage wären, den Minen auszuweichen. Vorraussetzung wäre jedoch, das genug Roboter vorhanden wären, die es der Roboterpopulation ermöglichte sich auf diese Umgebung einzustellen oder die Roboter in der Lage wären, sich auf Hardware-Ebene zu reproduzieren.

Das Minenfeld ist sicher ein extremes Beispiel. Es zeigt jedoch auf anschauliche Weise, dass die natürliche Selektion, im Zusammenhang mit der biologischen Reproduktion, ein verlässlicher Prozess zur Optimierung von Verhalten ist. Ersetzen wir nun die Minen durch ein im Agenten implementiertes Verfahren, welches feststellt ob der Agent zu nah an einen Erdhügel gefahren ist und dieses Muster entsprechend aussortiert, würden die Roboter auch ohne den terminalen Effekt der Minen lernen diesen auszuweichen.

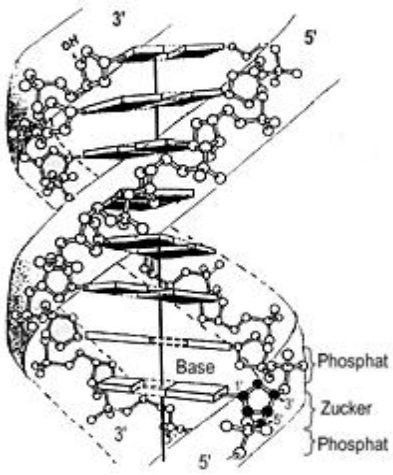


Abbildung 2.2: Genominformationen in einem DNA-Strang

Genetische Verfahren als Heuristische Suche

Die genetischen Verfahren gehören zu der Klasse der heuristischen Suchverfahren. Sie nutzen Wissen über die Problemstellung, um den Prozess der Problemlösung zu beschleunigen. Bei den genetischen Verfahren ist dieses Wissen durch die den Genomen zugeordnete Qualität gegeben.

Der Ansatz der genetischen Verfahren ist es, den Suchraum zu durchqueren, in dem bevorzugt die Lösungsansätze untersucht werden, die den als gut befundenen Ansätzen ähneln. Es wird also bevorzugt die ”Nachbarschaft” der erfolgreichen Genome untersucht.

Gehen wir bei den durch das Genom repräsentierten Merkmalen von quantitativen Attributen aus, kann die vektorielle Distanz der durch die Merkmale aufgespannten Vektoren als Ähnlichkeitsmetrik verwendet werden. Dabei wird das gerichtete Durchqueren durch eine Zufallskomponente aufgeweicht. Das Verfahren wählt zufällig Genome in den Bereichen, die es aufgrund seiner heuristischen Informationen als erfolgsversprechend klassifiziert.

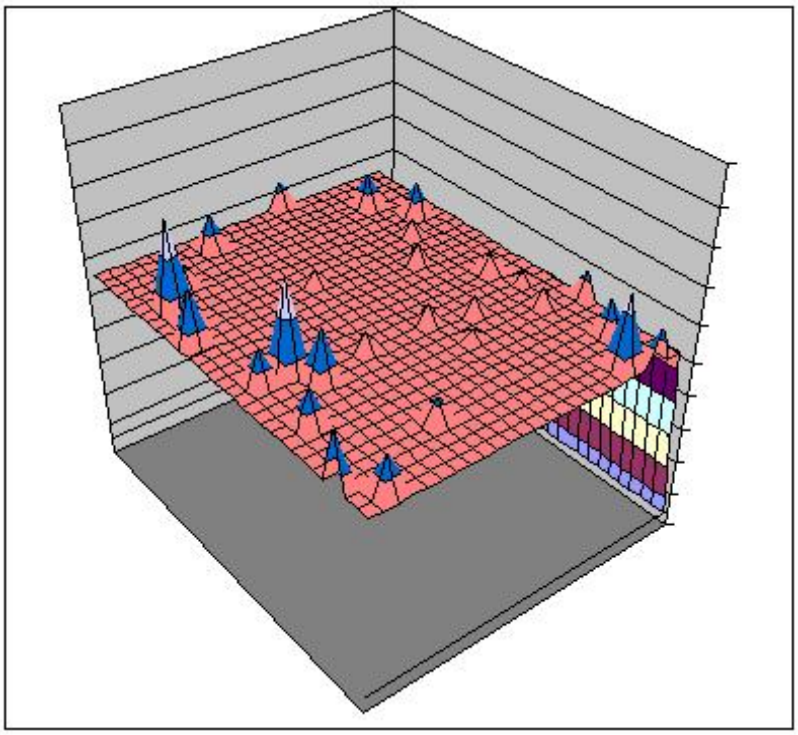


Abbildung 2.3: Die Urpopulation streut die Lösungsansätze

Die Abbildung 2.3 zeigt die durch ein genetisches Verfahren generierte Urpopulation von Genomen mit zwei Merkmalsausprägungen.

Die Y-Achse repräsentiert dabei die Qualität , X- und Z-Achse die Ausprägung eines quantitativen Merkmals. Bei der gebildeten Urpopulation ist eine relativ gleichmäßige Streuung der Lösungsansätze zu beobachten.

Das Verfahren versucht durch diesen Prozess Bereiche zu finden, in denen das Auffinden von erfolgreichen Genomen wahrscheinlich ist. Es generiert dazu Genome, die eine zufällige Semantik besitzen, jedoch strukturell konsistent sind.

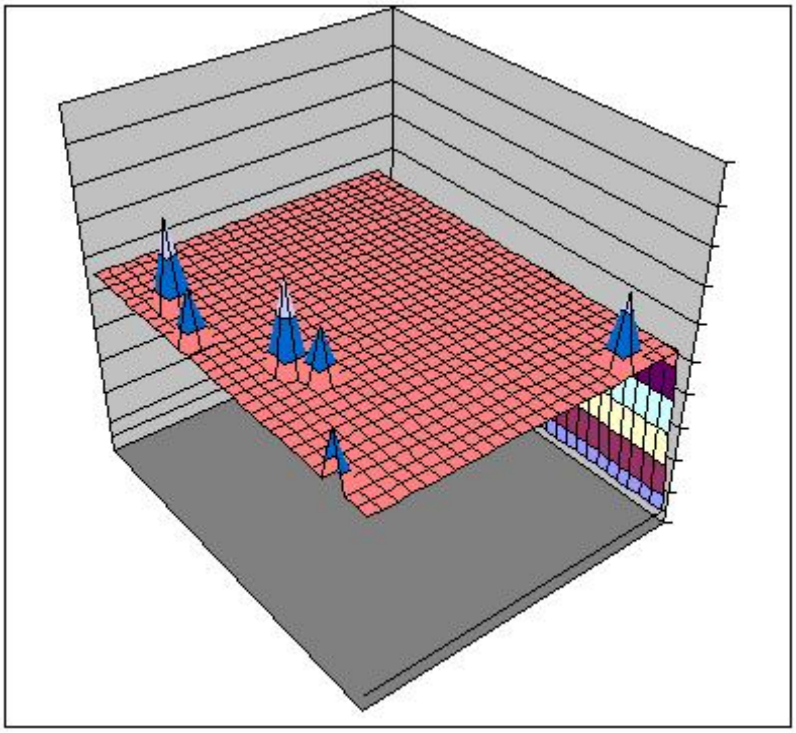


Abbildung 2.4: Das Selektionsverfahren verwirft die ”Verlierer”

Das Selektionsverfahren verwirft die schlechten Lösungsansätze der Urpopulation und selektiert so die Genome, die die Basis für die Berechnung der als nächstes zu untersuchenden Lösungsansätze darstellen.

Es zeigt sich, dass gute Lösungen verstreut im Raum liegen können, und so die strukturelle Ähnlichkeit der Genome nicht zwangsläufig auf eine qualitative Relation hindeuten muss. Die direkte Nachbarschaft eines guten Genoms muss, abhängig von der Problemstellung und ihrer Formalisierung, nicht notwendigerweise qualitativ hochwertige Genome enthalten.

Die Anwendung genetischer Verfahren zeigt jedoch, dass das bevorzugte Durchsuchen dieser Bereiche bei bestimmten Problemstellungen eine erfolgreiche Strategie darstellt. [2]

[2] Es bestätigt sich die alte Volksweisheit ”Der Apfel fällt nicht weit vom Stamm”

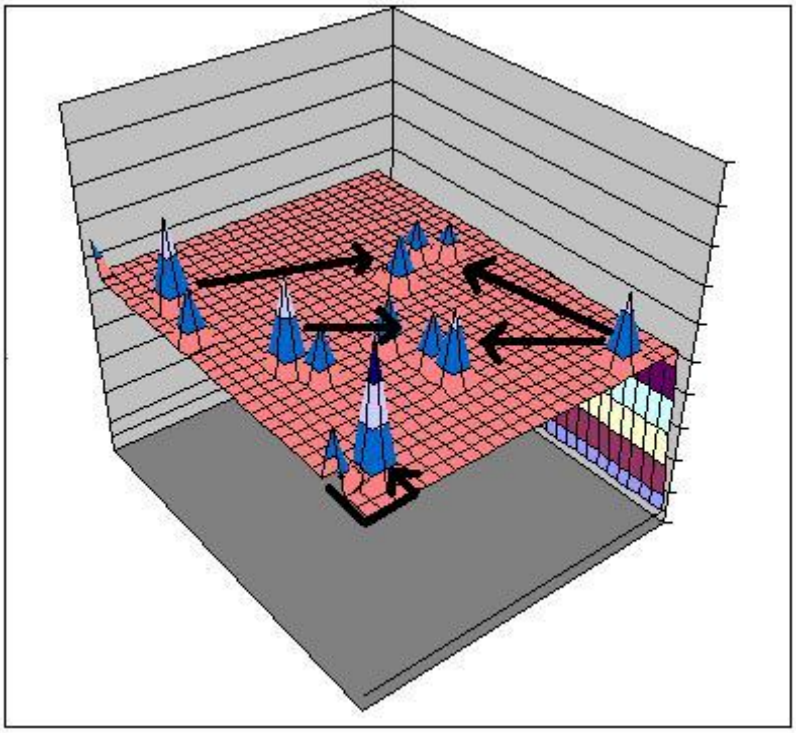


Abbildung 2.5: Die selektierten Genome bilden die neue Population

Das Verfahren gewinnt nach der Selektion durch Kreuzung und Mutation der selektierten Genome neue Lösungsansätze. Die Pfeile setzen in der Abbildung die Eltern mit ihren Kindern in Relation. Die Transformation erfolgt dabei zunächst durch eine Kreuzung. Die neu generierten Genome werden anschließend mutiert. Es kann dabei passieren, dass ein Genom mit sich selbst gekreuzt wird und so keine Modifikation der Genominformationen stattfindet.

Es ist zu beobachten, dass auch vermeintlich schlechte Genome gute Kandidaten in ihrer Nachbarschaft besitzen können. Das lokale Maxima im vorderen linken Bereich des Bildes stellt einen aus einem mittelmäßigen Genom erzeugten Lösungsansatz dar. In diesem Fall wurde die Modifikation durch eine Mutation bewirkt, die trotz ihrer geringen strukturellen Modifikation erheblichen Einfluss auf die Qualität des Genoms hatte.

Kapitel 3

Die Problemstellung

Verhaltensbasierte Agenten und maschinelle Lernverfahren sind erprobte Verfahren die ihre Anwendungsbereiche in der Wirtschaft und Wissenschaft gefunden haben [10]. Wie aber verhält es sich mit Systemen, die durch ihre Nähe zur Umwelt eine starke Echtzeitkomponente enthalten und die nicht in einer vom Programmierer definierten Umwelt eingebettet sind (siehe Kapitel 1.2 auf Seite 12)? Der verhaltensbasierte R.A. Brooks (MIT) formuliert es besonders prägnant:

> This suggests that problem solving behavior, language, expert knowledge and application, and reason, are all pretty simple once the essence of being and reacting are available. That essence is the ability to move around in a dynamic environment, sensing the surroundings to a degree sufficient to achieve the necessary maintenance of life and reproduction. This part of intelligence is where evolution has concentrated its time - it is much harder [9]

Brooks argumentiert weiter, dass Intelligenz in der Natur nie ohne einen Körper auftritt und existieren kann und so die Einbettung des Körpers in die Natur grundlegend ist. Diese Einbettung geschieht durch aufeinander aufbauende sensormotorische Rückkopplungsschleifen. Will man versuchen, ein künstlich intelligentes System zu konstruieren, so geht dies nur mit Hilfe und auf Basis von Robotern.

Der Autor grenzt in seinen Aussagen Intelligente Systeme durch zwei Eigenschaftsausprägungen ab. Die Situiertheit und die Verkörperung. Eine dynamische Umwelt erfordert ein Entscheidungsverfahren, welches situationsabhängige Entscheidungen trifft. Ein Schweißroboter, dessen Intelligenz auf eine Stapelverarbeitungsanwendung zurückgeführt werden kann, ist laut Brooks also eigentlich kein intelligentes System. Die geforderte Einbettung der Intelligenz in die Umwelt

wiederum erfordert einen Körper, der in den meisten Fällen durch einen Roboter repräsentiert wird. Auch unter diesem Aspekt würden laut Brooks viele Agenten als nicht intelligent gelten.

	Keine Verkörperung	Verkörperung
Keine Situiertheit	Batchprogramm	Schweißroboter
Situiertheit	Softwareagent	Mobiler Roboter

Beispiele zu Brooks Klassifikation

Brooks Abgrenzung ist stark an der Intelligenz biologischer Lebewesen orientiert, was eine durchaus berechtigter Standpunkt ist. Künstliche Intelligenz muss an der biologischen Intelligenz gemessen werden, da die biologische Intelligenz das unmittelbare Vorbild und der Ursprung aller durch Computer simulierter Intelligenz ist.

Wollen wir also einen intelligenten Agenten konstruieren, so müssen wir zusätzlich zur Situiertheit und Lernfähigkeit seine Verkörperung sicherstellen. Wie aber lassen sich die maschinellen Lernverfahren auf die Robotik übertragen? Die Anzahl der Selbstinspektionsvorgänge, die zur Erstellung einer qualitativ hochwertigen Wissensbasis nötig sind, können abhängig von der Problemstellung, Sensorik und Aktorik des Agenten bis in die Unberechenbarkeit gehen. Gerade bei Robotern ist die Interaktion mit der Umwelt zur Ausführung der Selbstinspektion jedoch zeitaufwändig. Es gilt zu prüfen, ob und wie maschinelle Lernverfahren angewendet werden können, um einen Roboter lernen zu lassen. Zu diesem Zweck soll folgender Versuch durchgeführt und dokumentiert werden:

> Es soll ein Agent entwickelt werden, der einen physikalischen Körper in Form eines Roboters besitzt. Der Agent verfügt über einen genetischen Optimierer, der das Verhalten des Agenten hinsichtlich einer exemplarischen Aufgabenstellung optimieren soll. Die Aufgabe des Roboters ist es, so zu manövrieren, dass er einer sich auf dem Boden befindlichen Linie folgt.

Kapitel 4

Das Design

Der grundlegende Ansatz des Versuchs ist es, ein genetisches Verfahren zu implementieren, welches Verhaltensmuster in der Form von Regelsätzen erstellt und diese zu einem Roboter sendet. Der Roboter besitzt eine virtuelle Maschine, die die Regeln in sensormotorische Rückkopplungen umsetzt, um die Qualität der Regelbasis zu bewerten. Als Sensor stehen ihm dabei entsprechend der Problemstellung ein Lichtsensor sowie ein Kontaktsensor zur Verfügung. Der genetische Optimierer führt auf Basis dieser verteilten Fitnessfunktion eine Optimierung durch um ein Verhalten zu generieren, welches die im Kapitel 3 auf Seite 28 definierte Aufgabe erfüllt.

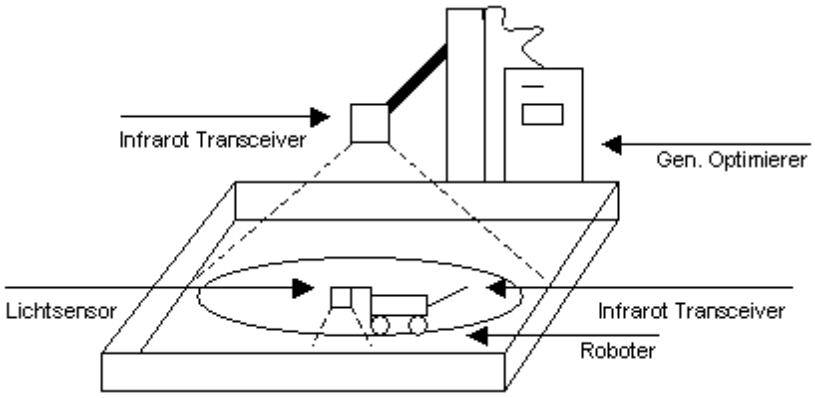


Abbildung 4.1: Der Versuchsaufbau

Die Hauptkomponenten des Systems sind der genetische Optimierer (die Evolutionsmaschine) und eine virtuelle Maschine, die die Genome (Regelsätze) interpretiert und so das eigentliche Verhalten des Agenten erzeugt.

Die auf dem Prozessor des Roboters laufende virtuelle Maschine kommuniziert auf Basis von Infrarotnachrichten mit dem auf einem externen PC laufenden genetischen Optimierer. Die Architektur entspricht einer Komponentenarchitektur, bei der die Schnittstellen der Komponenten durch ein Infrarotnachrichten basiertes ”Message Sending” bedienbar sind.

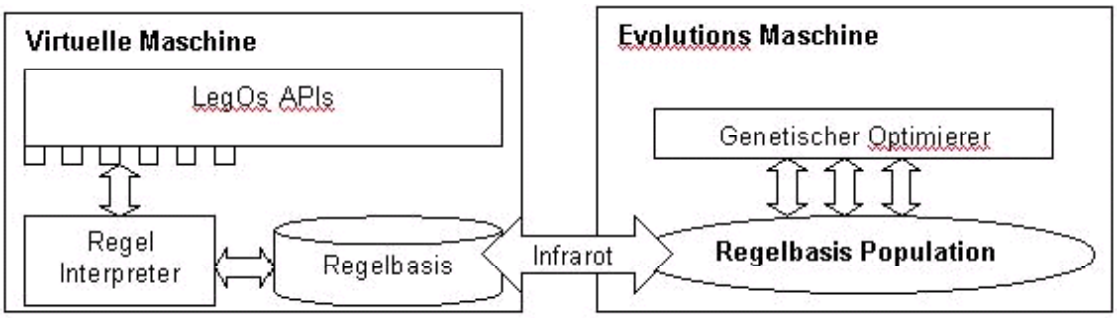


Abbildung 4.2: Grobkonzept des Agentensystem

4.1 Die virtuelle Maschine

Die virtuelle Maschine ist aus der Sicht des genetischen Verfahrens ein Teil der Fitnessfunktion. Sie implementiert dabei die eigentliche Qualitätsbewertung und die Abbildung dieser auf einen numerischen Wert. (siehe auch Kapitel 2.3.2 auf Seite 19).

Aus Sicht des Agenten stellt die virtuelle Maschine die verhaltensbildende Komponente dar, die die durch Genome repräsentierten Regelbasen in Verhalten umsetzt. Sie besteht aus einer Vorgangssteuerung, einer Regelbasis und dem eigentlichem Interpreter, der die in der Wissensbasis gespeicherte Regelbasis in Funktionsaufrufe an die Roboterbasis umsetzt. Zur Kommunikation mit der Evolutionsmaschine enthält die virtuelle Maschine weiter ein plattformspezifisches Kommunikationsmodul, welches für den Versand von für den Lernprozess relevanten Daten genutzt wird.

Abbildung 4.3 stellt vereinfacht den Vorgang der Qualitätsbewertung innerhalb der virtuelle Maschine da. Ein Regelsatz (Verhaltensmuster) wird über das Kommunikationsmodul empfangen und in die Wissensbasis des Roboters eingespeist. Der Interpreter verarbeitet die einzelnen Regeln und generiert im Zusammenspiel mit den von der Roboterbasis angebotenen API ein Verhalten. Während

dieses Prozesses sammelt der Interpreter Daten über das jeweilige Verhalten, die durch das im Interpreter enthaltene Fitness Objekt auf einen numerischen Wert abgebildet werden. Der berechnete Wert wird am Ende der Interpretation über das Kommunikationsmodul an das Lernverfahren zurück gesendet.

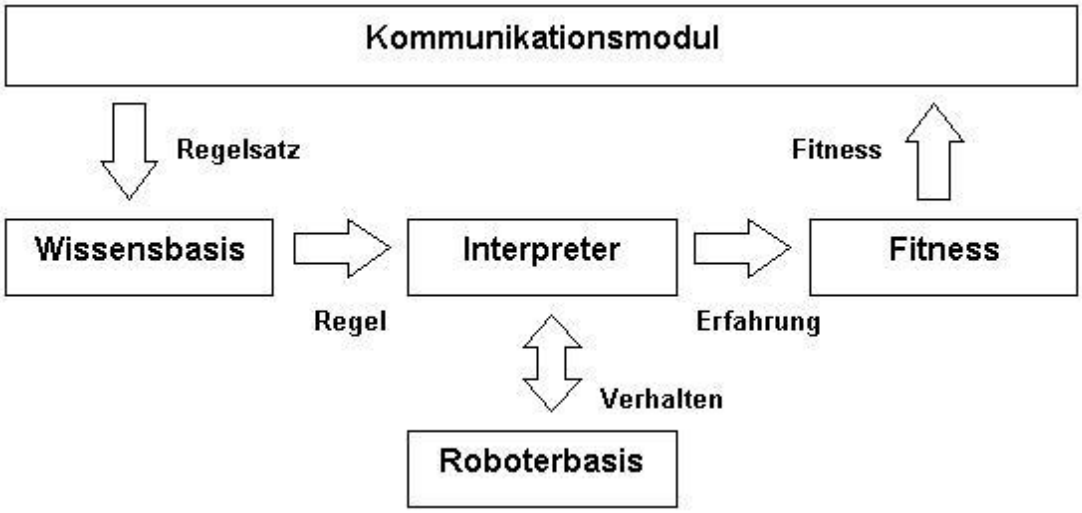


Abbildung 4.3: Informationsfluss in der virtuellen Maschine

4.1.1 Die Regelbasis

Die Regelbasis repräsentiert die vom Agenten verwendete Wissensbasis, die sein Verhalten determiniert. Dieses vom Interpreter ausgewertete Programm wird durch eine variable, geordnete Menge von Implikationsregeln repräsentiert, bei denen die Prämissen Aussagen über Sensordaten repräsentieren und das im Konklusionsteil stehende Symbol eine Aktion darstellt. Jede Regel besteht aus 2 Prämissensymbolen, einem logischen Operator, der die Wahrheitswerte der beiden Prämissen verknüpft, und einem Aktionssymbol.

Für S gleich der Menge der möglichen Prämissen und H gleich der Menge der Aktionen repräsentiert eine Regel je nach ihrem Verknüpfungsoperator eines der folgenden Prädikate:

$$
\begin{aligned}
p_1 \vee p_2 &\Rightarrow h \\
p_1 \wedge p_2 &\Rightarrow h \\
p_1 \wedge \neg p_2 &\Rightarrow h \\
p_1, p_2 &\in S \\
h &\in H
\end{aligned}
$$

Die logischen Operatoren

$$
\begin{aligned}
!(nand) \\
\&(and) \\
|(or)
\end{aligned}
$$

determinieren, welches der drei Prädikate auf die Symbole einer Regel angewendet wird. In einigen Versuchsdurchläufen stellt der ! Operator statt der *nand* Semantik ein ”Fallback” - Verhalten dar. Die Prämisse einer Regel, die das ! Symbol als logischen Operator enthält, evaluiert in diesem Fall unabhängig von den Prämissensymbolen der Regeln immer zu wahr.

Es gilt zu Untersuchen, in wie fern ein solches ”Fallback” - Verhalten den problemspezifischen Adaptionsvorgang im Agentensystem beschleunigen kann. Welche Semantik das ! Symbol in der entsprechenden Versuchsdurchführung beschreibt, ist der jeweiligen Versuchsdokumentation zu entnehmen.

Die Prämissen und Aktionen der Regeln repräsentieren die Schnittstelle des Agenten zur Umwelt. Die durch die Sensorik des Roboters aufgenommenen Informationen werden bei der Auswertung einer Prämisse interpretiert und auf einen Wahrheitswert abgebildet. Entsprechend stellen die Aktionen Zustände der Motoren des Roboters da. Für den Versuch wurden folgende Prämissen und Aktionen definiert:

Prämisse	Sensorzustand
EYE	Der Lichtsensor befindet sich auf schwarzem Untergrund
NOTEYE	Der Lichtsensor befindet sich auf weißem Untergrund
BUMP	Der Roboter ist gegen eine Arealbegrenzung gefahren.

Die Prämissen repräsentieren Sensorzustände

Aktion	Motorzustand
FWD	Der Roboter bewegt sich vorwärts
BCKWD	Der Roboter bewegt sich rückwärts
TURNLEFT	Der Roboter dreht sich gegen den Uhrzeigersinn
TURNRIGHT	Der Roboter dreht sich mit dem Uhrzeigersinn
LEFTFWD	Der Roboter fährt in einer Linkskurve vorwärts
RIGHTFWD	Der Roboter fährt in einer Rechtskurve vorwärts
BCKWDAFTERBUMP	Der Roboter weicht einer Arealbegrenzung aus

Aktionen repräsentieren Motorzustände

Für den Austausch von Programmen zwischen der virtuellen Maschine und dem Lernverfahren werden die Regeln auf eine Zeichenkette abgebildet. Die einzelnen Regeln werden dabei durch das Symbol ”+” voneinander getrennt und durch das vierstellige Tupel *p, p, h* und dem logischem Operator beschrieben. Die Regeln bilden in dieser Form die Wörter der durch folgenden Ausdruck definierten Sprache, wobei *p* ein Terminalsymbol ist, das eine Prämisse repräsentiert und *h* ein Symbol aus der Menge der möglichen Symbole für eine Aktion ist:

$$[+[||\&|!]pph]^{+}$$

In dieser Form werden die Prämissen und Aktionssymbole in einer für die Übertragung optimierten Repräsentation dargestellt:

Prämisse	Reduzierte Darstellung
EYE	E
NOTEYE	N
BUMP	U

Die Prämissen werden auf ein ASCII Zeichen abgebildet

Aktion	Reduzierte Darstellung
FWD	F
BCKWD	B
TURNLEFT	L
TURNRIGHT	R
LEFTFWD	‹
RIGHTFWD	›
BCKWDAFTERBUMP,	A

Die Aktionen werden auf ein ASCII Zeichen abgebildet

Beispiele zur Regelrepräsentation

Die Regel ”Wenn sich der Roboter über der schwarzen Linie befindet dann fahre geradeaus” würde folglich so beschrieben:

Interne Repräsentation	Reduzierte Repräsentation
+ & EYE EYE FWD	+ & EEF

Die Regel ”Wenn sich der Roboter nicht über der schwarzen Linie befindet dann fahre nach links” würde so beschrieben:

Interne Repräsentation	Reduzierte Repräsentation
+ & NOTEYE NOTEYE TURNLEFT	+ & NNL

4.1.2 Der Regelinterpreter

Der Interpreter repräsentiert mit der von ihm interpretierten Regelbasis das Verhalten des Roboters. [1] Der Regelinterpreter "parst" die Regelbasis und bedient das Fitness Objekt, welches er mit Informationen über Sensorzustände speist. Das Fitnessobjekt kapselt die Berechnung des numerischen Qualitätswertes, wohingegen der Interpreter der folgenden Funktionalitäten realisiert :

- **Neue Regelbasis setzen:** Setzt das als Parameter übergebene Regelwerk in die Wissensbasis (Regelbasis) des Interpreters ein.
- **Zur Ausgangsstellung zurückkehren:** Setzt eine Regelbasis, die den Roboter auf die schwarze Linie führt, als aktuelle Wissensbasis und interpretiert diese solange, bis der Roboter sich über schwarzem Untergrund befindet. Dieser Vorgang dient zur Sicherstellung der Vergleichbarkeit der berechneten Qualitätswerte (siehe Kapitel 4.1.3).
- **Regelbasis bewerten:** Führt den Interpretationsvorgang durch, bis der Roboter gegen eine Arealbegrenzung fährt oder eine definierte Zeitspanne verstrichen ist.

Die Regelinterpretation erfolgt durch eine serielle Abarbeitung einzelner Regeln. Evaluiert der Prämissenteil einer Regel zu wahr, so wird die im Aktionsteil spezifizierte Aktion vom Agenten ausgeführt, und die Regelinterpretation startet je nach verwendeter Verhaltensarchitektur vom Beginn der Regelbasis.

Wird eine "Subsumption" Architektur [11] verwendet, so beginnt die Interpretation erneut vom Beginn der Basis. Ist dies nicht der Fall, so werden auch die hinter der angewendeten Regel folgenden Regeln der Basis interpretiert, was zu einem Inteferieren verschiedener Aktionen (Regeln) führen kann und so das Gewicht der einzelnen Regeln bei der Verhaltensbildung reduziert. Die Regelbasis muss bei dieser Architektur mehr als ein Ganzes gesehen werden, bei dem das Gesamtverhalten nicht immer auf das Verhalten einzelner Regeln zurückgeführt werden kann.

Welche der beiden Architekturen die für die Problemstellung angemessenere ist, soll im durchgeführten Versuch ermittelt werden. Das Wechseln zwischen den beiden Architekturansätzen soll durch wenige einfache Modifikationen am Interpreter Objekt realisierbar sein.

[1] Man könnte unter einem anderen Gesichtspunkt auch sagen, dass der Roboter von einem Softwareagenten gesteuert wird, der über die API des Betriebsystems die Umwelt des Roboters manipuliert oder sich in ihr fortbewegt.

Die Evaluierung der Prämissen einer Regel erfolgt durch den Aufruf von Methoden, die den Zustand eines bestimmten Sensors auf einen Wahrheitswert abbilden und diesen zurückliefern. Jedem Prämissensymbol ist durch den Interpreter genau eine Methode zugeordnet, die einen der oben genannten Sensorzustände repräsentiert. Entsprechend führt eine implizierte Aktion zu einem Methodenaufruf, der die Motoren des Roboters ansteuert. Die eingestellte Motorkonfiguration bleibt solange gültig, bis eine andere Aktion ausgeführt wird. Der Agent entscheidet sich also für ein bestimmtes Verhalten und überprüft dann in kurzen Abständen, ob dieses Verhalten beibehalten werden soll oder ob ein anderes Verhalten in seiner Situation sinnvoller ist.

Die Dauer, die dem Interpreter für das Interpretieren und Bewerten einer Regelbasis zur Verfügung steht, ist ein Konfigurationsparameter des genetischen Verfahrens, der direkten Einfluss auf die Richtigkeit der Fitnessfunktion hat. Die Zeitspanne muss es dem Bewertungsverfahren ermöglichen, eine valide Aussage über die Qualität des generierten Verhaltens zu treffen.

Da ein Verhalten, welches den Roboter gegen eine der Arealbegrenzung manövriert, kein vielversprechender Kandidat auf der Suche nach einem Linefollow Verhalten ist, sollen diese Regelbasen von aussortiert werden. Der Interpretationsvorgang bricht beim Kontakt des Roboters mit einer Arealbegrenzung ab. Die Regelbasis wird in diesem Fall als unanwendbar klassifiziert und der Interpretationsvorgang terminiert.

4.1.3 Die Fitness

Die richtige Formalisierung der Verhaltensbewertung ist ein kritischer Prozess beim Design eines lernenden Systems. Ihre Richtigkeit entscheidet über den problemspezifischen Lernerfolg. Bei einer fehlerhaften Verhaltensbewertung kann es passieren, dass zwar erfolgreich gelernt wird, das Lernziel jedoch verfehlt wird, da die Lernaufgabe nicht korrekt formalisiert wurde. Die Berechnung der Qualität einer Regelbasis wird auf Seite der virtuellen Maschine durch das Objekt Fitness realisiert.

Das Fitness Objekt wird vom Regelinterpreter mit Informationen über gültige Prämissen versorgt und implementiert die Abbildung dieser Erfahrungswerte auf einen numerischen Qualitätswert. Es bietet Methoden zum Einspeisen der Erfahrungswerten an, die durch einen fest verdrahteten Berechnungsalgorithmus auf den Qualitätswert abgebildet werden. Die Erfahrungswerte werden dabei in Form von Wahrheitswerten übergeben, bei denen jedem Wahrheitswert eine der im Kapitel 4.1.1 definierten Aussagen über Sensorzustände (Prämissen) zugeordnet ist.

Der Algorithmus zur Berechnung des Fitnesswertes verwendet die absolute Häufigkeit des Eintretens der zur Qualitätsberechnung definierten Prämissen. Das Fitnessobjekt besitzt für jeden Sensorzustand einen Zähler, der die Häufigkeit des Eintretens der ihnen zugeordneten Prämissen darstellt. Die numerische Qualität berechnet sich aus einer zu spezifizierenden Formel, die die verschiedenen Zähler in Relation setzt.

Der Vorteil dieser Vorgehensweise ist die ”anytime” Eigenschaft des Bewertungsverfahrens. Da die Qualität durch Häufigkeiten gezählt wird, ist jederzeit ein Ergebnis verfügbar, obwohl die statistischen Randbedingungen für empirische Analysen und mögliche Besonderheiten der Aufgabenstellung beachtet werden müssen.

Für den durchgeführten Versuch wurden die beiden Sensorzustände EYE und NOTEYE als für die Qualitätsberechnung relevante Erfahrungswerte festgelegt. Nach der Interpretation jeder Regel einer Regelbasis wird eine Stichprobe genommen. Der Qualitätswert wird direkt durch den Quotienten der absoluten Häufigkeit des Eintretens der Prämisse EYE zur Häufigkeit des Sensorzustandes NOTEYE definiert. Zur Sicherstellung der Kompatibilität mit anderen Optimierungsverfahren wird das Ergebnis der Berechnung mit 100 multipliziert und abgerundet.

$$Fitness(R) = h(EYE)/h(NOTEYE) * 100)$$

Die Zähler starten mit einer Default-Belegung von 145 (EYE) zu 100 (NOTEYE). Der Grund hierfür ist die Tatsache, dass ein sich nicht bewegendes Muster qualitativ höher bewertet werden soll als ein Verhalten, welches gegen eine Wand fährt. Ein Verhalten, welches den Roboter gegen eine Wand fahren lässt, wird mit einer Fitness von 0 bewertet. Da der Quotient $145/100$ größer 0 ist, kann der genetische Optimierer eine qualitätsorientierte Unterscheidung zwischen diesen beiden Verhaltenstypen realisieren .

Das formale Ziel des genetischen Optimierers ist es, ein Genom zu erstellen, welches den Roboter so lange wie möglich auf schwarzem Untergrund hält. Auf den ersten Blick ist diese Formulierung kein ausreichendes Qualitätsmerkmal für das ”Line Follow” Problem. Unter Einbeziehung der dem Agenten zur Verfügung stehenden Aktionen scheint diese Funktion jedoch ausreichend. Da es keine Aktion gibt, die es dem Roboter ermöglicht, stehen zu bleiben, ist die Formel implizit

durch die Bedingung ”während der Roboter sich bewegt” erweitert. Was passiert jedoch, wenn die Regelbasis nur Regeln enthält die aufgrund ihrer Semantik nicht angewendet werden können ? Es gilt zu untersuchen, ob der beschriebene Mechanismus zu einer richtigen und allgemeingültigen Fitnessbewertung führt, oder ob die Fitnessfunktion durch die Hinzunahme eines Bewegungssensors (Shaft Encoder) erweitert werden muss.

Um die Vergleichbarkeit der Fitnesswerte zu gewährleisten, muss die Fitnessfunktion vor der Qualitätsbewertung die Vergleichbarkeit der Ausgangsposition des Roboters sichern. Dazu soll der Roboter vor dem Interpretationsvorgang auf die schwarze Linie manövriert werden und dort anhalten. Es gilt zu Untersuchen ob dieser Mechanismus die Vergleichbarkeit der Qualitätsbewertung sichern kann und damit die Fitnessfunktion eine allgemeingültige Grundlage für den Optimierungsprozess darstellt.

4.1.4 Die Vorgangssteuerung

Die Vorgangssteuerung organisiert den Prozess der Qualitätsbewertung in der virtuellen Maschine. Sie verwendet die öffentlichen Methoden der beschriebenen Objekte und implementiert so die dynamischen Prozesse in der virtuellen Maschine. Sie kapselt dazu die folgenden 4 Prozesse durch öffentliche Methoden:

- **Zur Ausgangsstellung zurückkehren:** Lädt eine Regelbasis, die den Roboter auf die schwarze Linie manövriert, und führt diese solange aus bis der Roboter an einer entsprechenden Position steht. Diese Methode wird vor jedem Bewertungsvorgang durchgeführt, um die Vergleichbarkeit der Qualität zu sichern. (siehe Kapitel 4.1.3)

- **Neue Regelbasis empfangen:** Empfängt eine neue Regelbasis von der Evolutionsmaschine mit Hilfe des Kommunikationsmodul und übergibt diese an den Regelinterpreter.

- **Regelbasis bewerten:** Weist den Interpreter an, die gesetzte Regelbasis auszuwerten und das daraus resultierende Verhalten zu bewerten.

- **Qualität der Regelbasis versenden:** Fragt den ermittelten Qualitätswert vom Interpreter ab und versendet ihn mit Hilfe des Kommunikationsmoduls an die Evolutionsmaschine.

4.1.5 Sequenzdiagramm der Selbstinspektion

Die Abbildung 4.4 stellt den zentralen Prozess der Qualitätsbewertung einer Regelbasis in einem Sequenzdiagramm dar. Die von den aufgeführten Methoden verwendeten Hilfsmethoden sind in diesem Diagramm nicht mit aufgeführt, um die Übersichtlichkeit zu gewährleisten.

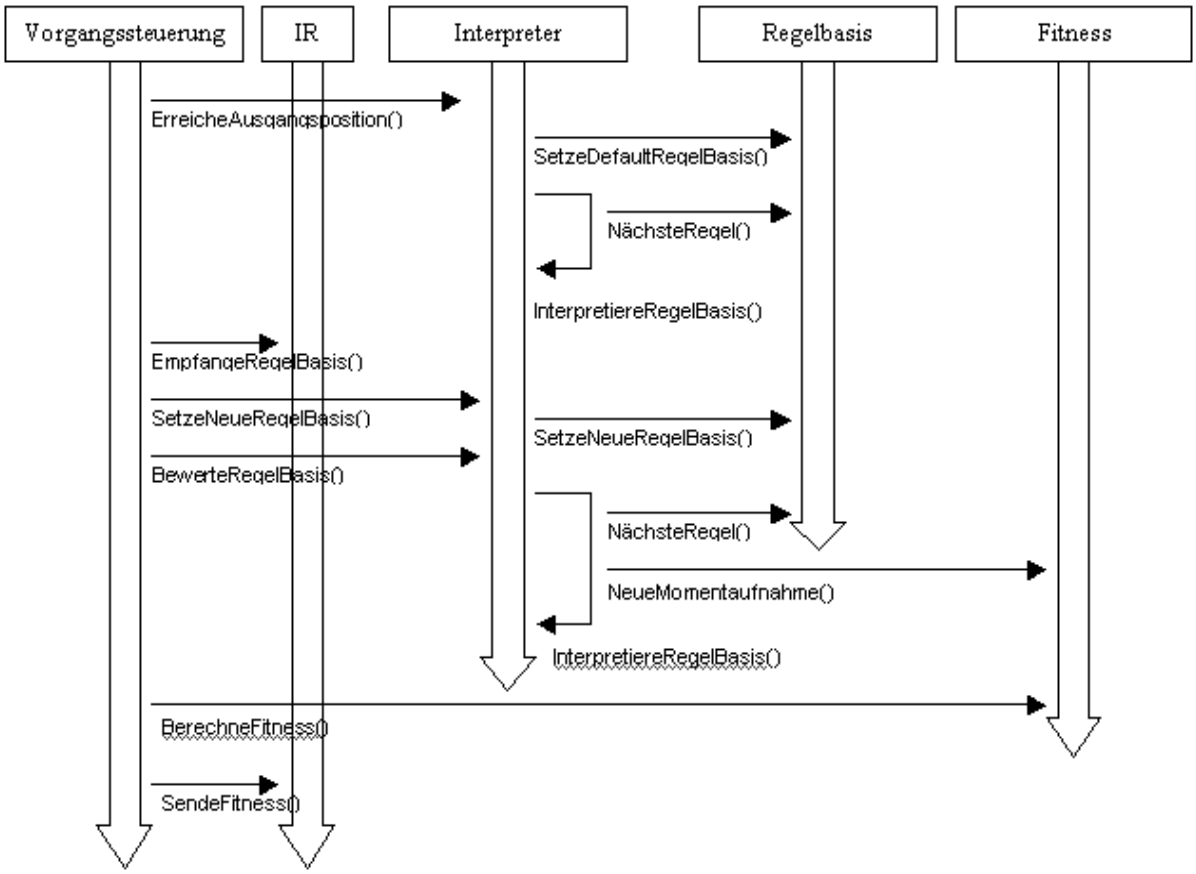


Abbildung 4.4: Sequenzdiagramm der Bewertung einer Regelbasis

4.2 Die Evolutionsmaschine

Die Evolutionsmaschine implementiert den eigentlichen Lernvorgang des Agenten. Sie erstellt und transformiert die Regelbasen, die das Verhalten des Agenten determinieren. Die Evolutionsmaschine besteht aus einem genetischen Optimierer (siehe Kapitel 2.3.2 auf Seite 19) und einem Kommunikationsmodul. Der genetische Optimierer verwendet definierte Verfahren zur Transformation der durch Genomobjekte repräsentierten Regelbasen. Die Fitnessfunktion dieser Genomobjekte versendet die zu bewertende Regelbasis mit Hilfe eines Kommunikationsobjektes zur virtuellen Maschine und wartet auf einen Fitnesswert. Der empfangende Wert stellt den Rückgabewert der Fitnessfunktion und damit die Grundlage für den vom genetischen Optimierer durchgeführten Lernprozess dar.

4.2.1 Das Genom

Eine von der virtuellen Maschine interpretierte Regelbasis wird seitens der Evolutionsmaschine durch einen Baum repräsentiert, der die einzelnen Regeln der Regelbasis hierarchisiert. Der Baum besitzt einen Fan-out von 5, welcher sich au s der verwendeten Interpretersprache ableitet, und eine variable Tiefe. Die Knoten einer Ebene repräsentieren je eine Regel. Die von links nach rechts angeordneten Knoten einer Ebene stellen ein Wort der im Kapitel 4.1.1 definierten Sprache dar. Die Abbildung des Genoms auf eine Zeichenkette ist dabei durch eine Breitentraversierung des Baumes gegeben.

Obwohl dem Trennzeichen ”+” von der virtuellen Maschine keine Semantik zugeordnet ist, ist es in der Repräsentation des Genoms enthalten. Es repräsentiert in diesem Fall eine einfache bedingte Verzweigung, deren Sprungziel bei nicht Eintreten der durch die Prämissen definierten Bedingung der Beginn der nächsten Regel ist.

Diese Struktur erlaubt es, bei einer möglichen Erweiterung der virtuellen Maschine, auch andere Compound Statements (while Schleifen etc.) zu implementieren. Die in reduzierter Darstellung beschriebene Regelbasis $+|EEF+|NNL$ würde durch den in Abbildung 4.5 dargestellten Baum repräsentiert:

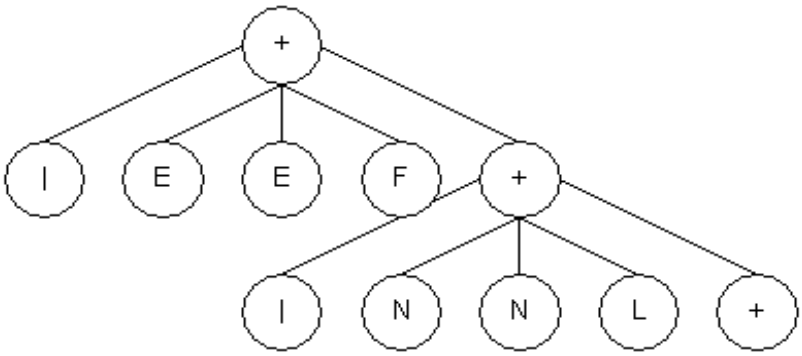


Abbildung 4.5: Genom des Programmes +|EEF+|NNL

Die Qualitätsbewertung eines Genoms durch die Fitnessfunktion, wird weitestgehend durch die virtuelle Maschine implementiert. Dies hat zur Folge, das der im Genom Objekt codierte Teil der Fitnessfunktion ausschließlich für die Abwicklung des Nachrichtenaustauschs mit der virtuellen Maschine zuständig ist.

Neben der Fitnessfunktion ist der Default-Konstruktor des Genom Objekt ein für die Optimierung wichtiger Mechanismus . Er implementiert den für die Erstellung der Ur-Population benötigten Prozess der Generierung von Genomen mit zufälliger Semantik. Die Genome müssen trotz ihrer zufälligen Semantik eine syntaktisch Korrektte Struktur besitzen, d.h. sie müssen von einem Parser, der Sätze der im Kapitel 4.1.1 auf Seite 32 definierten Sprache parst, fehlerfrei geparst werden.

4.2.2 Die Populationsparameter

Die verwendete Populationsgröße muss die von der Fitnessfunktion benötigte Zeitspanne zur Qualitätsbewertung beachten. Die der virtuellen Maschine zur Bewertung eines Genoms zur Verfügung stehende Zeitspanne ist ein Versuchsparameter, der bei einer festgelegten Dauer der Optimierung in direkter Relation zur verwendeten Populationsgröße des genetischen Verfahrens steht.

Bei einer vorgegebenen Dauer der Bewertung eines Individuums, die z.B. durch eine echtzeitbezogene Lernaufgabe festgelegt ist, ist die Anzahl der Generationen in Relation zur Größe der Population eine Stellgröße, die die algorithmische Eigenschaft des Suchverfahrens beeinflusst.

Werden nur wenige aber große Populationen generiert, so wird die Suche weniger stark auf die Qualitätshügel im Suchraum zugehen. Stellen wir die Informationen eines Genoms als Vektor in einem N dimensionalen Raum dar, so ist die

Heuristik des Verfahrens dadurch gegeben, das es die Vektoren am ehesten auf ihre Qualität hin prüft, die in der Nachbarschaft eines erfolgreichen Genomvektors liegen (siehe Kapiel 2.3.2). Vergrößern wir die Populationen und reduzieren entsprechend die Anzahl der Generationen, so werden zwar genau so viele Genome untersucht, es wird jedoch seltener überprüft, ob ein noch besseres Genom erstellt wurde, dessen Nachbarschaft möglicherweise noch bessere Genome enthält. Viele kleine Generationen haben den Vorteil, dass eine deutlich häufigere Kontrolle der Qualität der Genome stattfindet und so Sackgassen oder Qualitätshügel früher erkannt werden können. Andererseits geht der Algorithmus in diesem Fall stärker auf die Qualitätsmaxima zu, was nicht immer von Vorteil sein muss (siehe auch hier Kapitel 2.3.2 auf Seite 19).

Das richtige Gleichgewicht zwischen den beiden Stellgrößen zu finden, ist ein problemspezifischer Prozess, der nur auf Basis eines praktischen Versuchs durchgeführt werden kann. Um eine grobe Einschätzung realistischer Populationsparameter zu erhalten, kann jedoch die folgende Abschätzung durchgeführt werden.

Sei die für den Versuch definierte Dauer des Interpretationsvorgangs $t_{individuum}$ 15 Sekunden. Bei einer Populationsgröße $P_{AnzahlIndividuen}$ von 10 benötigte die Bewertung einer ganzen Population einen Zeitraum von 150 Sekunden. Sei weiter die Anzahl der Generationen $\#P$ in unserem Evolutionsprozess gleich 10, so hätte der Agent ca. eine halbe Stunde Zeit, das gestellte Problem zu lösen, da sich die Dauer der Gesamtevolution $t_{evolution}$ aus der Anzahl der Generationen multipliziert mit der Dauer für die Bewertung einer Population ergibt.

$$\#P = \frac{t_{evolution}}{t_{individuum} * P_{AnzahlIndividuen}}$$

$$t_{evolution} = \#P * t_{individuum} * P_{AnzahlIndividuen}$$

4.2.3 Das Abbruchkriterium

Das Abbruchkriterium definiert das Ende des Optimierungsvorgangs. Die genetischen Verfahren verwenden zwei Arten von Abbruchkriterien. Das Ende der Optimierung wird entweder durch das Erreichen eines bestimmten Qualitätswertes oder durch das Erreichen einer bestimmten Generationsanzahl definiert. Eine qualitätsorientiertes Abbruchkriterium hat den Effekt, dass das Verfahren bei einem festgelegten Qualitätsgrad des besten Indiduums terminiert. Dies hat jedoch

keinen Einfluss auf die Eigenschaften des Suchprozesses. Zusätzlich führt die variable Anzahl der Generationen zu einer nicht vorhersehbaren Dauer des Lernvorgangs, was die Rationalität des Handelns eines verkörperten Agenten beeinflussen kann.

Eine festgelegte Dauer des Lernvorgangs führt zwar nicht in jedem Fall zum Erfolg, stellt aber sicher, dass sich der Agent nicht in einem unlösbaren Problem verrennt und zumindest angenährte Lösungen an einem festgelegten Zeitpunkt liefert. Dies ist bei einem in eine natürliche Umwelt eingebetteten Agenten eine Eigenschaft die über den Erfolg seines Handelns entscheiden kann.

Als Abbruchkriterium sei deshalb das Erreichen einer bestimmten Generation im Optimierungsprozess definiert. Für den ersten Versuch wird der im Kapitel 4.2.2 diskutierte Wert von 10 festgelegt.

4.2.4 Die Transformationverfahren

Die Kreuzungsmethode

Die Kreuzungsmethode definiert den Vorgang der systematischen Modifikation von Genomen. Sie implementiert den ”gerichteten” Teil der Suche nach dem optimalen Individuum und basiert auf den durch die Fitnessfunktion gewonnenen Informationen. Die durch Kreuzung entstandenen Kinder zweier Genome repräsentieren den Versuch des Optimierungsverfahrens, die erfolgreichen Komponenten zweier Genome zu vereinen.

Zur Einschränkung des Suchraums stellt die Integrität der Informationsstruktur der neu generierten Genome eine Randbedingung dar, die es einzuhalten gilt. Für den durchgeführten Versuch bedeutet dies, dass das neu generierte Programm lt. der im Kapitel 4.1.1 beschriebenen Sprache syntaktisch richtig ist. Zur Vereinfachung des Optimierungsvorgangs wurde keine programmkomponentenorientiere Qualitätsbewertung vorgesehen. Die qualitätsorientierte Bevorzugung einzelner Regeln beim Kreuzungsvorgang kann durch die Kreuzungsmethode nicht umgesetzt werden. Die Kreuzungsmethode kreuzt also zufällig die Information der systematisch ausgewählten Genome, ohne dabei Genome zu generieren die vom Interpreter nicht fehlerfrei geparst werden können.

Wie können diese Anforderungen umgesetzt werden ? Auf Grund der im Kapitel 4.2.1 auf Seite 41 spezifizierten Struktur des Genoms, kann eine strukturerhaltene Kreuzung zweier Individuen durch das vertauschen zweier Äste der durch die Genome repräsentierten Bäume erreicht werden. Als Randbedingung gilt hier,

dass nur an den Knoten getauscht wird, die an der selben Position innerhalb einer Regel liegen. Eine geeignete Sollbruchstelle stellt das Trennzeichen ”+” da. Es wird so sichergestellt, das eine erfolgreiche Regel nicht ”zerschnitten” wird. Je ein zufällig gewählter ”+” Knoten der zu kreuzenden Bäume stellt dabei die Bruchstelle der zu vertauschenden Äste dar.

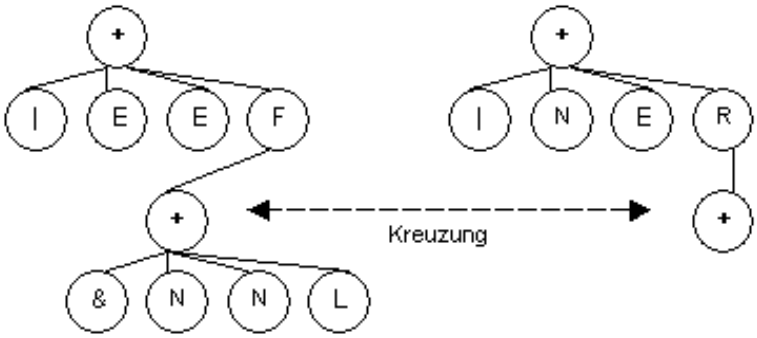


Abbildung 4.6: Unterbaumkreuzung

Die Mutationsmethode

Die Mutationsmethode implementiert die zufällige Modifikation von durch Kreuzung entstandenen Genomen. Sie modifiziert den Informationsgehalt eines Genoms mit einer gegebenen Wahrscheinlichkeit, ohne dessen strukturelle Integrität zu verletzen. Das im Versuch verwendete Verfahren zur Mutation tauscht den Informationsgehalt zweier Knoten in einem Genom, die sich an der selben Position innerhalb der Regeln befinden:

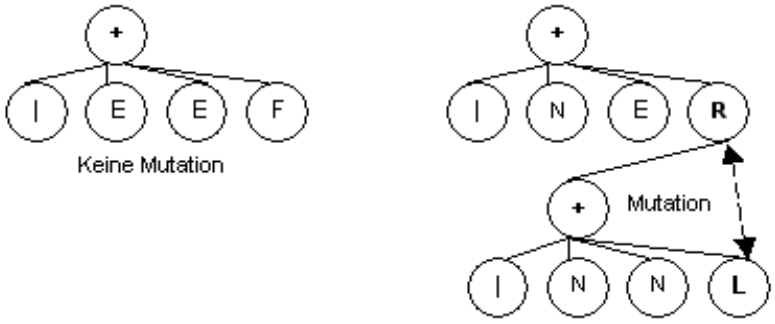


Abbildung 4.7: Swap node Mutation

Die Wahrscheinlichkeit, dass ein solcher Vorgang für den Knoten eines Baumes angestoßen wird, ist eine Stellgröße des Optimierers, den es entsprechend der Aufgabenstellung anzupassen gilt.

4.2.5 Das Selektionsverfahren

Das Selektionsverfahren legt fest, nach welchem Verfahren die Individuen selektiert werden, um zu einer neuen Generation überführt zu werden (siehe Kapitel 4.2.4 auf Seite 44). Aus der Sicht des Suchverfahrens entspricht dieser Vorgang einer priorisierten Auswahl des als nächstes zu bewertenden Knoten im Suchbaum.

Die Eigenschaften des durch das genetische Verfahren implementierte Suchverfahrens, werden zum einen durch die Kreuzungs- und Mutationsmethoden definiert, können aber auch durch das Selektionsverfahren beeinflusst werden. Eine direkte Übernahme des besten Individuums in die nächste Generation führt z.B. zu einer monoton steigenden Qualität des besten Individuums, da es nicht durch ein schlechteres ersetzt werden kann.

Die Methode der ”überlappenden Populationen” ist gerade bei einer begrenzten Anzahl von Generationen vorteilhaft, da sie schnell auf die Qualitätshügel im Suchraum zugeht. Selektionsstrategien, die mit nicht überlappenden Populationen arbeiten, gehen das Risiko ein, bei Abbruch der Evolution ohne eine anwendbare Lösung ”dazustehen”. Andererseits bringen überlappende Populationen die Problematik der ”Hill Climbing” Algorithmen mit sich, da gute Regelbasen schnell über vermeintlich schlechtere Regelbasen, die aber über eine bessere Grundlage zur Weiterentwicklung bieten könnten, dominieren. Gerade bei komplexen Problemstellungen kann dies eine Eigenschaft sein, die die Effektivität des Lernvorgangs stark reduziert.

Aufgrund der fixen Generationenanzahl (siehe Kapitel 4.2.3 auf Seite 43) und der geringen Anzahl an Populationen (siehe Kapitel 4.2.2 auf Seite 42) ist die Verwendung eines mit überlappenden Populationen arbeitenden Selektionsverfahrens notwendig. Die Anzahl der direkt überführten Genome ist eine Stellgröße, die den für die Aufgabenstellung geeigneten Grad an Innovation in der Population definiert. Für das verwendete Selektionsverfahren sei folgende Randbedingung definiert:

Die 5 besten Individuen in einer Population von 10 gehen direkt in die nächste Generation über. Die restlichen Individuen der Folgegeneration werden durch die gewählte Transformationsmethode aus den Genomen der zu selektierenden Population generiert.

Für die Auswahl der zu transformierenden Genome wird ein qualitätsbezogenes Selektionsverfahren angewendet. Es ordnet jedem Genom einen Wahrscheinlichkeitswert zu, der sich direkt aus dem Quotienten der Qualität des entsprechenden Genoms und der Summe der Qualität aller Genome ableitet. Dieser Wert repräsentiert die Wahrscheinlichkeit dafür, das das entsprechende Genom bei einer Kreuzung herangezogen wird. Das gewählte Verfahren lässt es zu, dass auch schlechtere Genome mit in die Generierung einer neuen Population einbezogen werden und so die ”Hill Climbing” Eigenschaft aufgeweicht wird. Die Auswahl der beiden für eine Kreuzung verwendeten Genome erfolgt dabei unabhängig voneinander, was impliziert, dass ein Genom beide Elternteile eines neu generierten Genoms darstellen kann.

Auf eine dynamische Anpassung der Selektionsparameter während der Evolution wurde verzichtet. Eine sich über den Evolutionsprozess verkleinernde Anzahl der direkt übernommenen Genome hätte jedoch den Effekt, dass der genetische Optimierer zu Beginn der Evolution eher viel ”experimentiert”, sich aber zum Ende auf erprobte Strategien verlässt und so stärker auf die Maxima im Suchraum zugeht.

4.3 Das Kommunikationsmodul

Das Kommunikationsmodul implementiert die infrarotbasierte Kommunikation zwischen Evolutionsmaschine und virtueller Maschine. Es setzt die aus dem Design des Gesamtsystems resultierenden Anwendungsfälle in die folgenden Funktionalitäten um:

- **Führe eine Handshake-Aufforderung durch:** Die virtuelle Maschine versucht eine Kommunikationsverbindung zur Evolutionsmaschine aufzubauen. Zu diesem Zweck wird solange eine als Handshake Aufforderung definierte Nachricht versendet, bis eine Bestätigungsnachricht von der Evolutionsmaschine erhalten wurde.

- **Führe Handshake-Bestäigung durch:** Die Evolutionsmaschine wartet auf eine Handshake Nachricht von der virtuellen Maschine und bestätigt ihren Empfang mit Versand einer Bestätigungsnachricht.

- **Versende ein Genom:** Die Evolutionsmaschine versendet ein zu bewertendes Genom an die virtuelle Maschine. Der Übertragungsvorgang endet mit einer Empfangsbestätigung der virtuellen Maschine.

- **Empfange ein Programm:** Die virtuelle Maschine wartet auf eine Nachricht von der Evolutionsmaschine und bestätigt ihren Empfang mit einem Handshake Signal. Bei der empfangenen Nachricht handelt es sich um ein von der Evolutionsmaschine versendetes Programm, welches mit Abschluss der Kommunikation zur Interpretation zur Verfügung steht.

- **Sende Fitness:** Die virtuelle Maschine versendet einen Ganzzahlwert solange, bis eine Bestätigung von der Evolutionsmaschine empfangen wurde.

- **Empfange Fitness:** Die Evolutionsmaschine wartet auf eine Nachricht von der virtuellen Maschine. Der Empfang der Nachricht wird durch den Versand einer Bestätigungsnachricht bestätigt.

- **Behandle Fehler in der Kommunikation:** Fehlerhafte Nachrichten oder Synchronisierungsprobleme werden vom Kommunikationsmodul bemerkt und entsprechend behandelt.

4.3.1 Die Architektur des Kommunikationsmoduls

Die Objekte des Kommunikationsmodul sind sowohl Teil der virtuellen Maschine als auch der Evolutionsmaschine. Trotz der unterschiedlichen Aufgaben, die sie in den beiden Komponenten erledigen, sind generalisierbare Eigenschaften beider

Klassen offensichtlich. Um die Wiederverwendbarkeit und Wartbarkeit der Teilmodule zu sichern, leiten die beiden Kommunikationsobjekte von einer abstrakten Oberklasse ab, die die generalisierbaren Eigenschaften und Methoden beider Objekte in einer abstrakten Oberklasse kapselt. Die genaue Arbeitsweise der Kommunikationsobjekte wird aufgrund der starken Abhängigkeit von den verwendeten Bibliotheken im Kapitel 5 Die Implementierung beschrieben.

Kapitel 5

Die Implementierung

5.1 Der Roboter

5.1.1 Lego Mindstorm RIS

Das LEGO MINDSTORMS Robotics Invention System ist ein in Zusammenarbeit des MIT und der Lego Group entstandenes Set von Komponenten, welches zur Erstellung von mobilen Robotern geeignet ist. Es enthält neben den üblichen Lego Bausteinen, Sensoren und Motoren sowie den programmierbaren Baustein RCX. Dieser Baustein enthält einen Hitachi-8300 Prozessor und 32 KB Hauptspeicher. Er verfügt über eine Infrarotschnittstelle, mit der Daten zwischen verschiedenen RCX Bausteinen oder einem PC ausgetauscht werden können.

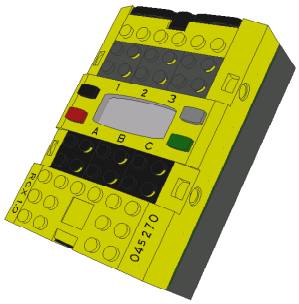


Abbildung 5.1: Der RCX Baustein

Die im Set enthaltenen Sensoren umfassen 2 Kontaktsensoren und einen Lichtsensor. Die Kontaktsensoren verfügen über einen Klickschalter und werden von der Firmware als aktiviert angesehen, wenn dieser eingedrückt ist. Die Lichtsensoren liefern beim Auslesen eine Ganzzahl zurück, aus der unter anderem beschränkte Rückschlüsse über die Farbe des Untergrunds gezogen werden können.

Der RCX Baustein kann mit verschiedenen Programmiersprachen programmiert werden. Für nähere Informationen sei hier auf LugNet [15], das wohl größte Forum, das sich mit der Programmierung des RCX beschäftigt verwiesen.

5.1.2 Die Hardware Konfiguration

Um die im Kapitel 3 beschriebene Aufgabe zu lösen wurde der Roboter wie folgt konfiguriert:

- **Fahrwerk** Der Roboter besitzt zwei mit DC-Motoren gekoppelte Räder zur Navigation. Ein drittes, mitlaufendes Rad ist im hinteren Teil des Roboters angebracht.
- **Kontaktsensor** Ein mit einer Stossstange verbundener Kontaktsensor ist an der Frontseite des Roboters angebracht, um die Kollision mit einer Arealbegrenzung zu verhindern.
- **Lichtsensor** Der Lichtsensor des RIS ist ebenfalls an der Frontseite des Roboters angebracht und tastet den Boden vor sich ab. Er dient zur Erkennung der auf dem Boden des Versuchsareals aufgezeichneten Linie
- **Infrarotspiegel** Da der Infrarotsensor des RCX an der Seite des Bausteins angebracht ist, der IR-Tower des PCs jedoch über dem Versuchsareal angebracht ist, müssen die Signale, die an den PC gesendet oder von ihm Empfangen werden, über einen Spiegel umgeleitet werden, der vor dem Tranceiver des Roboters angebracht ist. werden.

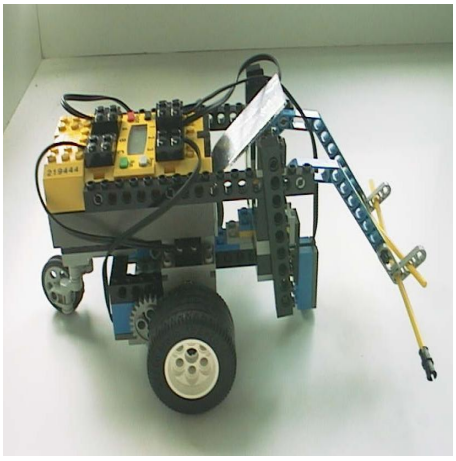


Abbildung 5.2: Die verwendete Roboterkonfiguration

5.2 Das Versuchsareal

Die Umwelt des Agenten wird durch einen 1 x 1.20 Meter großen Holzkasten repräsentiert. Das Areal ist weiß lackiert und wird von 10 cm hohen Wänden umschlossen. Auf dem Boden ist mit schwarzem Klebeband ein Oval gezeichnet, welches die vom Roboter abzufahrende Idealstrecke darstellt. Der IR-Tower des RIS Sets, der von der Evolutionsmaschine zur Kommunikation verwendet wird, wird durch einen 1.2 Meter hohen Galgen gehalten, der an der Arealbegrenzung befestigt ist. Der Tower ist dabei so ausgerichtet, dass er die gesamte Fläche des Areals mit seinen Signalen erreichen kann. Durch ein einfaches Stecksystem kann die Höhe des Infrarottranceivers variiert werden, indem die Halterungslatte in einen anderen Steckplatz geschoben wird.

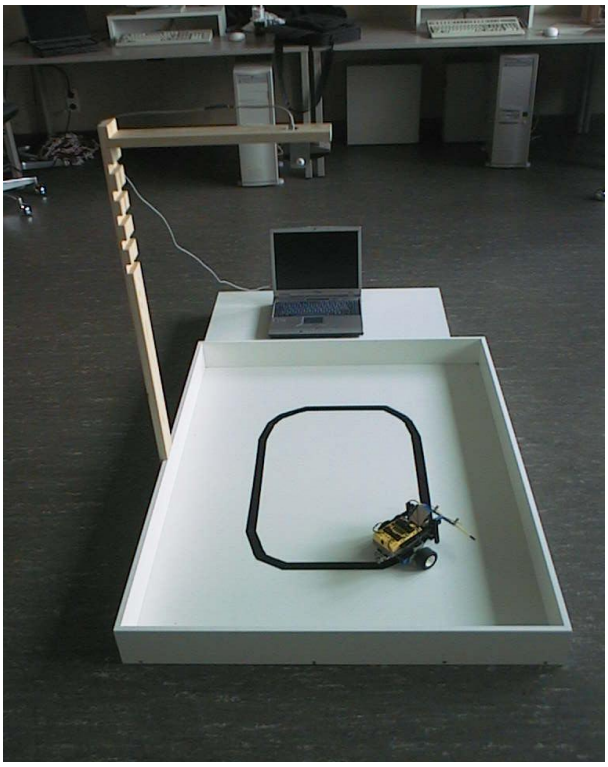


Abbildung 5.3: Der Versuchsaufbau

5.3 Die virtuelle Maschine

Die virtuelle Maschine wird physikalisch vom sich auf dem RCX befindlichen Hitachi Prozessor interpretiert. Ihr primäre Aufgabe ist es, die von der Evolutionsmaschine generierten Verhaltensmuster (Regelbasen) zu interpretieren und ihre Qualität zu bewerten (siehe Kapitel 4 auf Seite 30). Das folgende Unterkapitel beschreibt die Umsetzung der im Design beschriebenen Teilkomponenten. Es beschreibt weiter das zugrundeliegende Betriebsystem und die verwendeten Bibliotheken und Programmiersprachen.

5.3.1 Das Betriebsystem

Der RCX Baustein ist vom Hersteller mit einer Firmware ausgestattet, die den Zugriff auf die Speicher Ressourcen des Bausteins stark eingrenzt. Die C ähnliche Sprache "Not Quit C" erlaubte z.B. zum Zeitpunkt der Versuchsdurchführung eine maximale Variabelenanzahl von 32, was für das geplante System nicht ausreichend war.

Erst mit Ersetzen der Standardfirmware wird diese Begrenzung aufgehoben. Die Entwicklungsumgebung LegOS beinhaltet neben einer neuen Firmware, die den Zugriff auf den gesamten Speicher des Bausteins erlaubt, einen C ++ Cross Compiler sowie eine Bibliothek für den Zugriff auf die Schnittstellen des RCX.

In der Bibliothek ebenfalls enthalten ist das "Lego Network Protokoll" Modul, welches die Kommunikation zwischen RCX Bausteinen und einem PC via Infrarot ermöglicht. Für nähere Informationen sei auf das LNP howto [16] verwiesen. Im Rahmen seiner Studienarbeit an der HAW Hamburg wurde von Oliver Christ ein TCP/IP Stack in den LegOs Kernel integriert, der damit ein standardisiertes und internetfähiges Netzwerkprotokoll anbietet. Da sich die Erweiterung zum Zeitpunkt des dokumentierten Versuchs noch in der Entwicklung befand, konnte sie noch keinen Einfluss auf die Systementwicklung haben. Für detailliertere Informationen bezüglich des LegOS Betriebssystems und seiner Programmierung sei auf das Extreme Mindstorms [12] Buch verwiesen.

5.3.2 Das BehaviourPattern Objekt

Die im Kapitel 4.1.1 definierte Regelbasis wird durch das Objekt **Behaviour Pattern** implementiert. Es dient als Container Objekt und repräsentiert die vom Agenten verwendete Wissensbasis. Das Objekt enthält die von der Evolutionsmaschine generierten Regeln, die das Verhalten des Agenten determinieren. Die Regeln werden dabei in einem Array von Enumerationselementen gespeichert. Die Enumera-

tion **eWord** umfasst alle im Design definierten Symbole für Prämissen, Aktionen und logischen Operatoren.

Um die Struktur des Kommunikationsmoduls einfach zu halten, enthält die Enumeration außerdem die vom Kommunikationsmodul verwendeten Steuerzeichen. Dies ist insofern eine Vereinfachung, als das alle empfangenen Infrarotnachrichten auf den selben Typ abgebildet werden können und so eine einheitliche Verarbeitung und Speicherung realisiert werden kann. Auf die Bedeutung dieser Steuerzeichen wird im Kapitel 5.5 auf Seite 71 genauer eingegangen. Die Enumeration stellt sich unter Verwendung der definierten Symbole wir folgt dar:

enum eWord (IR_HELLO, ... AND, ... ,FWD ...,EYE ...)

Das vom BehaviourPattern Objekt zur Speicherung der Regeln verwendete Array *pattern[50]* ist als Instanzvariable realisiert. Die maximale Länge der im Array enthaltenen Regelbasis wird so auf 50 festgesetzt, was einer Anzahl von 10 Regeln entspricht.

Die Organisation der Daten entspricht einer Stack - Verwaltung, wobei der Zugriff auf die Elemente der Regelbasis durch eine Stream-Schnittstelle realisiert wird. Die interne Organisation des Iterationsprozesses wird durch zwei Zähler realisiert. Zum einen wird der Zähler *nWritePointer* hochgezählt, wenn ein neues Element hinzugefügt wird. Werden die Elemente ausgelesen (wird der Stack abgebaut), so repräsentiert die Variable *nReadPointer* den Index des Elementes im Array, welches zur Zeit das oberste im Stack repräsentiert. Die Iteration enden genau dann (der Stack ist genau dann vollständig abgebaut), wenn die Variable *nReadPointer* gleich der Variable *nWritePointer* ist.

- *isAtEnd()* Stellt fest ob die Iteration über die Regelbasis am Ende ist, und liefert eine 0 zurück wenn weitere Regeln zur Abarbeitung verfügbar sind.
- *reset()* Setzt den Iterationsindex auf 0, so das die Methode *nextPatternPart()* das erste Symbol der ersten in der Regelbasis enthaltenen Regel zurückliefert.
- *nextPatternPart()* Liefert das nächste Symbol der durch den Iterationfortschritt bestimmten Regel zurück. Die Reihenfolge in der die einzelnden Prämissen, Aktionen und logische Verknüpfungen zurückgeliefert werden ist durch die im Kapitel 4.1.1 definierte Sprache bestimmt. Auf Programmebene sind diese Symbole durch die Elemente einer Enumaration repräsentiert. Die Benennung der Elemente entspricht den im Wissensinterpretations Kapitel genannten Symbolen für Prämisssen und Aktionen.

- *void addPatternPart(eWord aPatternPart)* Fügt ein neues Regelsymbol an das Ende der Regelbasis an (”pusht” ein neues Symbol auf den Stack). Für das Anhängen einer vollständigen Regel muss diese Methode insgesamt vier mal aufgerufen werden. Eine Regel muss immer vollständig zugefügt werden. Werden nicht alle Symbole der Regel eingefügt, so entspricht die Regelbasis nicht mehr der im Kapitel 4.1.1 definierten Sprache und ihre Interpretation wird abgebrochen.[1]

- *void asDefaultPattern()* Löscht das vorhandene Regelwerk und fügt eine Regelbasis ein, die bei ihrer Interpretation den Roboter auf schwarzen Untergrund manövriert und dann anhält. Zu diesem Zweck existiert eine fest im Code verdrahtete Regelbasis, die in der Form eines Arrays von **eWord** Enumarationselementen vorliegt: *eWord defaultProgramm[] = { IF , OR , NOTEYE ... } ;*

5.3.3 Das Interpreter Objekt

Das Objekt **Interpreter** interpretiert die im Objekt **BehaviorPattern** gespeicherte Regelbasis (Interpreter Programm). Es generiert im Zusammenspiel mit der Roboterbasis (der Firmware und ihren API's) das Verhalten des Roboters.

Jeder Interpretationsvorgang beginnt mit dem Setzen einer neuen Regelbasis. Die *setBehaviour(...)* Methode nimmt als Parameter ein Objekt vom Typ **BehavioutPattern** und überschreibt die bisherige Regelbasis mit dem übergebenen Objekt. Ist eine neue Regelbasis gesetzt, so wird der Interpretationsvorgang durch den Aufruf der *doIt()* Methode gestartet. Die *doIt()* Methode delegiert die eigentliche Interpretation an die Methode *runForFactUntrue(...)* die den Ablauf der Interpretation organisiert.

Die *runForFactUntrue(...)* Methode nimmt als Parameter ein in der eWord Enumeration enthaltenes Prämissensymbol und bricht die Interpretation genau dann ab, wenn das übergebene Prämissensymbol zu wahr evaluiert oder wenn ein definiertes Zeitintervall verstrichen ist. Für die Bewertung einer Regelbasis ist dieses Symbol durch die Prämisse **BUMP** gegeben, da wie im Kapitel 4.1.3 festgelegt, ein Programm welches den Roboter gegen die Arealbegrenzung manövriert, als nicht anwendbar angesehen wird. Tritt dieser Fall ein, so wird vor dem Abbruch der Interpretation die Methode *fatalFailure()* des Fitnessobjekt aufgerufen, welche eine entsprechende Modifikation des Qualitätswertes implementiert. Anders verhält es sich, wenn das Default Programm interpretiert wird. In diesem

[1] Um dadurch die Effektivität vorangegangener, vollständiger Regeln nicht zu neutralisieren, beginnt die Interpretation vom Beginn der Basis.

Fall ist das Abbruchkriterium durch die Prämisse **EYE** gegeben. Der Interpreter bricht hier nicht nach der definierten Zeitspanne sondern erst mit Erreichen der schwarzen Linie ab.

Die *runForFactUntrue(...)* Methode verwendet das Stream Protokoll des BehaviourPattern Objektes um die einzelnen Symbole der Regelbasis zu interpretieren. Die Symbole werden mit der Methode *getNextPatternPart()* geholt und verarbeitet. Wurde eine Regel interpretiert, prüft der Interpreter durch den Aufruf der *isAtEnd()* Methode des BehaviourPattern Objektes, ob weitere Regeln vorhanden sind oder die Interpretation von vorne beginnt. Nach der Interpretation jeder Regel wird eine Momentaufnahme der Situation durch den Aufruf der *evaluateFitness(..)* Methode des Fitnessobjektes erstellt.

Die Interpretation einer Regel des Programms ist in zwei Phasen eingeteilt: Die Interpretation der Prämisse und die entsprechende Durchführung der Aktion.

Interpretation der Prämisse

Der Wahrheitswert des Prämissenteils wird durch die Methode *isPreconditionValid()* evaluiert. Sie ermittelt den Wahrheitswert der Prämisse und liefert diesen zurück. Prämissen und Aktionen werden dabei wie im Kapitel 4.1.1 auf Seite 32 beschrieben durch festverdrahtete Methodenaufrufe ausgewertet, wobei die den Prämissen zugeordneten Methoden immer einen Wahrheitswert zurückliefern. Die Prämisse **EYE** wird beispielweise durch die folgende Methode ausgewertet:

```
bool isOverDark()
{
        return (LIGHT(SENSOR_2) ›SENSOR_DARK) ;
};
```

Wobei die Funktion *LIGHT(...)* eine durch die Roboterbasis gegebene Funktion zum Auslesen eines Lichtsensors ist, und der Wert **SENSOR_DARK** einen Schwellwert darstellt, der festlegt, welche Sensorenwerte als schwarze Linie interpretiert werden sollen.

Die *isPreconditionValid()* Methode holt sich nacheinander 3 Symbole vom BehaviourPattern Objekt und speichert diese in Variablen. Es handelt sich bei den Symbolen um die beiden Prämissen der Regeln und dem logischen Verknüpfungsoperator. Die Wahrheitswerte der Prämissen werden wie oben beschrieben ermittelt und dann durch die Methode *logicalOperator(...)* , mit dem logischen Operator

verknüpft. Das Ergebniss der Methode stellt den Rückgabewert der *isPreconditionValid()* Methode da.

Durchführung der Aktion

Evaluiert die *isPreconditionValid()* Methode zu wahr, so wird die Aktion der Regel durch die Methode *doAction()* ausgeführt. Kann die Prämisse nicht bewiesen werden, so wird die Aktion durch die Methode *dismissAction()* verworfen.

Die *doAction()* Methode holt sich das nächste Symbol vom **BehaviourPattern** Objekt und übergibt es beim Aufruf der *perform(...)* Methode als Parameter. Die *peform(...)* Methode besteht aus mehreren If Statements, die die jeweiligen Aktionen implementieren:

```
if(action == FWD)
{
      d.debug(”FWD”);
      motor_a_speed(150);
      motor_c_speed(150);
      motor_a_dir(fwd);
      motor_c_dir(fwd);
};
```

Kann die Prämisse nicht bewiesen werden, so wird das Aktionssymbol durch den Aufruf der *dismissAction()* Methode vom **BehaviourPattern** geholt und verworfen. Die interpretation der nächsten Regel kann beginnen.

Ablauf der Regelinterpretation in Pseudocode Notation

Der Prozess der Regelinterpretation kann in Pseudocode Notation wie folgt beschrieben werden:

```
while(! timeHasExpired() || terminatingPremissesIsTrue() )
{
   while(behaviourPattern.isAtEnd())
   {
      behaviourPattern.nextRuleInRuleBase();
      if(isPreconditionValid())
```

```
        {
            doAction();
        }else
        {
            dismissAction();
        }
        fitness.evaluateFitness(evaluatePremisses(EYE));
    }
    behaviourPattern.reset();
};
```

Besonderes Verhalten

Die Komplexität der Aktionen ist zumeist auf den Grad des oben gezeigten Codes beschränkt. Die einzige Ausnahme bildet dabei die Aktion **BCKWDAFTERBUMP** .

Die Aktion **BCKWDAFTERBUMP** ist nicht im Alphabet der Evolutions-Maschine enthalten. Da der Interpretationsvorgang bei der Qualitätsbewertung durch die Aktivierung des Kontaktsensors abgebrochen wird, wird diese Aktion nur dann durchgeführt, wenn das ”Zur Ausgangsposition” Programm verwendet wird (siehe auch Kapitel 4.1.2 auf Seite 36).

Die Aktion repräsentiert das Verhalten des Agenten, wenn der Roboter gegen eine Arealbegrenzung stößt. Um sich von dieser Begrenzung zu lösen, führt der Agent eines von drei ”fest verdrahteten” Bewegungsmustern aus. Diese drei Varianten sind notwendig, um auch bei ungünstigen Situationen wie Ecken oder einem sehr flachen Auftrittswinkel auf die Begrenzung nicht in Deadlock Situationen zu geraten. Die drei Verhaltensmuster werden im Ringpuffer-Verfahren angewendet und setzen den sich an der Wand befindlichen Roboter auf verschiedene Arten zurück:

- In einer geraden Linie zurück
- In einer Rückwärtslinkskurve oder
- In einer Rückwärtsrechtskurve

Dieses Verhalten erwies sich im Versuch als ausreichend, um Deadlock-Situationen zu verhindern, muss jedoch bei einer Erweiterung des Versuchsaufbaus auf seine Anwendbarkeit überprüft werden.

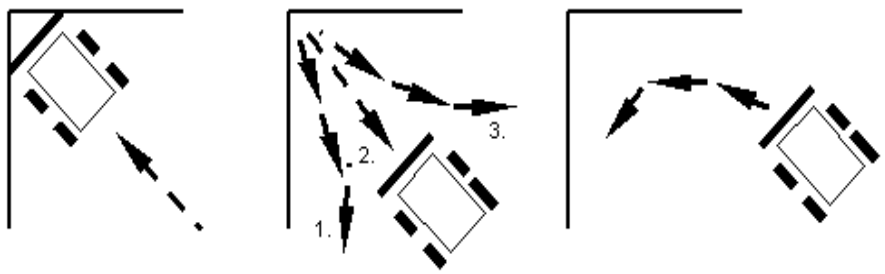


Abbildung 5.4: Ausnahmebehandlung nach Kollision

5.3.4 Das Fitness Objekt

Die im Kapitel 4.1.3 beschriebene Berechnung des Qualitätswertes einer Regelbasis wird durch das Objekt Fitness implementiert. Es wird vom Regelinterpreter mit Informationen über die Wahrnehmung des Roboters versorgt. Eine Momentaufnahme wird durch den Aufruf der Methode *evaluateFitness(..)* realisiert. Sie nimmt den Wahrheitswert der Prämisse **EYE** zum Zeitpunkt der Momentaufnahme als Parameter und erhöht entsprechend dem Wahrheitswert eine der beiden durch Instanzvariablen realisierten Zähler *nTimeSpendOverLine* oder *nTimeSpendOffLine* .

```
void evaluateFitness(bool isOverDark)
{
   if(isOverDark) { timeSpendOverLine() }
   else  timeSpendOffLine()
}
```

Die Abbildung der beiden Zähler auf den numerischen Qualitätswert wird durch die Methode *asInteger()* realisiert. Die Methode dividiert wie im Kapitel 4.1.3 beschrieben den *nTimeSpendOverLine* durch den *nTimeSpendOffLine* Zähler und multipliziert das Ergebnis mit 100. Das Ergebnis wird durch einen Cast auf den Typ **int** abgerundet und zurückgegeben.

Verhält sich der Roboter unakzeptierbar falsch, so wird die Fitness durch den Aufruf der *fatalFailure()* Methode auf 0 gesetzt. Dies ist bei der gewählten Versuchskonfiguration der Fall, wenn der Roboter gegen die Arealbegrenzung stößt. Die Entscheidung, wann ein Verhalten als unanwendbar gilt, ist dabei fest im Interpreter Objekt verdrahtet.

5.3.5 Das FitnessDeterminationProcess Objekt

Das **FitnessDeterminationProcess** Objekt implementiert die Vorgangssteuerung der virtuellen Maschine. Es wird in der *main(...)* Funktion der Applikation instanziiert und verwendet. Die Vorgangssteuerung kapselt die 4 Hauptprozesse des Bewertungsvorgangs als öffentliche Methoden und organisiert den Informationsfluss in der virtuellen Maschine (siehe auch Kapitel 4 auf Seite 30). Sie versendet und empfängt mit Hilfe des Kommunikationsmoduls für den Lernprozess relevante Daten und stößt die Interpretation der zu bewertenden Regelbasis durch den

Interpreter an. Die 4 Phasen der Verhaltensbewertung werden durch die folgenden Methoden implementiert:

- **bool setToDefaultPosition():** Jeder Interpretationsvorgang startet an einer vergleichbaren Stelle des Versuchsareals. Soll der Roboter zur Ausgangsposition zurückkehren, so wird zunächst die Methode *asDefaultPattern()* eines neu instanziierten BehaviourPattern Objekts aufgerufen. Sie kopiert die Elemente des *defaultProgramm[..]* Arrays in die Wissenbasis (siehe Kapitel 5.3.2 auf Seite 5.3.2). Das BehaviourPattern Objekt wird anschliessend durch den Aufruf der *setBehaviourPattern(...)* Methode in den Interpreter "eingesetzt". Der Anstoss der Interpretation erfolgt durch den Aufruf der *runForFactUntrue(..)* Methode des **Interpreter** Objekts, wobei als Parameter die Prämisse EYE übergeben wird.

- **receiveBehaviourPattern():** Das Objekt bezieht eine neue Regelbasis von der Evolutionsmaschine und übergibt sie an den Interpreter. Zur Sicherstellung der Kommunikationsbereitschaft des Kommunikationspartners wird vor dem eigentlichen Vorgang des Empfangs die Methode *establishConnectionForPatternReceive()* des FitnessDeterminationProcess Objektes aufgerufen. Sie ruft die Methode *doHandshake()* des **RCXCommunicator** Objekts auf und wartet, bis diese erfolgreich beendet wird.

  Terminiert die Methode erfolgreich, so wird der eigentliche Empfang der neuen Regelbasis durch die Methode *receivePattern(...)* angestossen. Die *receivePattern(...)* Methode empfängt ein neues Programm durch den Aufruf der *receiveMessage()* Methode des **RCXCommunicator** Objektes. Die Elemente der empfangenen Zeichenkette werden mit Hilfe der *charToInt(...)* Methode des **Debugger** Objektes in ein **eWord** Enumerationselement konvertiert und mit der *addPatternPart(...)* Methode in ein das vom Interpreter verwendete **BehaviourPattern** Objekt eingepflegt. (siehe auch Kapitel 5.3.2 auf Seite 30).

- **determineFitness():** Die Vorgangssteuerung weist den Interpreter an, die empfangene Wissensbasis auszuwerten und ihre Qualität zu bewerten, indem sie die Methode *doIt()* des **Interpreter** Objektes aufruft.

- **sendFitnessValue():** Die durch die *determineFitness()* Methode ermittelte Qualität wird mit Hilfe des Kommunikationsmoduls an die Evolutionsmaschine gesendet. Wie auch beim Empfangen der Regelbasis wird vor dem eigentlichen Übertragen des Fitnesswertes ein Handshake durch die *establishConnectionForFitnessValueTransmission()* Methode durchgeführt. Ansch-

liessend wird der berechnete Qualitätswert vom **Interpreter** Objekt durch die *asInteger()* abgefragt und mit Hilfe der *sendFitnessPattern()* Methode an die Evolutionsmaschine versendet.

5.4 Die Evolutionsmaschine

Der genetische Optimierer und seine Konfigurationsparameter stellen einen der Hauptuntersuchungsgegenstände dieser Arbeit dar. Die Evolutionsmaschine führt, wie im Kapitel 4 beschrieben, die qualitätsorientierte Manipulation der vom Agenten verwendeten Wissensbasis durch, um so sein Verhalten hinsichtlich der gegebenen Aufgabenstellung zu verbessern.

5.4.1 Die GAlib

Da der Untersuchungsgegenstand dieser Arbeit nicht die Implementierung sondern die Anwendung eines genetischen Verfahrens ist, wurde auf die Implementierung eines eigenen Optimierers verzichtet. Statt dessen sollte ein bereits erprobtes, fehlerfreies Verfahren verwendet werden, welches die im Design besprochenen Randbedingungen des Optimierungsvorgangs als konfigurierbare Parameter anbietet. Nach eingehender Sichtung möglicher Kandidaten wurde aufgrund ihrer konkurrenzlosen Mächtigkeit die in C++ implementierte GAlib verwendet.

Die am MIT entwickelte **GAlib** ist ein auf genetischen Verfahren basierendes Framework für Optimierungsprobleme. Sie stellt verschiedene, in Objekten gekapselten Optimierungsstrategien und den dabei verwendeten Datenstrukturen zur Verfügung und ist in C / C++ / Objective C implementiert. Die Bibliothek bietet ein breites Spektrum von vordefinierten Datentypen und zeichnet sich besonders durch eine saubere, objektorientierte Programmstruktur aus:

> GAlib contains a set of C++ genetic algorithm objects. The library includes tools for using genetic algorithms to do optimization in any C++ program using any representation and genetic operators.[14]

Die verschiedenen Datenstrukturen und Strategien liegen in Klassen (siehe Abbildung 5.5) vor, und können in einer entsprechenden Systemimplementierung einfach untereinander ausgetauscht oder erweitert werden. Der Pool von vordefinierten Datenstrukturen und Strategien enthält dabei Methoden und Klassen für:

- Verschiedene Optimierungsstrategien
  (GAGeneticAlgorithm Objekte)

- Verschiedene Qualitätsberechnungsmethoden
  (GAScalingScheme Objekte)
- Verschiedene Selektionsverfahren
  (GASelectionScheme Objekte)
- Verschiedene Repräsentationsformen für Lösungsansätze
  (GAGenome Objekte)
- Verschiedene Kreuzungs- und Mutationsstrategien
  (Methoden der GAGenom Objekte)

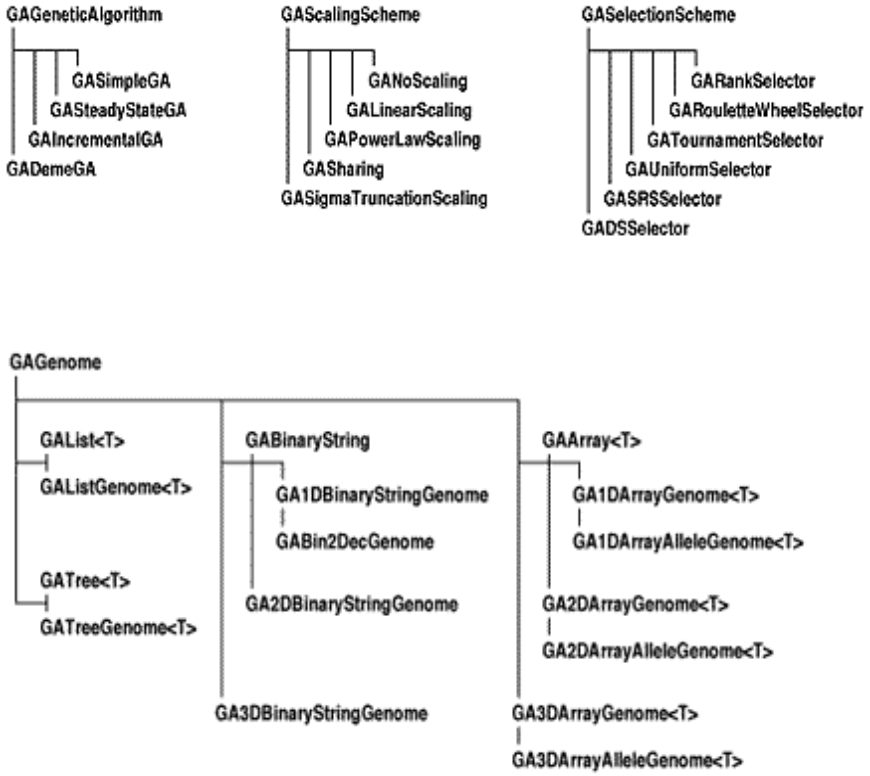


Abbildung 5.5: Die Objekte der GAlib

Die Unterklassen der **GAGeneticAlgorithm** Klasse stellen verschiedene Strategien zur Populationsorganisation und Überführung da. Sie implementieren verschiedene Optimierungsstrategien durch einfache, überlappende oder parallele Populationen, wobei ihre spezielle Konfiguration durch entsprechende Variable und Konstanten erfolgt.

Die Objekte der **GAScalingScheme** Hierarchie dienen zur Normalisierung oder Abbildung der durch die Fitnessfunktion übermittelten Qualitätswerte. Dies ist beispielsweise notwendig, wenn es sich bei den von der Fitnessfunktion gelieferten Werten um negative Zahlen handelt.

Die **GASelectionScheme** Objekte determinieren den Prozess der Selektion von Genomen, die für die Erstellung der nächsten Generation herangezogen werden. Die Unterklassen des **GASelectionScheme** enthalten dabei verschiedene Mechanismen, die bestimmen, welche Genome unter welchen Umständen zur Kreuzung herangezogen werden.

Die **GAGenom** Objekte stellen vordefinierte Strukturen zur Abbildung der zu erstelltenden Lösungsansätze zur Verfügung. Die Vielfalt der Objekte soll es nicht nur ermöglichen, eine unkomplizierte Abbildung der zu optimierenden Informationsstruktur durchzuführen, sondern auch die Performanz des Optimierungsvorgangs zu sichern. Die verschiedenen Genomtypen sind hinsichtlich der in ihnen gehaltenen Informationen optimiert. Die Dokumentation der GAlib enthält entsprechende Empfehlungen, welche Genom Objekte für welche Arten von Informationen verwendet werden sollten. Da die Performanz des Optimierungsvorgangs in diesem Versuch durch den Prozess der Selbstinspektion bestimmt wird, kann die Auswahl des Genomtyps ausschließlich unter Strukturellen Gesichtspunkten durchgeführt werden.

Die Dokumentation der GAlib sieht eine Modellbildung in drei Schritten vor:

- **Repräsentations Definition** Abbildung der Problemlösungsansätze auf eine Datenstruktur (Genom).
- **Transformations Definition** Definition von Kreuzungs- und Mutationsoperatoren.
- **Qualitätsbewertungs Definition** Definition der formalen Qualitätsbewertung eines Lösungsansatzes.

Die genannten Prozesse werden durch eine Anzahl von der Bibliothek beigefügten Beispielen dokumentiert. Sie entsprechen dem im Kapitel 2.3.2 auf Seite 19 beschriebenem Prozess der Modellbildung. Für weitere Informationen sei auf die Dokumentation der GAlib [14] verwiesen.

5.4.2 Das RCXInterpreterWord Objekt

Im Gegensatz zur virtuellen Maschine verwendet die Evolutionsmaschine einen komplexen Datentyp zur Repräsentation der Programmsymbole (siehe auch Ka-

pitel 5.3.2 auf Seite 53). Das **RCXInterpreterWord** Objekt enthält neben der eigentlichen Datenhaltung, die wie bei der virtuellen Maschine durch eine Variable vom Typ **eWord** realisiert ist, auch Hilfsmethoden für die zufällige Generierung von den in den Genomen enthaltenden Regelbasen.

Das **RXCInterpreterWord** unterscheidet die Terminalsymbole der Interpretersprache (siehe Kapitel 4.1.2 auf Seite 36) durch die folgenden in der Enumeration **WordType** definierten Symbolklassen:

- **COMPOUND** Das Symbol ist das zur Serialisierung der Regeln verwendete Trennzeichen "+". Auf der Seite der Evolutionsmaschine repräsentiert dieses Symbol eine eine If - Anweisung.
- **LOGICALOPERATOR** Das Symbol ist eines der 3 Logischen Verknüpfungs Operatoren **AND** , **OR** oder **ANDNOT** .
- **SENSORINPUT** Das Symbol ist eine der in Kapitel 4.1.1 definierten Prämissen
- **ACTION** Das Symbol ist eine der in Kapitel 4.1.1 definierten Aktionen

Aus grammatikalischer Sicht stellen diese Symbolklassifikatoren die Nonterminale einer Grammatik dar, die die von der Programmsprache beschriebenen Sätze fehlerfrei parsen.

Das **RCXInterpreterWord** Objekt besitzt zwei Konstrukor-Methoden, die sich durch ihre Parameter unterscheiden. Wird der Konstruktor mit einem Parameter vom Typ **WordType** aufgerufen, so generiert das Objekt zufällig ein Symbol, welches vom entsprechenden Terminaltyp ist. Der Typ des vom **RCXInterpreterWord** Objekt gehaltenden Symbols definiert sich über die Instanzvariable **eWordType** , die eines der oben beschriebenen Enumerationselemente enthält. Er kann durch Verwendung der *isOfTyp(...)* Methode abgefragt werden. Der Aufruf

RCXInterpreterWord(LOGICALOPERATOR)

generiert z.B. ein **RCXInterpreterWord** Objekt, dessen Instanzvariable *nWord* den Elementindex eines der **eWord** Elemente **AND**, **OR** oder **ANDNOT** enthält und dessen Variable *eWordType* mit dem Wert **LOGICALOPERATOR** belegt ist. Wird der Konstruktor mit einem Parameter vom Typ **eWord** aufgerufen, so wird die Ganzzahlabbildung des Parameters in die Instanzvariable *nValue* gesichert und der Wert der Variable *eWordType* über eine Lookup Table ermittelt. Der Aufruf

RCXInterpreterWord(FWD)

generiert also ein **RCXInterpreterWord** Objekt, welches den Elementindex des Enumerationselements **FWD** der Variable *nValue* zuordnet und *eWordType* auf den Wert **ACTION** setzt.

Der *«-Operator* implementiert eine Abbildung der Variable *nValue* auf eine Zeichenkette. Das bedeutet, dass der Ganzzahlwert des repräsentierten Terminal Symbol auf einen ihm zugeordneten ASCII Charakter abgebildet wird. Diese Methode wird zur Ausgabe des Programmes für Debug oder Kommunikationszwecke benutzt und verwendet zur Abbildung die Konvertierungsmethode *intToChar(...)* des Debugger Objektes.

5.4.3 Das GATreeGenom Objekt

Das **Genom** Objekt repräsentiert das Verhalten des Roboters seitens der Evolutionsmaschine. Es ist damit das direkte Gegenstück des **BehaviorPattern** Objekt der virtuellen Maschine.

Die GAlib bietet ein breites Spektrum von vordefinierten Datentypen zur Realisierung der Genom Objekte. Sie unterscheiden sich in der Repräsentation der in ihnen gehaltenen Informationen. Die Repräsentationsarten reichen von einfachen Zeichenketten und Listen bis hin zu mehrdimensionalen Arrays und Bäumen (siehe Kapitel 5.4.1 auf Seite 62). Der Typ der in den Genom Objekten gehaltenen Daten wird bei Instanziierung des Genom Objektes durch einen Template Mechansimus [13] determiniert. Ebenfalls bei Instanzzierung wird die zur Bewertung der in dem Genom gespeicherten Informationen verwendete Fitnessfunktion festgelegt. Die Fitnessfunktion ist durch eine objective Funktion realisiert und wird als Parameter des Konstruktors übergeben. Die Instanzzierung:

GATree RcxInterpreterWord genome(objective)

generiert ein Genome Objekt vom Typ **GATreeGenome** , welches als Genominformation Objekte vom Typ **RCXInterpreterWord** enthält und zur Berechnung der Fitness die Funktion *objective(...)* verwendet.

Aufgrund der im Kapitel 4 festgelegten Baumstruktur des Genoms wurde als Genomrepräsentation der Datentyp **GATreeGenom** verwendet. Neben dem oben genannten Instanzzierungsparameter bietet das Objekt die folgenden für den Optimierungsvorgang relevanten Konfigurationsmethoden:

- **initializer(...)** nimmt als Parameter die Adresse einer Funktion, die ein Genom mit zufälliger Semantik generiert. Die im Versuch verwendete Funktion *TreeInitializer(...)* nimmt als Parameter ein GATreeGenome Objekt und erstellt unter Verwendung seiner **insert(...)** Methode ein Genom mit zufälliger Semantik, welches dann zur Bildung der Urpopulation verwendet wird. Die Erstellung dieses Programms geschieht unter Berücksichtigung der durch die im Kapitel 4.1.1 definierten Satzstruktur. Die Struktur ist auf Implementierungsebene durch das **eWordType programmGrammer[...]** Array definiert, welches die im Kapitel 5.4.2 definierten Symboltypen in einer der Interpretersprache entsprechenden Reihenfolge anordnet.

  Die *TreeInitializer(...)* Methode generiert, unter Verwendung des *RCXInterpreterWord(...)* Konstruktors, **RCXInterpreterWord** Objekte und fügt sie in den Baum ein. Dies geschieht in der durch die **programmGrammer[...]** Enumeration definierten Reihenfolge, in dem die Funktion über die Enumeration iteriert und das aktuelle Enemurationssymbol als Parameter dem Konstruktor des **RCXInterpreterWord** Objektes übergibt. Die grammatikalische Richtigkeit der Programmstruktur bleibt auf diese Weise gesichert, wohingegen die Semantik der so generierten Regeln zufällig bleibt.

- **crossover(...)** Die *crossover(...)* Methode legt die Methode fest, die bei der Kreuzung zweier Genomobjekte verwendet werden soll. Die Adresse der zu verwendenen Kreuzungsmethode wird dabei als Parameter übergeben. Das im Kapitel 4 determinierte Verfahren zur Kreuzung wird in unserem Versuchsaufbau durch die Methode *GATreeGenome RcxInterpreterWord :: OnePointCrossover* realisiert. Sie transformiert zwei GATreeGenome, indem sie die repräsentierten Bäume an einem zufällig gewählten Knoten kreuzt und die beiden so entstandene Bäume als Kinder zurückliefert. Das heißt, dass jeder der beiden Bäume ”zerschnitten”, und je ein Ast mit dem Ast des anderen Baumes zu einem neuen zusammengeführt wird. (Siehe Kapitel 4.2.4).

  Da die Integrität der Programmstruktur durch diesen Vorgang nicht verletzt werden darf, wurde die Methode *GATreeGenome RcxInterpreterWord ::OnePointCrossover* durch eine Methodenüberladung modifiziert. Die Methode implementiert den Vorgang so, das nur an Knoten gekreuzt wird, die vom Typ **COMPOUND** sind (siehe Kapitel 5.4.2). Da eine Regel eine konstante Länge von 5 besitzt und das **COMPOUND** Symbol immer das erste Symbol einer Regel ist, werden nur die Knoten als Kreuzungspunkte herangezogen, deren Index im Baum (siehe Kapitel 5.4.1) ohne Rest durch 5

Teilbar ist.

- **mutator(...)** Setzt die als Parameter übergebene Adresse einer Methode als Mutationsmethode. Die in der Evolutionsmaschine verwendete Methode *GATreeGenome RcxInterpreterWord ::SwapNodeMutator* des **GATreeGenom** Objektes vertauscht zwei zufällig gewählte Knoten im als Parameter übergebenen Baum. Auch diese Methode musste zur Erhaltung der Integrität modifiziert werden und wurde wie die Kreuzungsmethode überladen. Auf Grund der oben genannten Eigenschaften des Baumes wurde die Methode so verändert, dass die vertauschten Knoten sich, gemessen am Index des Baumes, immer in einem Abstand von 5 zueinander befinden müssen. Dies hat zur Folge, dass nur Knoten, deren Wert vom selben Worttyp (siehe Kapitel 5.4.2 auf Seite 64) ist, vertauscht werden und so die strukturelle Integrität eingehalten wird.

- **write(...)** Die *write(...)* Methode bildet die im Genom gehaltene Baumstruktur auf die für die Kommunikation mit der virtuellen Maschine verwendete Zeichenkette ab. Sie nimmt als Parameter ein Objekt vom Typ **ostream**, welches in C++ die Oberklasse aller zur Ausgabe von Daten verwendeten Streamklassen darstellt und beschreibt diese nach dem im Kapitel 4.2.1 beschriebenem Verfahren.

5.4.4 Das GAPopulation Objekt

Das vom Optimierer verwendete Container Objekt zur Organisation der Individuen einer Generation ist das **GAPopulation** Objekt. Es wird durch den Aufruf seines Konstuktors mit 2 Parametern instanziiert. Der erste Parameter ist ein instanzziertes **GATreeGenom** Objekt (siehe Kapitel 5.4.3 auf Seite 66), welches zur Bildung der Urpopulation verwendet wird. Der zweite Parameter spezifiziert die Größe der Population und ihrer Folgepopulationen:

GATreeGenome RcxInterpreterWord genome(objective)

Erstellt ein GATreeGenome Objekt mit der Fitnessfunktion *objective* , welches Daten vom Typ **RcxInterpreterWord** in einem Baum organisiert.

GAPopulation swappop(genome, 10)

Erstellt ein Populationsobjekt, welches 10 **GATreeGenome** Objekte mit zufälliger Semantik enthält.

5.4.5 Das GASteadyStateGA Objekt

Der im Kapitel 4.2.5 festgelegte Vorgang der Selektion wird durch das Objekt **GASteadyStateGA** realisiert. Das **GASteadyStateGA** Objekt implementiert ein genetisches Verfahren mit überlappenden Populationen. Das bei der Auswahl der zur Kreuzung herangezogenen Individueen verwendete Selektionsschema (siehe Kapitel 4.2.5) ist dabei durch das **RouletteSelectionScheme** Objekt gekapselt. Das **GASteadyStateGA** Objekt leitet von seiner abstrakten Oberklasse **GAGeneticAlgorithm** ab und organisiert den Prozess der Populationsüberführung. Das Objekt wird durch seinen parametrisierten Konstruktor initialisiert. Als Parameter wird eine Instanz des verwendeten Genome Objekts übergeben (in diesem Fall ein **GATreeGenome** Objekt). Um die im Kapitel 4 spezifizierten Konfigurationsparameter des Evolutionsvorgangs zu implementieren wurden die folgenden Methoden des **GASteadyStateGA** Objektes verwendet:

- **population(...)** Durch Aufruf der *population(...)* Methode wird die Urpopulation des Evolutionsprozesses in Form eines Objektes vom Typ **GAPopulation** an das Optimierungsverfahren übergeben.

- **nGenerations(...)** Die *nGenerations(...)* Methode setzt die Anzahl der Generation, die die künstliche Evolution umfassen soll. Es definiert so das Abbruchkriterium, vorausgesetzt es wurde kein anderes Abruckriterium definiert.

- **pMutation(...)** Die Wahrscheinlichkeit für eine zufällige Mutation eines im Genom enthaltenden Baumknoten kann durch die Methode *pMutation(...)* gesetzt werden. Sie wird durch eine Flieskommazahl als Parameter übergeben und spezifiziert wie hoch die Wahrscheinlichkeit für die Mutation eines Knotens des durch das Genom repräsentierten Baumes ist. Die Mutationsmethode wird dabei ausschliesslich auf die Kinder eines Kreuzungsvorgangs angewendet.

- **nReplacement(...)** Spezifiziert die Anzahl von Individuen, die direkt in die nächste Generation übergehen. (Siehe auch Kapitel 4.2.5 auf Seite 46)

Ein Generationswechsel wird durch den Aufruf der *step()* Methode realisiert. Die *step(...)* Methode organisiert die Selektion und Transformation der in der Population befindlichen Genom Objekte auf Basis der in diesem Kapitel beschriebenen Objekte. Jedes Genom der Ausgangspopulation wird durch den Aufruf seiner Fitnessfunktion bewertet. Anschließend wird die durch die *nReplacement(...)* Methode spezifizierte Anzahl von Genomen direkt die Folgepopulation kopiert [2]. Für

[2] Die Anzahl der Direktmigranten berechnet sich aus der Populationsgröße minus dem durch die *nReplacement(...)*Methode spezifizierten Wert

die Generierung der restlichen Genome werden je zwei Genome durch das beschriebene Selektionsverfahren ausgewählt und gekreuzt. Anschließend wird für jeden Knoten der entstanden Genome "gewürfelt" und mit der durch die *pMutation* Variable festgelegten Wahrscheinlichkeit mutiert.

5.4.6 Die Fitness Funktion

Wie im Kapitel 4 beschrieben, ist der Vorgang der Fitnessberechnung dezentral implementiert. Die Evolutionsmaschine delegiert die eigentliche Bewertung an die virtuelle Maschine, indem sie das zu bewertende Genom via Infrarot zum Roboter überträgt.

Auf der Evolutionsmaschine wird die Fitnessfunktion durch eine objective C Funktion realisiert. Sie wird dem Genom Objekt zur Laufzeit zugeordnet (siehe Kapitel 5.4.3 auf Seite 66) und wird vom Programmierer definiert. Der Parameter dieser Funktion ist immer vom Typ der verwendeten Genom Klasse, und stellt das zu bewertende Genom dar, welches so im Funktionsrumpf verfügbar ist.

Der auf der Evolutionsmaschine laufende Teil der Fitnessfunktion reduziert sich auf die Kommunikationsabwicklung mit der virtuellen Maschine. Sie verwendet dazu neben der *doHandshake()* Methode die *getMessageStream()* Methode des **PCCommunicator** Objektes um das Genom zum Roboter zu übertragen. Das von der *getMessageStream()* Methode zurückgelieferte **FileinputStream** Objekt [13] , wird mit Hilfe der *write(...)* Methode des **GATreeGenome** Objekt beschrieben. Die *write(...)* Methode führt eine Breitentraversierung des Baumes durch und schreibt den Inhalt jedes besuchten Knoten mit Hilfe des *«-Operators* auf den **FileinputStream** (siehe Kapitel 5.4.2). Die daraus resultierende Abbildung entspricht dann einer Zeichenkette, die Sätze der im Kapitel 4.1.1 auf Seite 32 definierten Sprache enthält.

Der Übertragungsvorgang beginnt mit der Durchführung eines Handshakes durch den Aufruf der *doHandshake()* Methode eines in der Funktion instanziierten **PCCommunicator** Objektes. Der beschriebene **FileinputStream** wird mit der Methode *broadCastMessageStream()* geschlossen und die mit ihm erstellte Datei versendet. Wurde die Nachricht erfolgreich übertragen, so versucht die Fitness Funktion einen weiteren Handshake zu initiieren. Sobald dies gelingt, hat die virtuelle Maschine die Bewertung des Genoms beendet. Die Fitness Funktion ruft anschliessend die Methode *receiveMessage()* auf, um den Fitnesswert von der virtuellen Maschine zu beziehen. Der empfangene Wert stellt den Rückgabewert der Fitnessfunktion und damit den Qualitätswert des Genoms dar.

5.5 Das Kommunikationsmodul

Die IRCommunicator Objekte stellen einen objektorientierten Layer um das Kommunikationsprotokoll Lnp der LegOS Entwicklungsumgebung dar. Es bietet Mechanismen zur Übertragung von Zeichenketten und numerischen Werten sowie einen primitiven Handshake Mechanismus zur Sicherstellung der Aufnahmebereitschaft des Kommunikationspartners. Alle Nachrichten werden dabei ohne eine spezielle Adressierung versendet.

Die Infratottrancevier des RIS sind für die Kommunikation unter Laborbedingungen entworfen worden und arbeiten nicht zuverlässig bei starker, externer Lichteinstrahlung. Einfallendes Sonnenlicht kann die Kommunikation schwer oder unmöglich machen. Gleiches gilt für im Umkreis befindliche Infrarotsender, die die Nachrichtenübertragung ebenfalls stören.

5.5.1 Das Lnp Protokoll

Die LegOs Entwicklungsumgebung (siehe Kapitel 5.3.1 auf Seite 53) verwendet ein einfaches Netzwerkprotokoll zur Übertragung der Infrarotnachrichten zwischen dem RCX Baustein und einem PC. Durch die Anweisung

#include ‹lnp.h›

werden die benötigten Funktionen deklariert und stehen so anschließend zur Verfügung. Das Lnp Protokoll bietet sowohl die Möglichkeit der adressierten Versendung von Nachrichten, als auch den Versand im Broadcast Verfahren. Die adressierte Variante wird zur Kommunikation mit mehreren RCX Bausteinen verwendet. Da der durchgeführte Versuch sich auf die Kommunikation zwischen RCX und PC beschränkt, wurde hier das Broadcast Verfahren verwendet.

Zum Versenden und Empfangen von Infrarotnachrichten werden 2 Funktionen der Lnp Bibliothek verwendet:

- **lnp_integrity_set_handler(..):** nimmt die Adresse einer Funktion als Parameter, die wiederum eine Zeichenkette und die Länge der übergebenen Zeichenkette als Parameter nimmt. Sie registriert die übergebene Funktion als Callback Funktion im LegOS Kernel und ruft sie immer dann auf, wenn eine Infrarotnachricht vom Kernel empfangen wurde. Die entsprechende Behandlung der übergebenen Zeichenkette obliegt dann der übergebenen Callback Funktion.

- **lnp_integrity_write(..):** nimmt eine Zeichenkette und ihre Länge als Parameter und versendet sie im Broadcastverfahren

Empfang und Versand sind durch asynchrone Verfahren realisiert. Das bedeutet, dass die Nachrichten ”blind” versendet werden, ohne eine Bestätigung des Kommunikationspartners zu erwarten. Übertragungsfehler werden vom Empfänger auf Basis einer mitgesendeten Checksumme erkannt, so dass zumindest fehlerhaft übertragene Nachrichten bemerkt und durch ein entsprechendes Exception-Handling korrigiert werden können.

5.5.2 Das SMWL Protokoll

Das Simple Message Wrapper for Lnp Protokoll ist ein auf Lnp basierendes, proprietäres Netzwerkprotokoll, welches zur Realisierung der Infrarotkommunikation im durchgeführten Versuch dient. Es implementiert eine Nachrichtentypisierung und einen einfachen Handshake Mechanismus. Das Protokoll wird von den Unterklassen des IRCommunicator Objektes verwendet und bildet mit ihnen aus softwaretechnischer Sicht einen Softwareadapter, der die Evolutionsmaschine und virtuelle Maschine verknüpft.

Auf Lnp Ebene wird jede Nachricht als Zeichenkette repräsentiert. Es verzichtet so auf eine Typisierung der versendeten Daten. Um den Prozess der Nachrichtenverarbeitung zu strukturieren, sieht das SMWL Protokoll eine Typisierung vor, die durch eine Ganzzahl, die sich am Beginn jeder versendeten Zeichenkette befindet, realisiert ist. Beim Empfang jeder Nachricht wird diese Zahl durch die entsprechende Methode auf den Enumerationstyp **eMessageTyp** ”gecastet”:

- **0 (ERROR)** Die empfangene Zeichenkette war leer oder der Kommunikationspartner hat eine Ausnahmebehandlung eingeleitet.
- **1 (HANDSHAKE)** Die Nachricht ist die Bestätigung einer gesendeten Nachricht oder ein Teil eines Handshake.
- **2 (MESSAGE)** Die Nachricht enthält prozessbezogene Daten.

Ist die empfangene Nachricht vom Typ **HANDSHAKE** oder **MESSAGE** , so werden die folgenden Symbole entsprechend ihres Typ ausgewertet.

Handelt es sich bei der Nachricht um den Typ **HANDSHAKE** , so enthält der Nachrichtenrumpf eine Ganzzahl, die auf einer der beiden Handshake Nachrichten abgebildet wird:

- **0 (IR_HELLO)** Die Handshake Nachricht repräsentiert die Aufforderung zur Durchführung des Handshakes, das heißt der Sender der Nachricht erwartet als Antwort die Nachricht **IR_HOWAREU** .
- **1 (IR_HOWAREU)** Die Handshake Nachricht repräsentiert die Antwort auf eine Aufforderung zur Durchführung des Handshakes oder die Bestätigung einer empfangenen Nachricht.

Handelt es sich bei der Nachricht um den Typ **MESSAGE** , so enthält der Nachrichtenrumpf einen Satz der im Kapitel 4.1.1 beschriebenen Sprache oder einen Fitnesswert. Der Satz ist in Form einer Zeichenkette repräsentiert und wird vom entsprechenden Communicator Objekt unter Verwendung der *charToInt(...)* Methode des Debugger Objektes auf ein Array vom Typ **eWord** (siehe auch Kapitel 5.3.2 auf Seite 53) abgebildet. Handelt es sich bei der Nachricht um einen Qualitätswert so wird dieser durch ein einfaches Verfahren auf einen entsprechende Zahlentyp konvertiert.

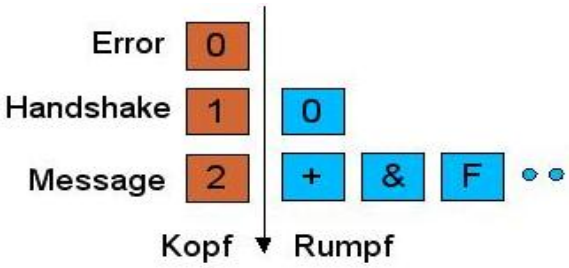


Abbildung 5.6: Die 3 Nachrichten Typen

5.5.3 Die IRCommunicator Objekthierarchie

Das Objekt IRCommunicator stellt die abstrakte Oberklasse der Plattform spezifischen Communicator Objekte dar. Sowohl der auf dem RCX verwendete RCX-Communicator als auch das auf dem PC verwendete Objekt PCCommunicator leiten von dieser Klasse ab und implementieren ihre abstrakten Methoden

5.5.4 Der RCX Communicator

Der RCXCommunicator wird von der virtuellen Maschine zum Empfang der Verhaltensmuster und dem Versand der Fitnesswerte instanziiert und verwendet. Das Objekt registriert bei seiner Instanziierung die Callback Funktion *ir_integrity_handler(...)* für den Nachrichtenempfang (siehe auch Kapitel 5.5.1 auf Seite 71). Die Funktion konvertiert die empfangene Zeichenkette in ein Array von **eWord** Enumerationselementen und signalisiert dem **RCXCommunicator** Objekt, dass eine neue Nachricht empfangen wurde:

```
void my_integrity_handler(const unsigned char *data, unsigned char len){
   if(!irState)
   {
      irDataLength = len - 2
      int msg
      for (int i = 0; (i ‹len); i++) {
         msg = ((int) data[i]) - 48
         msg = d.wordToInt(data[i])
         irdata[i] = (eWord) msg
   }
   for(int i = len (i ‹maxPatternSize) i++)
   {
      irdata[i] = IR_BLANK
   }
   irState = 1
}
```

Das Array *irMessages[maxPatternSize]* ist global definiert, wird aber vom **RCXCommunicator** wie eine Instanzvariable verwendet. [3]. Nur die Callback Methode greift nicht über entsprechende Getter/Setter Methoden des RCXCommunicator Objekts zu. Die Variable **irState** signalisiert dabei dem **RCXCommunicator** Objekt, dass eine neue Nachricht empfangen wurde. Diese Variable wird von einem Polling basierten Event-Mechanismus des **RCXCommunicator** abgefragt, der in der *receiveMessage()* Methode näher besprochen wird.

Das **RCXCommunicator** Objekt besitzt 4 öffentliche Methoden, die die im Kapitel 4.3 besprochenen Funktionen implementieren:

- **doHandshake()** Wartet auf die durch die *doHandshake()* Methode des **PC-Communicator** versendete Handshake Nachricht: **IR_HELLO** und beantwortet sie mit der Handshake Nachricht **IR_HOWAREU** , um danach zu terminieren. Alle Nachrichten anderen Typs werden ignoriert.

- **sendMessage(...)** Versendet eine als Parameter übergebene Zahl solange, bis die Nachricht **IR_HOWAREU** vom Typ **HANDSHAKE** empfangen wurde.

- **receiveMessage()** Wartet auf den Aufruf der Callback Funktion, die für den Nachrichtenempfang im Kernel registriert wurde, und quittiert ihren Empfang durch eine Handshake Nachricht. Die Methode läuft dabei in eine Schleife, die dann terminiert, wenn das von der Callback Funktion manipulierte Flag *irState* gesetzt wird. Wird die registrierte Callback Funktion vom Kernel aufgerufen, so setzt sie das Flag und die Schleife terminiert. Anschließend wird die Handshake Nachricht **IR_HOWAREU** einmal gesendet und das Flag *irState* für den nächsten Nachrichtenempfang wieder auf Falsch gesetzt.

- **getReceivedMessageAt(int anIndex)** Liefert das Zeichen an der Position *anIndex* der empfangenden Zeichenkette zurück. Die Adressierung wird dabei direkt auf dem in der Callback Funktionen gefüllten Array *irData[..]* durchgeführt. Der Index *anIndex* wird dabei um eins erhöht, da das an erster Stelle stehende Typisierungssymbol nicht Teil der versendeten Nachricht ist. Diese Methode wird für die Übertragung der empfangenen Nachrichten in die Regelbasis (**BehaviourPattern**) der virtuellen Machine verwendet.

[3] Unter Gesichtspunkten der Objektorientierung wäre in diesem Fall auch die Verwendung einer Klassenvariable möglich gewesen

5.5.5 Der PC Communicator

Der PCCommunicator ist Teil der **Evolutionsmaschine** (siehe Kapitel 4) und dient zur Kommunikation mit dem **RCXCommunicator** . Das Objekt ist ausschließlich für die Verwendung auf Windows Systemen ausgelegt, da es zur Kommunikation nicht direkt die Lnp Bibliothek einbindet, sondern das vorkompilierte Programm **ir.exe** durch einen Systemaufruf anstößt. Das Programm basiert auf den MFC und verwendet die LNP Bibliothek für Windows. Es wird mit 4 Parametern aufgerufen:

- Als erstes wird eine der beiden Zeichenketten **RECEIVE** oder **SEND** übergeben. Dieser Parameter spezifiziert, ob die folgenden Parameter die Randbedingungen für den Empfang oder den Versand einer Zeichenkette sind.
- Der zweite Parameter spezifiziert die serielle Schnittstelle, an der der IR Tower angeschlossen ist.
- Der dritte Parameter ist ein Timeoutwert, der spezifiziert, wann die Kommunikation abgebrochen werden soll, wenn keine Nachricht empfangen oder gesendet werden konnte.
- Der letzte Parameter gibt den Pfad zu der Datei an, die die zu sendende Nachricht enthält.

Der Inhalt der Datei OUTMSG wird z.B. über die erste serielle Schnittstelle 8 Sekunden lang gesendet, indem das Programm wie folgt aufgerufen wird:

ir.exe SEND com1 8000 OUTMSG

Entsprechend führt der Aufruf

ir.exe RECEIVE com1 8000 INMSG

dazu das 8 Sekunden lang versucht wird eine Nachricht über den an der ersten seriellen Schnittstelle angeschlossenen Tower zu empfangen und den Inhalt der empfangenen Nachricht in die Datei INMSG zu speichern. Für die Versuchsdurchführung sind diese Aufrufe in den Batch Dateien **send.bat** und **rec.bat** gekapselt.

Die Methoden des **PCCommunicator** dienen zum Versand eines zu bewertenden Genoms auf der einen und dem Empfang der berechneten Qualität des versendeten Genoms auf der anderen. Das Objekt implementiert dazu die folgenden öffentlichen Methoden:

- **doHandshake()** Sendet solange ein Acknowledge Signal bis eine Antwort des **RCXCommunicator** empfangen wurde.
- **getMessageStream()** Liefert ein Objekt der Klasse **FileinputStream** zurück, auf den beliebige Zeichen geschrieben werden können. Das Streamobjekt wird in der Fitnessfunktion der Evolutionsmaschine an die *write(..)* Methode eines Genoms übergeben, welche den Inhalt des Genoms wie im Kapitel 5.4.3 beschrieben auf eine Zeichenkette abbildet.
- **broadCastMessageStream()** Versendet die auf das **FileinputStream** geschriebenen Zeichen und wartet auf ein Acknowledge durch den Kommunikationspartner.
- **receiveMessage()** Empfängt eine Ganzzahl und sendet ein Acknowledge zum Kommunikationspartner.
- **getReceivedMessage()** Liefert die empfangende Zahl zurück.

Das in der Abbildung 5.7 enthaltende Sequenzdiagramm beschreibt den Nachrichtenaustausch zwischen virtueller Maschine und der Evolutionsmaschine. Die Infrarotkommunikation repräsentiert dabei die über das LNP Protokoll ausgetauschten Daten, die durch das beschreibene SMWL Protokoll definiert sind.

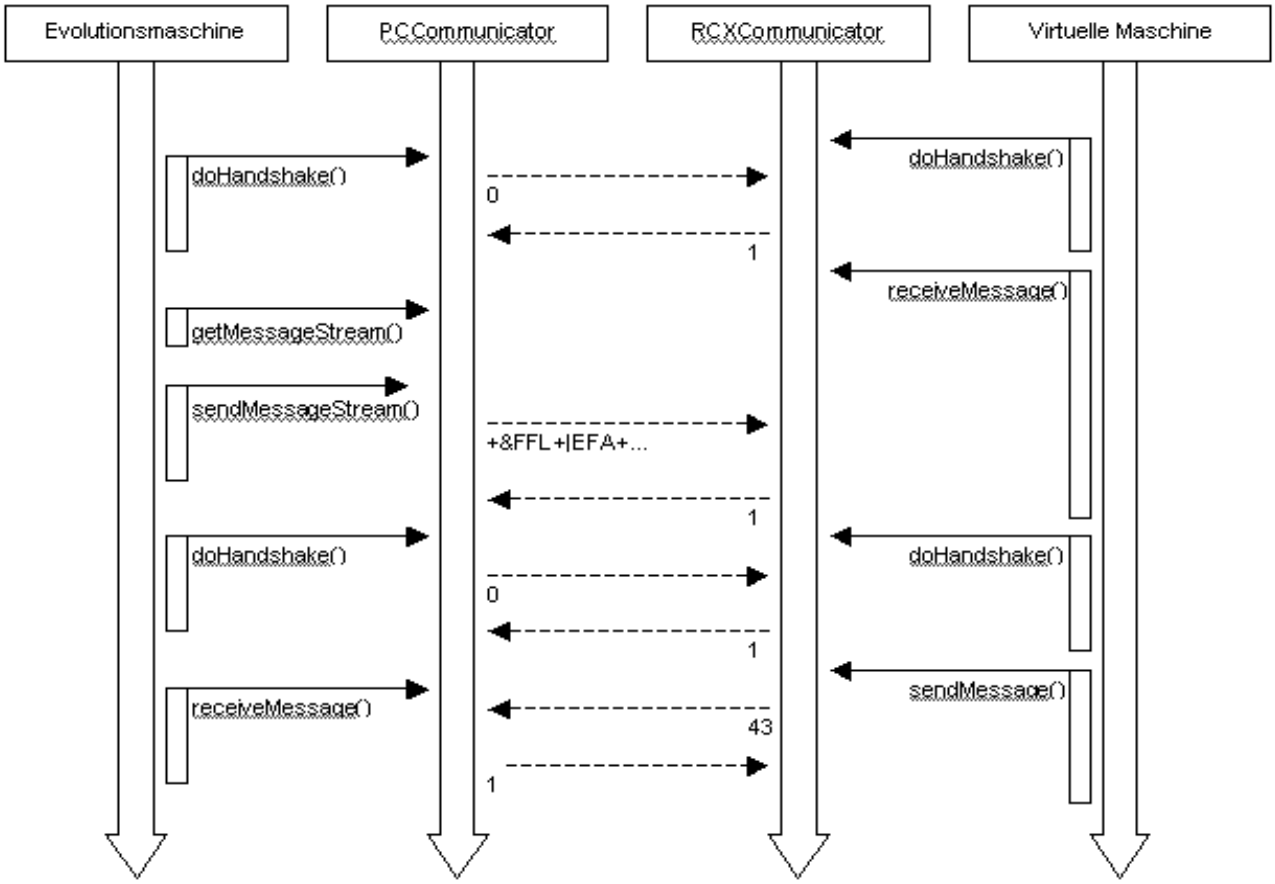


Abbildung 5.7: Sequenzdiagramm der Infrarotkommunikation

5.6 Die Debugger Objekthierarchie

Die Entwicklung von ”Embedded Systems”, wie mobile Roboter sie darstellen, ist unter Entwicklungsgesichtspunkten ein besonderer Vorgang. Das Debugging von auf dem System laufenden Programmen ist oft ein komplizierter Vorgang, da aufgrund der häufig beschränkten Ressourcen und I/O Geräten ein direktes Debugging nicht möglich ist. Der RCX besitzt ein einzeiliges Display und 32 KB Hauptspeicher. Da zum Zeitpunkt der Entwicklung keine entsprechende Methode für ein PC gestütztes Debugging zur Verfügung stand, muss sich der Debug-Vorgang auf das Ausgeben von Informationen beschränken, die den Randbedingungen der Hardware gerecht werden.

Zu diesem Zweck wurde eine Objekthierarchie implementiert, die ein rudimentäres Debugging auf dem RCX Baustein ermöglicht.

5.6.1 Das Debugger Objekt

Das Debugger Objekt stellt die abstrakte Oberklasse der beiden Objekte **RCXDebugger** und **PCDebugger**. Sie deklariert die Methoden, die von ihren Unterklassen zur Ausgabe von Debug Informationen implementiert und genutzt werden.

Darüber hinaus implementiert sie die Konvertierungsmethoden *intToChar(...)* und *charToInt(...)* . Die Methoden implementieren den im Kapitel 4.1.1 auf Seite 32 beschriebenen Vorgang der Abbildung von programminternen Repräsentationen der Regelsymbole auf die für die Kommunikation verwendete Darstellung. Die Abbildung vollzieht sich in 2 Schritten: Die durch die eWord Enumerationssymbole repräsentierten Programmsymbole werden zunächst auf ihren Integerwert (der Index des Symbols in der Enumeration) abgebildet. Der Integerwert wird anschließend über die Adressierung in einer Lookup Table in den entsprechenden ASCII Charakter konvertiert. Die **IRCommunicator** Objekte verwenden diese Methoden, um die von der Evolutionsmaschine generierten Programme für die Infrarot Kommunikation auf eine Zeichenkette abzubilden.

1. Enumeration: | *EYE EYE FWD* (Programm interne Repräsentation)
2. Integerwert: *5 12 12 7* (Zwischendarstellung)
3. Zeichenkette: |*EEF* (Reduzierte Repräsentation)

5.6.2 Das RCXDebugger Objekt

Um eine vereinheitlichte Behandlung von Debuginformationen zu ermöglichen, wurde das **RCXDebugger** Objekt implementiert. Alle von der virtuellen Maschine generierten Debuginformationen werden durch den Aufruf der *debug(...)* Methode des **RCXDebugger** Objektes ausgegeben. Die *debug(...)* Methode gibt die übergebene Zeichenkette, eine Ganzzahl oder ein Enumerationselement, auf der LCD Anzeige aus und wartet 300 Millisekunden. Dies hat Einfluss auf die Echtzeitvorgänge (Kommunikation und Regelinterpretation) des Gesamtsystems. Wird die Methode *debug(...)* beispielsweise zwischen zwei Regelinterpretationen aufgerufen, so führt dies dazu, das der Roboter 300 Millisekunden nicht auf Sensoreingaben reagieren kann. Er würde also ”blind” gegen eine Wand fahren, da die eingestellte Motorenkonfiguration gültig bleiben würde. Ebenfalls implementiert ist die Methode *accousticDebug(...)*, die eine der Methode übergebene Ganzzahl als Ton moduliert in seiner Frequenz ausgibt. Diese Methode kann für das Signalisieren bestimmter Zustände verwendet werden, eignet sich jedoch weniger für die Inspektion von Variabeleninhalten oder verwendeten Parametern.

5.6.3 Das PCDebugger Objekt

Obwohl die Evolutionsmaschine keine Beschränkungen im Bezug auf die Ausgabe von Debuginformationen besitzt, wurde auch hier ein Debugger Objekt implementiert. Der **PCDebugger** übernimmt die gleiche Aufgabe wie der **RCXDebugger** , ist aber so implementiert, das er die übergebene Zeichenkette direkt an das cout Objekt weiterleitet. Die so eingeführte Zwischenschicht soll ein späteres Portieren der Evolutionsmaschine auf den RCX Baustein erleichtern (siehe Kapitel 7.2).

Kapitel 6

Die Versuchsdurchführung

Die Versuchdurchführung soll Aufschluss über die Anwendbarkeit genetischer Verfahren auf Problembereiche der Robotik geben und im speziellen klären, ob diese in angemessener Zeit zur Problemlösung führen.

Der Versuch besteht aus 4 Versuchsdurchläufen, die sich in der Konfiguration des Versuchaufbaus unterscheiden. Die Analyse der beim Optimierungsprozess erstellten Logdateien soll Aufschluss über die Effektivität des Lernprozesses und seinen Teilprozessen geben. Im Speziellen soll sie klären, ob die gewählte Kreuzungsmethode eine ausreichende Systematik zum effektiven Durchqueren des Suchraumes bietet und die Mutationsverfahren eine ausreichenden Streuung der Lösungsansätze gewährleisten, ohne die Heuristik des Suchverfahrens zu neutralisieren. Es soll weiter untersucht werden, ob die Formalisierung der Lernaufgabe (Fitnessfunktion) korrekt durchgeführt wurde und so das vorgegebene Problem gelöst werden kann.

6.1 Die untersuchten Eigenschaften

Jeder Versuchdurchlauf wird eingangs durch zwei Tabellen dokumentiert, die die wichtigsten Eigenschaften und Randbedingungen des Suchverfahrens dokumentieren. Zur Visualisierung der Verfahrenseffektivität enthält die Dokumentation weiter zwei Diagramme, welche die durchschnittliche Qualität der drei besten Genome einer Generation und die durchschnittliche Qualität der neu generierten Genome mit dem entsprechenden Generationsindex in Relation setzen. Die Graphen werden durch eine Regressionsgerade geschnitten, um eine valide Aussage über die Veränderung der Qualität während der Optimierung zu treffen. Eine anschließende Analyse der Graphen versucht die Eigenschaften der Funktionen zu

dokumentieren und Kausal- oder Korrelations-Beziehungen zwischen ihnen aufzudecken.

6.1.1 Die Checkliste

Die Versuche werden durch eine Reihe von Eigenschaftsausprägungen dokumentiert. Die Attribute stellen zum einen die Konfigurationsparameter des genetischen Optimierers und zum anderen die Eigenschaften des eigentlichen Optimierungsvorgangs dar:

Anzahl der Iterationen über die Regelbasis

Der Parameter spezifiziert die Dauer, die jedem Genom zur Qualitätsbewertung zur Verfügung steht. Da eine durch ein Zeitintervall definierte Interpretationsdauer aufgrund der Hardware und Betriebssystem spezifischen Komponenten keinen angemessenen Vergleichswert liefert, wird dieser Parameter in Iterationsanzahl gemessen. Der Wert gibt also an, wie oft die Regelbasis von vorne interpretiert wird (siehe Kapitel 4.1.2 auf Seite 36). Ob das zurücksetzen der Regelbasis durch Feuern einer Regel (Subsumption) oder durch die Interpretation der letzten Regel in der Wissensbasis erfolgt, ist hier unabhängig. Der Selbstinspektionsvorgang kann deshalb in seiner zeitlichen Ausdehnung variieren . Dies muss jedoch hingenommen werden, da eine durch ein Zeitintervall determinierte Interpretationsdauer zu einer variablen Anzahl von Iterationsschleifen führen würde, was die Vergleichbarkeit des Fitnesswertes problematisch gestaltet. Genome mit einer niedrigen Anzahl von Iterationsdurchläufen würden aufgrund der niedrigen Stichprobenanzahl einen weniger aussagekräftigen Wert liefern als ein Genom, welches durch unverhältnismässig viele Iterationsschleifen bewertet wurde.

Anzahl der Generationen / Grösse der Populationen

Die beiden Eigenschaftsausprägungen bilden mit der Anzahl der Iterationen die Stellgrössen, die die Dauer des Optimierungsvorgangs determinieren. Im speziellen ist der Zusammenhang zwischen der Anzahl der Generationen und der größe der Populationen zu untersuchen, da sie die algorithmische Eigenschaft des Suchverfahrens unmittelbar verändern. Die Dauer des Interpretationsvorgangs ist zwar ebenso unmittelbar an der Determinierung der Gesamtdauer des Optimierungsvorgangs beteiligt, verändert jedoch ausschließlich die statistische Aussagekraft der Fitnesswerte und nicht die algorithmischen Eigenschaften des Suchverfahrens. Eine aussagekräftige Bewertung der Qualität ist jedoch eine unverzichtbare Randbedingung, die bei der Festlegung der Interpretationsdauer beachtet werden

muss. (siehe Kapitel 4.1.3 auf Seite 37). Eine von der eigentlichen Qualität vollständig unabhängige Fitnessfunktion würde zu einem unsystematischen ”herumspringen” im Suchraum führen und würde somit einer nicht heuristischen Suche gleichkommen.

Anzahl der Direktübernahmen

Der Parameter bestimmt den Überlappungsgrad zweier aufeinander folgender Populationen. Dieser Parameter ist bei der begrenzten Anzahl der Generationen eine sensible Stellgröße. Einerseits muss das Verfahren schnell an guten Lösungen festhalten, um am Ende der Optimierung nicht ohne Lösung ”dazustehen”. Anderseits muss es genug Innovation zulassen, um eine Optimierung zu ermöglichen. Wird der Grad der Überlappung zu hoch gewählt, so besteht die Gefahr, dass die Qualität des besten Indivuums auf einem lokalen Qualitätsmaximum stagniert.

Wahrscheinlichkeit für Mutation

Die Mutation stellt eine von der systematischen Suche (siehe Kapitel 4.2.4 auf Seite 44) entkoppelte Komponente des Suchverfahrens dar. Der Eigenschaftswert repräsentiert die Wahrscheinlichkeit, dass eine Teilinformation des Genoms zufällig verändert wird. Auch hier ist die Dauer der Gesamtevolution von entscheidender Bedeutung. Wird z.B. eine qualitativ geringwertige Urpopulation erstellt, so stellt die Mutation eine wichtige Methode zur Kompensierung der schlechten Ausgangsposition bei der Suche dar. Wird der Mutationsgrad hingegen zu hoch gewählt, so kann dies dazu führen, dass die Kinder zweier erfolgreicher Genome eine zu große Distanz im Ausprägungsraum zu ihren Eltern besitzen.

Shaftencoder

Da die von der Evolutionsmaschine generierten Muster auch Programme beinhalten können, die aufgrund ihrer logischen Struktur keine Aktionsausführung ermöglichen, und somit der Roboter keine Bewegung durchführt, muss möglicherweise ein bewegungsmessender Sensor in die Bewertung des Genoms einbezogen werden. Die Interpretationsmethode der vom Shaftencoder generierten Daten, wird durch die Eigenschaftsausprägung *Shaftencoder* in der Checkliste des Versuchs dokumentiert.

Wahrscheinlichkeit für Kreuzung

Die Selektionswahrscheinlichkeit repräsentiert die Wahrscheinlichkeit, dass ein Genom für den Kreuzungsvorgang ausgewählt wird. Die an einem Kreuzungsvorgang beteiligten Genome werden für jede Kreuzung durch ”Würfeln” ermittelt.

Der Wert der Kreuzungswahrscheinlichkeit wird durch das **GARouletteWheel-Selector** (siehe Kapitel 5.4.1) Verfahren ermittelt. Das bedeutet, dass die Wahrscheinlichkeit eines Genoms, an einer Kreuzung teilzunehmen, durch die relative Häufigkeit des Fitnesswertes gemessen an der Summe der Fitnesswerte einer Generation ist. Dieses Verfahren schließt auch die Kreuzung schlechterer Genome nicht aus, benachteiligt diese jedoch. Das Verfahren lässt auf diese Weise Kreuzungen von Genomen mit sich selbst zu, was dann eher einer Mutation als einer Kreuzung entspricht.

Semantik des ! Operator

Der logische Operator ! wurde im Kapitel 4.1.1 mit der Semantik der logischen *nand* Verknüpfung belegt. Diese Semantik gilt jedoch nur für einen Teil der durchgeführten Versuche. Die Interpreterkonfiguration der ersten beiden Versuche interpretiert den Operator als ein Fallback Verhalten. Das bedeutet, dass der Prämissenteil unabhängig von den beiden enthaltenen Prämissen immer zu wahr evaluiert. Diese Sonderbehandlung stellt eine Reduktion des durch die Programmgrammatik aufgespannten Suchbaums dar, da den in der Regel enthaltenden Prämissen keine Bedeutung meht zu gemessen wird.

Subsumption

Der Parameter gibt an, ob der Interpreter eine Subsumptions-Architekur umsetzt oder das Verhalten durch die Interferenz der anwendbaren Regeln entsteht (siehe Kapitel 4.1.2). Es gilt zu Untersuchen, ob dieser Architekturansatz für die gestellte Aufgabe sinnvoll ist oder nicht. Ein Interferieren verschiedener Verhalten (wie es auch bei menschlichem Verhalten beobachtet werden kann) kann im Bereich der Feinsteuerung eines Roboters durchaus sinnvoll sein.

6.1.2 Der Stammbaum des Genoms

Der Stammbaum des Genoms stellt die Entwicklung der beiden besten Genome aus der Urpopulation in einem gerichteten Graphen dar. Jedes Genom wird durch die Genominformation und eine ihm zugeordnete Qualität dargestellt. Ist das Genom das Kind zwei bestehender Genome, so enthält der Knoten das durch die Kreuzung entstandene Genom und die bewertete, mutierte Variante des Kindes. Die Richtung der Pfeile stellt dabei die Generationshierarchie dar.

6.2 1. Versuch

Der erste Versuch soll Aufschluss über die Anwendbarkeit der gewählten Systemarchitektur und der Konfiguration des gewählten Lernverfahrens geben. Aufgrund der begrenzten Anzahl der Versuchsdurchläufe enthalten alle Versuchsergebnisse eine starke Zufallskomponente. Die Analyse der Versuchsergebnisse und die daraus abgeleiteten Rückschlüsse müssen unter diesem Gesichtspunkt gesehen werden.

6.2.1 Die Versuchsparameter

Parameter	Wert
Anzahl der Iterationen über die Regelbasis	1000
Anzahl der Generationen	10
Grösse der Populationen	10
Anzahl der Direktübernahmen	5
Wahrscheinlichkeit für Mutation	0.5
Transformationsorganisation	Qualitätsgewichtetes Roulette
Semantik des ! Operators	Prämisse evaluiert immer zu Wahr
Shaftencoder	Nein
Subsumption	Nein

Konfiguration der Versuchsparameter

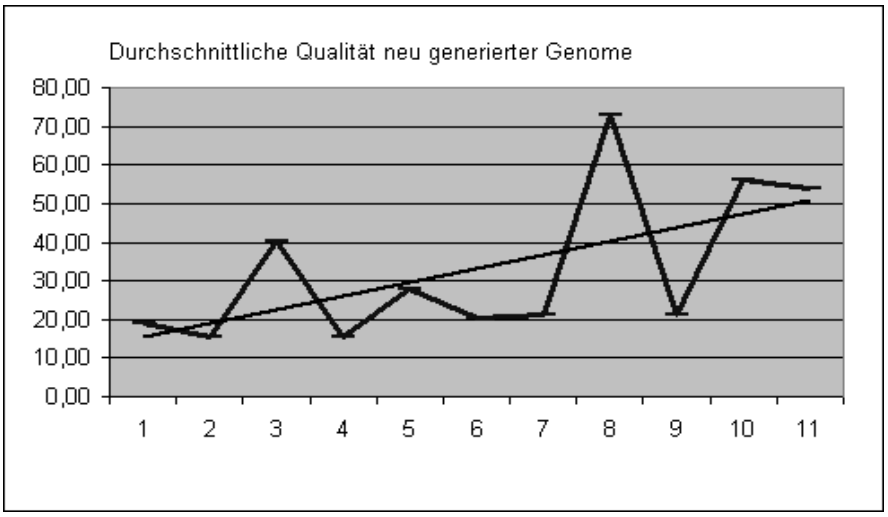
6.2.2 Versuchsdurchlauf 1

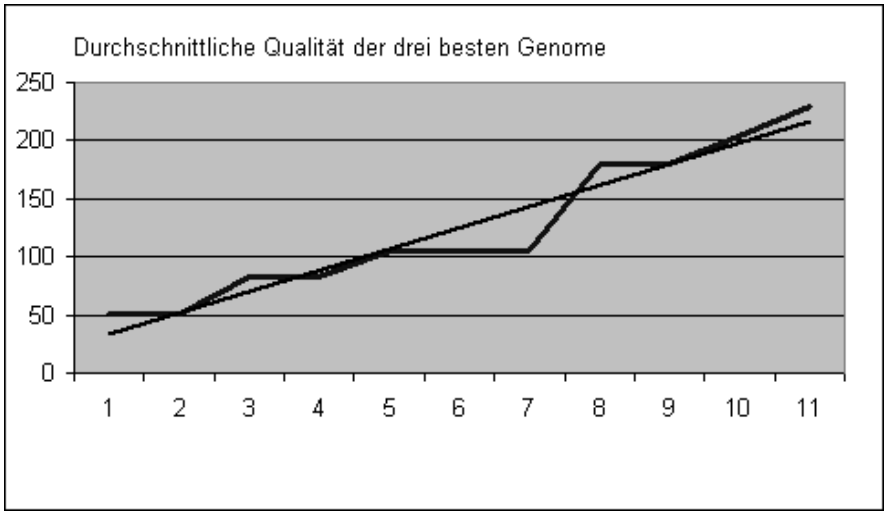
Checkliste

Evolutionseigenschaft	Eigenschaftsausprägung
Generation in der das beste Genom generiert wurde	8
Das beste Muster ist aus einer Kreuzung entstanden	Ja (Selbstkreuzung)
Steigung der Qualität der Kinder	3,5
Steigung der Qualität der drei Besten	18,7

Eigenschaften des Suchprozesses

Graphische Analyse





Ergebnisdiskusion

Beide Graphen besitzen eine steigende Regressionsgerade. Da ein Überlappungsgrad von 5 gewählt wurde, wird die Qualität der besten drei Genome wie zu erwarten durch eine monoton steigende Funktion repräsentiert. Positiv zu bewerten ist der eindeutige Anstieg der Qualität der neu generierten Genome, was für die verwendete Heuristik des Optimierungsverfahren spricht. Das heißt, dass trotz der zufälligen Komponenten [1] sich die heuristische Eigenschaft des Suchverfahrens durchsetzen kann und die steigende Qualität der Gesamtpopulation sich nicht nur auf ein Bewerten und Selektieren zurückführen lässt. Andererseits enthält das Verfahren genug zufällige Anteile, um auch scheinbar schlechte Genome mit in die Berechnung des als nächstes zu bewertenden Genoms einzubeziehen. Das Verfahren läuft also nicht Gefahr, sich in lokalen Maxima festzufahren, wie es die Hill Climbing Verfahren tun können.

Die Qualitätssprünge beider Graphen korrelieren im Bezug auf ihren Generationsindex. Die auftretenden Wechsel zwischen über mehrere Generationen konstant bleibender Qualitätsplateaus der Besten und dem dann relativ starken Anstieg der Qualität kann seine Ursache im Erreichen eines bestimmten Verhaltenstyp haben. Das bedeutet, dass ein Programm was z.B. blind gegen eine Wand rennt, natürlich eine weitaus geringere Fitness besitzt(nämlich 0) als ein Programm, was sich einfach im Kreis dreht und so einen Fitnesswert größer 0 besitzt. Aus dem Verfolgen der Linie resultiert logischerweise eine wiederum deutlich höhere Fitness als aus einem sich im Kreis drehen. Da es Aufgrund des beschränkten Suchraums wenig Spielraum für Lösungen zwischen diesen drei Typen gibt, wird das beste Genom erst dann ersetzt, wenn ein Muster des jeweils besseren Verhaltenstyp gefunden wurde, welches dann wiederum gleich einen erheblichen Anstieg der Qualität zur Folge hat. Diese Begründung ist jedoch als Hypothese zu begreifen und könnte erst durch eine ausreichende Anzahl von Stichproben bewiesen werden.

Etwas anders verhält es sich mit der Qualität der Kinder, da hier vermehrtes Auftreten gleicher Lösungstypen zu einer Erhöhung des Durchschnittswert führen sollte. Die Funktion ist nicht monoton steigend, sondern zeigt starke Schwankungen im Abbildungsraum. Die abgeleitete Regressionsgrade zeigt jedoch die positive Tendenz des Gesamtverlaufs. Das Auf und Ab der Funktion begründet sich wiederum in der vom Zufall beeinflussten Transformation und der Beschränktheit des Problemraums. Leichte Veränderungen eines guten Musters können schon eine erhebliche Qualitätseinbusse nach sich ziehen und umgekehrt. Entsprechend

[1] Auch der systematische Teil des Verfahrens ist vom Zufall beeinflusst

kann die Qualität der Kinder stark variieren. Die zufällige Komponente der Kreuzung wird jedoch mit zunehmender Qualität der 5 Besten aufgeweicht und lässt so einen tendenziellen Anstieg der durchschnittlichen Qualität zu.

Stammbaum des besten Genoms

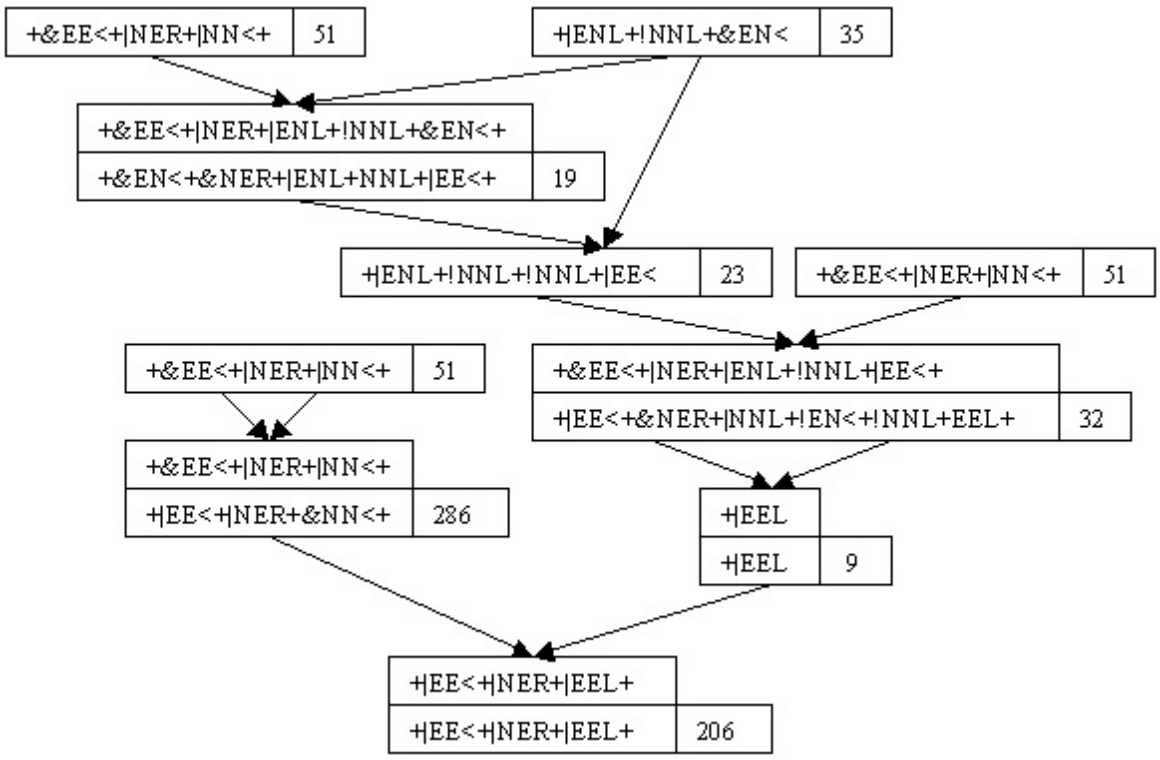


Abbildung 6.1: Stammbaum der beiden besten Genome des 1.Versuch

Der Graph des Versuchs dokumentiert die positiven Auswirkungen des hohen Überlappungsgrads des Verfahrens. Obwohl das beste Pattern (mit einer Fitness von 286) erst in der 7. Generation generiert wird, ist es ein direkter Nachfahre eines der in der Urpopulation generierten Genome. Die Direktübernahme der Besten ermöglicht es dem Verfahren, ein erfolgreiches Genom immer wieder mit anderen, neu generierten, Genomen zu kreuzen. Dem Verfahren bleibt also mehr Zeit, verschiedene Variationen guter Genome auf ihre Qualität hin zu überprüfen.

Der Vorfahre des Besten kreuzt sich als erstes mit sich selbst, was jedoch keine Veränderung nach sich zieht.Der entscheidende Modifikationsschritt wird erst durch die anschließende Mutation geleistet. Die Mutationsmethode tauscht den

logischen Operator der letzten Regel mit dem der ersten. Aus logischer Sicht bedeutet dieser Schritt keine Modifikation der Semantik. Warum die Qualität des Genom trotzdem um ein Vielfaches erhöht wird, kann unter anderem mit zwei Hypothesen erklärt werden.

Zum einen wird die Interpretation der Prämissen seriell und für jede Regel erneut durchgeführt.[2] Dies könnte durch eine Modifikation des Interpretationsvorgangs umgangen werden. Würden die Sensoren nur einmalig, am Anfang der Interpretation der Regelbasis ausgelesen, würden alle Regeln mit derselben Faktenbasis arbeiten. Der Grenzbereich zwischen Linie und weißer Fläche ist ein Beispiel dafür, dass eine Regel, die dieselbe Prämisse durch ein logisches Oder verknüpft, eher angewendet wird als eine Regel, die dies durch einen Und Operator tut.

Zum anderen könnte auch eine fehlerhafte Fitnessfunktion eine solche Qualitätsschwankung hervorrufen. Obwohl die vom System generierten Lösungen anwendbar waren und die Qualität der Kinder eine steigende Tendenz zeigte, kann z.B. ein mit der Startposition der Roboters korrelierter Fitnesswert nicht ganz ausgeschlossen werden. Der Eintrittswinkel des Roboters auf die schwarze Linie kann Einfluss auf das Verhalten des Roboters haben. Anderseits ist die steigende Qualität der Kinder ein Zeichen für die Heuristik des Suchverfahrens, was diese Hypothese eher unwarscheinlich erscheinen lässt.

Eine ebenfalls zu beobachtende Eigenschaft ist, dass das Kind zweier Genome oft einen mit den Eltern korrelierten Qualitätswert besitzt. Bei vielen Kindern ist zu beobachten, dass ihr Qualitätswert im von der Qualität der Eltern definierten Intervall liegt oder manchmal sogar wie eine Mittelwertbildung anmutet. So entsteht z.B. das zweitbeste Genom direkt aus dem besten Genom. Da es mit einem qualitativ schlechten Genom gekreuzt wird, verliert es jedoch fast ein Drittel seiner Qualität. Dieser Effekt sei jedoch im Hinblick auf die geringe Anzahl der Versuche zunächst dem Zufall zuzuordnen.

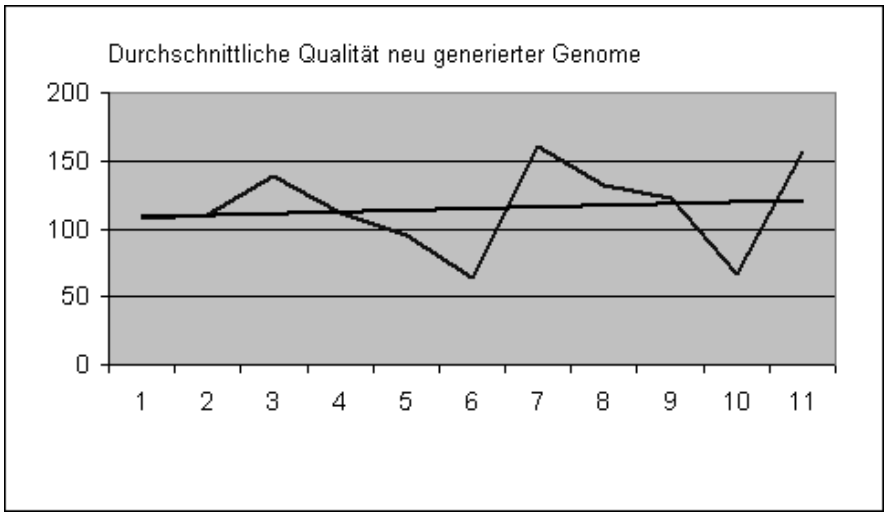
[2] Das bedeutet aus Sicht eines Regelsystems, dass sich die Faktenbasis zwischen zwei Regelinterpretationen verändern kann

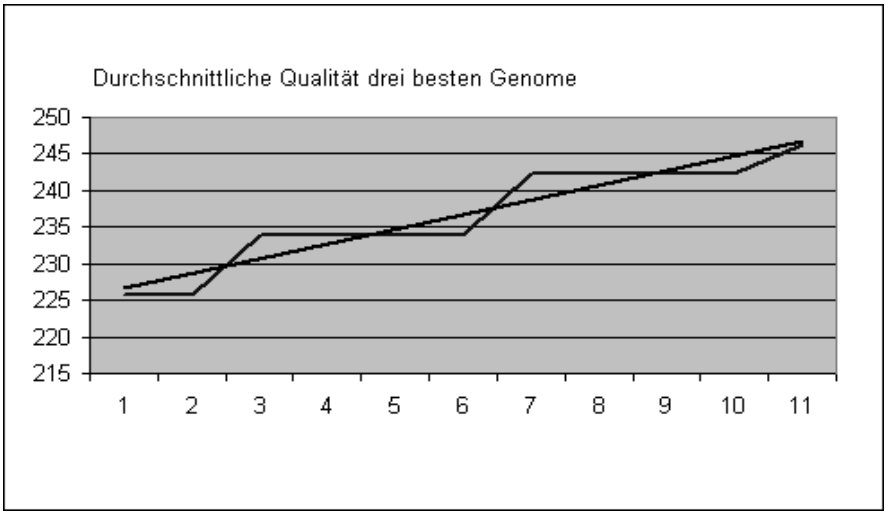
6.2.3 Versuchsdurchlauf 2

Evolutionseigenschaft	Eigenschaftsausprägung
Generation in der das beste Genom generiert wurde	7
Das beste Muster ist aus einer Kreuzung entstanden	Ja
Steigung der Qualität der Kinder	1,13
Steigung der Qualität der Besten	2

Eigenschaften des Suchprozesses

Graphische Analyse





Ergebnisdiskussion

Der zweite Versuchsdurchlauf mit der genannten Versuchskonfiguration zeigt die gleichen Eigenschaften wie der erste, mit dem Unterschied, dass beide Funktionsgraphen eine weitaus geringere durchschnittliche Steigung besitzen. Das Verfahren startet bereits auf einem sehr hohen Qualitätsniveau. Wie bereits in der Analyse des 1. Versuchsdurchlauf beschrieben, können die Muster im Bezug auf ihre Problemlösungsstrategie in nur 3 Klassen eingeteilt werden. Es ist für das Verfahren also sehr schwer, qualitativ höherwertige Alternativen zu bereits guten Mustern zu entwickeln. Dies schlägt sich in beiden Graphen nieder. Die Lernkurve dieses Systems scheint bei dieser Aufgabenstellung in hohen Qualitätsbereichen zu stagnieren.

6.2.4 Schlussfolgerungen

Die gewählte Versuchkonfiguration scheint den Anforderungen der Problemstellung zu genügen. Das Verfahren generierte in drei von vier Versuchsdurchläufen Muster, die der Linie folgten. Die im Kapitel 4.1.3 angeführte Problematik des ”stehen bleiben” führte im nicht erfolgreichen Durchlauf zu einer falschen Bewertung der Muster. Logischerweise wird ein über der Linie stehender Roboter mit einer hohen Qualität bewertet. Die Fitnessfunktion ist also trotz des überwiegenden Anteils an erfolgreichen Versuche nicht robust und damit falsch.

6.3 2. Versuch

Der im zweiten Versuch verwendete Interpreter setzt im Unterschied zum ersten Versuch eine Subsumptions-Architektur um (siehe Kapitel 4.1.2 auf Seite 36). Diese Modifikation soll Aufschluss darüber geben, inwiefern eine Subsumptions-Architektur eine merkliche Steigerung der Lerneffektivität nach sich zieht. Da die Fitnessfunktion keine Bewertung einzelnder Regeln durchführt, sondern immer die gesamte Regelbasis bewertet wird, ist jedoch keine erhebliche Veränderung der Lerneffektivität zu erwarten.

6.3.1 Die Versuchsparameter

Parameter	Wert
Anzahl der Iterationen über die Regelbasis	1000
Anzahl der Generationen	10
Grösse der Populationen	10
Anzahl der Direktübernahmen	5
Wahrscheinlichkeit für Mutation	0.5
Transformationsorganisation	qualitätsorientiertes Roulette
$\nu Semantikdes!Operators$	Prämisse evaluiert immer zu Wahr
Shaftencoder	Nein
Subsumption	Ja

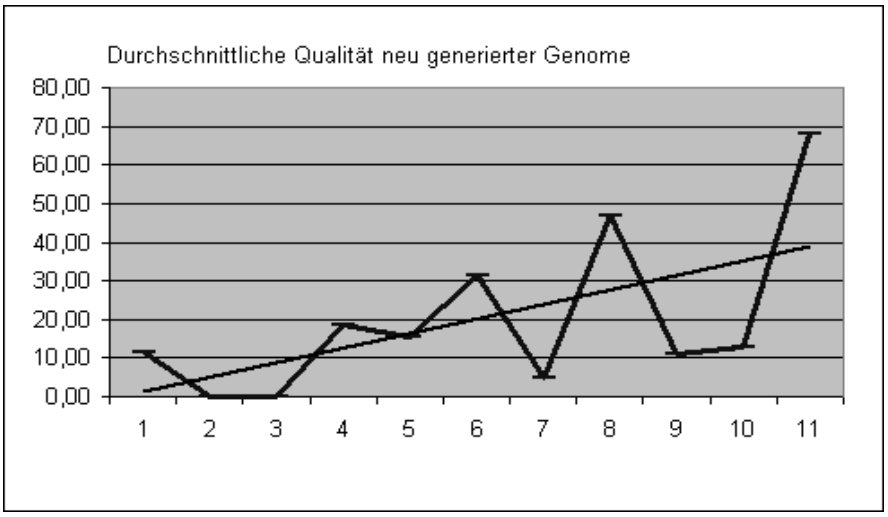
Konfiguration der Versuchsparameter

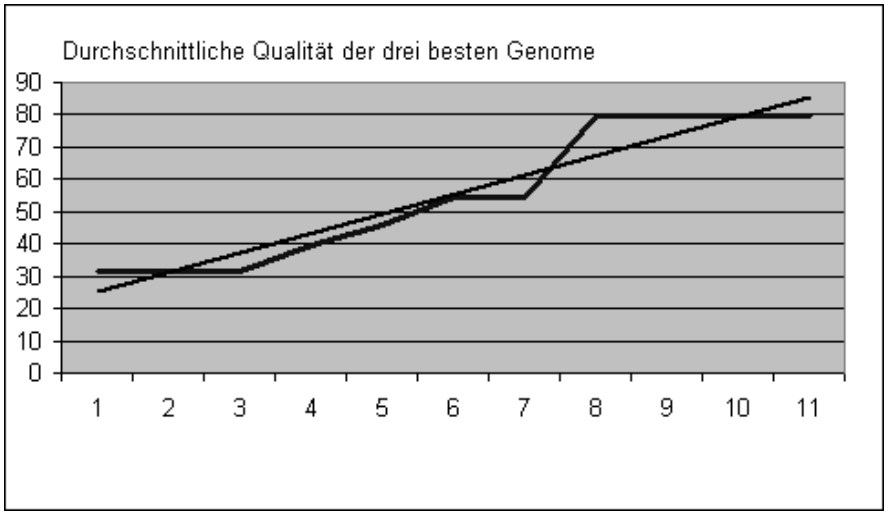
6.3.2 Checkliste

Evolutionseigenschaft	Eigenschaftsausprägung
Generation in der das beste Genom generiert wurde	8
Das beste Muster ist aus einer Kreuzung entstanden	Ja
Steigung der Qualität der Kinder	3,7
Steigung der Qualität der Besten	6,05

Eigenschaften des Suchprozesses

Graphische Analyse





Ergebnisdiskussion

Die durch die Anwendung der Subsumptions-Architekur durchgeführte Modifikation der Fitnessfunktion führt zu einem deutlich geringerem absoluten Fitnesswert der Genome. Da die Interpretationsdauer durch die Anzahl der durchgeführten Iterationen über die Regelbasis determiniert wird und die Subsumptions-Architektur die Iteration bei einer angewendeten Regel abbricht, werden im Durchschnitt weniger Erfahrungen gesammelt. Gute Verhaltensmuster, die sich die meiste Zeit über der Linie befinden, können bei einer höheren Stichprobenanzahl ihre gute Qualität durch einen höheren absoluten Qualitätswert belegen. Dies ist auf die Eigenschaft zurückzuführen, dass mit jeder interpretierten Regel ein Erfahrungswert an die Fitness übergeben wird. Um eine absolut vergleichbare Fitness zu erhalten, wäre es notwendig, die Fitness erst mit Beenden der Iteration über eine komplette Regelbasis mit einem Erfahrungswert zu speisen.

Die Steigung beider Graphen ist ähnlich wie in den ersten beiden Versuchen positiv. Obwohl der absolute Wert der Steigung auf Grund der Architekturunterschiede variert kann von vergleichbaren Versuchsergebnissen gesprochen werden.

Schlussfolgerung

Die Subsumptions-Architektur bietet bei der gewählten Qualitätsbewertung keine deutlich gesteigerte Lerneffektivität. Dies wäre erst bei einer Regel bezogenen Bewertung zu erwarten gewesen. Die Qualität eines Lernverfahrens hängt also nicht nur von der verwendeten Architektur zur Verhaltensgenerierung, sondern auch von der Fähigkeit das Verhalten entsprechend granular zu bewerten ab. Ohne dem Lernverfahren eine entsprechende Methode zur detaillierten Bewertung und damit zu einer spezielleren Modifikation zur Verfügung zu stellen, scheint keine Steigerung der Lerneffektivität möglich.

6.4 3. Versuch

Der dritte Versuch soll die in den ersten Versuchen festgestellte Fehlerhaftigkeit der Fitnessfunktion durch den Einsatz eines Shaftencoders korrigieren. Der Shaftencoder wird ausschließlich dazu verwendet, festzustellen, ob sich der Agent überhaupt bewegt hat. Wird ein definierter Wert von der Summe der durch den Shaftencoder gelieferten Sensordaten nicht überschritten, gilt das Muster als nicht anwendbar (siehe Kapitel 4.1.3). Zusätzlich wurde der in den ersten Versuchen als Default-Verhalten definierte Operator ! nun als *nand* Verknüpfung interpretiert,

was eine Vergrößerung des Suchraums darstellt, da bei der Interpretation als Fallback Verhalten, die Prämissen der Regel keinen Einfluss auf das Verhalten hatten und somit nicht zum effektiven Suchraum gezählt werden konnten.

6.4.1 Die Versuchsparameter

Parameter	Wert
Anzahl der Iterationen über die Regelbasis	1000
Anzahl der Generationen	10
Grösse der Populationen	10
Anzahl der Direktübernahmen	5
Wahrscheinlichkeit für Mutation	0.5
Transformationsorganisation	qualitätsorientiertes Roulette
Semantik des ! Operators	nand
Shaftencoder	Ja
Subsumption	Ja

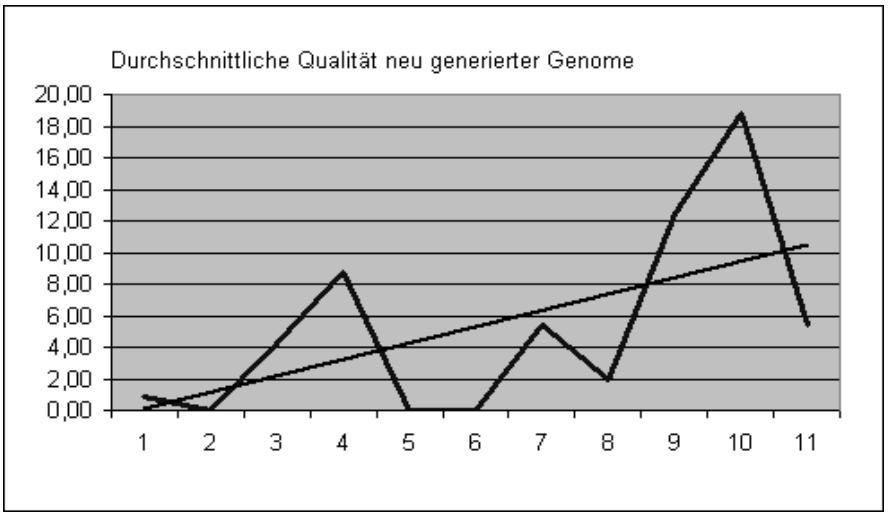
Konfiguration der Versuchsparameter

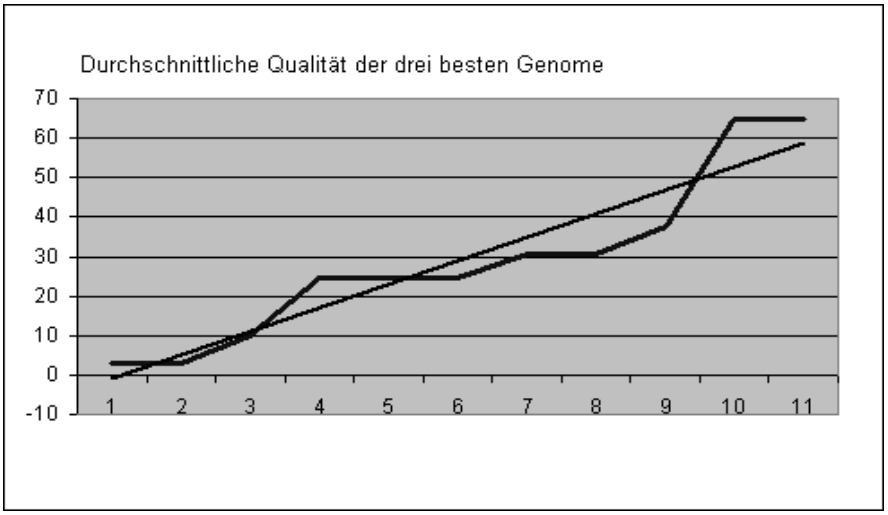
6.4.2 Checkliste

Evolutionseigenschaft	Eigenschaftsausprägung
Generation in der das beste Genom generiert wurde	10
Das beste Muster ist aus einer Kreuzung entstanden	Ja
Steigung der Qualität der Kinder	1,03
Steigung der Qualität der Besten	3,5

Eigenschaften des Suchprozesses

Graphische Analyse





Ergebnisdiskussion

Die durchschnittliche Qualität der Kinder liegt wiederum deutlich unter der des zweiten Versuchs. Die geringe Qualität kann durch den Einsatz des Shaftencoders erklärt werden. Zum einen werden Muster, die nicht das geforderte Minimum an Bewegung generieren, auf 0 gesetzt. Es ist zu beobachten, dass wesentlich mehr 0 Kandidaten in der Menge der neu generierten Genome auftauchen. Dies hat eine Reduzierung des Qualitätsdurchschnitts der Kinder zur Folge. Zum anderen fallen die hoch bewerteten aber falsch eingeschätzten Genome weg. Der Qualitätsgraph der besten drei Genome ist mit dem des zweiten Versuchs vergleichbar. Auch das ist logisch, da die erfolgreichen Muster nicht von der Shaft Encoder Regel beeinflusst werden.

Das Verfahren generierte in jedem Versuchsdurchlauf trotz der Erweiterung des Suchraumes ein anwendbares Muster. Die Problematik der falschen Fitnessfunktion wird durch die Anwendung des Shaftencoders kompensiert. Die Fitnessfunktion ist also robuster und richtiger als die der ersten beiden Versuche.

6.4.3 Schlussfolgerungen

Die erfolgreiche Korrektur der Fitnessfunktion hat den gewünschten Effekt auf die Qualität des Gesamtverfahrens gehabt. Ein Shaftencoder scheint bei der Bewertung von Vorgängen, die eine freie Bewegung des Roboters beinhalten, notwendig. Auch dieser Versuchsdurchlauf zeigt jedoch, dass das Verfahren für die Problemstellung anwendbar ist.

6.5 4. Versuch

Der vierte Versuch soll Aufschluss darüber geben, ob das Optimierungsverfahren auch mit einem reduziertem Zeitaufwand erfolgreich durchgeführt werden kann. Als Stellgröße soll hier die Dauer des Interpretationsvorgangs dienen. Entsprechend dazu soll die Anzahl der Individuen pro Population vergrößert werden, um zu überprüfen ob sich eine Erhöhung der Varianz erfolgreicher Genome positiv auswirken (siehe Kapitel 4.2.2 auf Seite 42) .

6.5.1 Die Versuchsparameter

Parameter	Wert
Anzahl der Iterationen über die Regelbasis	700
Anzahl der Generationen	10
Grösse der Populationen	20
Anzahl der Direktübernahmen	5
Wahrscheinlichkeit für Mutation	0.5
Transformationsorganisation	Die besten 5 generieren 5 Kinder
Shaftencoder	Summe der Shaftencoder muss über 50 liegen um gewertet zu werden

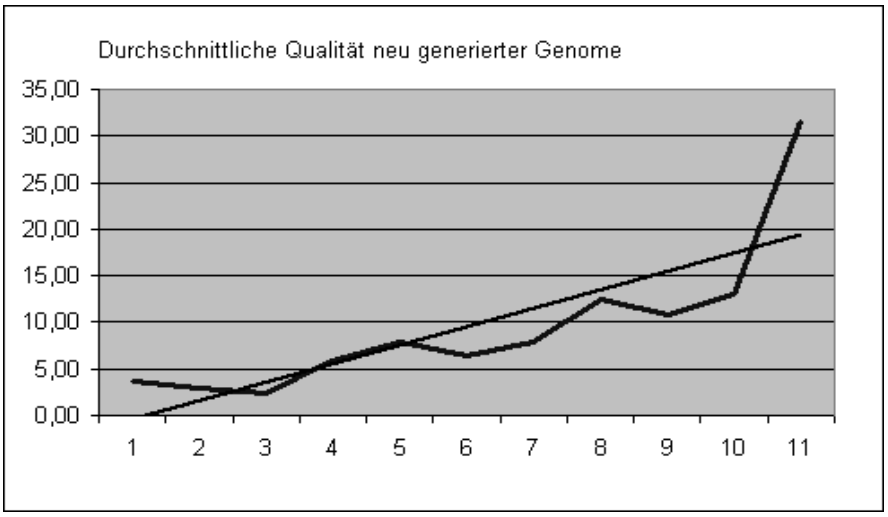
Konfiguration der Versuchsparameter

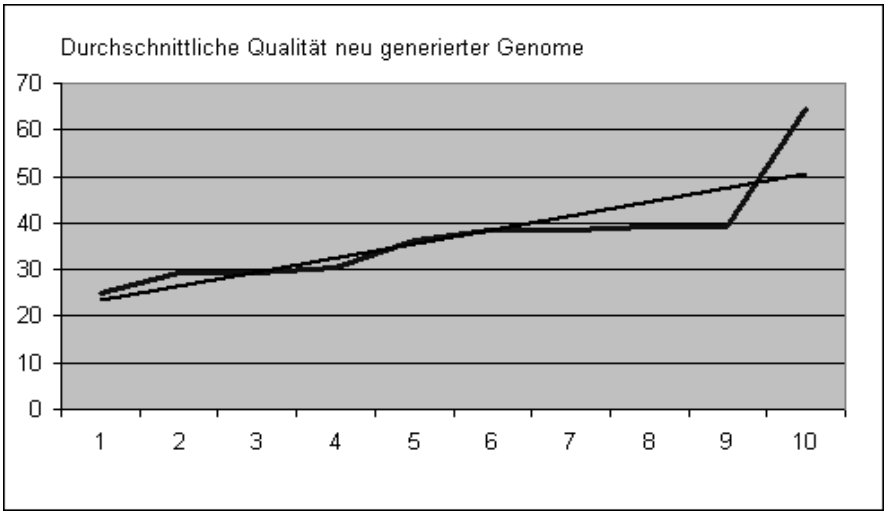
6.5.2 Checkliste

Evolutionseigenschaft	Eigenschaftsausprägung
Generation in der das beste Genom generiert wurde	9
Das beste Muster ist aus einer Kreuzung entstanden	Ja
Steigung der Qualität der Kinder	1,9
Steigung der Qualität der Besten	3,0

Eigenschaften des Suchprozesses

Graphische Analyse





Ergebnisdiskussion

Auch hier zeigen sich die bei den vorhergehenden Versuchen beobachteten Eigenschaften der Kurven. Das Verfahren scheint auf den ersten Blick erfolgreich zu sein. Das generierte beste Muster löst das Problem jedoch nicht. Die Fitnessfunktion arbeitet durch die Reduktion der Interpretationsdauer nicht mehr korrekt. Das beste Pattern generiert ein einfaches sich um die eigene Achse drehen. Der Grund, warum dieses Pattern sich gegenüber möglichem Linefollow Verhalten durchsetzen kann, liegt in der Tatsache, dass das Linefollow Problem nicht genug Zeit hat, um seine Qualität auszuspielen. Viele der Linefollow Muster fahren eine weiche Links oder Rechtskurve, wenn sie auf eine weiße Fläche stoßen. Steht der Roboter in der Ausgangskonfiguration ungünstig, dauert es einen gewissen Zeitraum, bis der Roboter in einem Winkel auf die Linie trifft, die ein sauberes Verfolgen der Linie ermöglicht (siehe Kapitel 4.1.3 auf Seite 37). Da die Interpretation bei der gewählten Versuchsdauer jedoch schon vorher abbricht, sind die drehenden Muster bei dieser Fitnessfunktion im Vorteil. Dies belegt, dass eine statistisch ausreichende Stichprobenzahl (also die Dauer der Interpretation) mit zu der korrekten Formalisierung der Qualität eines durch einen Agenten generierten Verhaltens gehört.

Aus verfahrenstechnischer Sicht hat die Vergrösserung der Population dazu geführt, das der Qualitätsgraph der Kinder keinen Schnittpunkt mit der X-Achse mehr besitzt. Durch die Größe der Population ist die Wahrscheinlichkeit mindestens ein Muster das sich bewegt und nicht gegen eine Arealbegrenzung fährt zu generieren gestiegen.

6.5.3 Schlussfolgerungen

Der Versuch zeigt, dass eine fehlerhafte Formalisierung der Lernaufgabe nicht anhand der Effizienz des Suchprozesses bemerkt werden kann. Obwohl der Optimierungsvorgang an sich richtig und erfolgreich ist, wird in dieser Versuchskonfiguration am Ziel vorbei gelernt. Würde die Interpretationsdauer weiter reduziert, so würde der Qualitätswert zunehmend an Aussagekraft verlieren (von Zufall bestimmt werden) und somit auch die von ihm abhängende Lerneffektivität senken. Da in diesem Fall die Reduzierung der Interpretationsdauer eine Modifikation der Lernaufgabe implizierte, lernte das System aus algoritmischer Sicht erfolgreich.

Kapitel 7

Zusammenfassung & Aussichten

7.1 Zusammenfassung

Die vorliegende Arbeit dokumentiert die Architektur eines adaptiven, verkörperten Agentensystems. Die Verkörperung wird dabei durch einen mobilen Roboter repräsentiert. Die Adaptionsfähigkeit wird durch ein genetisches Verfahren realisiert, welches das Verhalten des Agenten hinsichtlich seines Zieles optimiert. Der verwendete Roboter basiert auf dem im Kapitel 5.1.1 näher beschriebenen *LEGO®MINDSTORMS™RoboticsInventionSystem™* und stellt damit eine kostengünstige, einfache und damit in einigen Aspekten beschränkte Plattform für die Umsetzung der Verkörperung dar.

Der Versuch zeigt, dass genetische Verfahren grundsätzlich in der Lage sind, das Verhalten mobiler Roboter auf eine Weise zu modifizieren, dass diese in der Lage sind, einfache Aufgaben zu lösen. Es zeigt sich, das das Verfahren mit einer geringen Anzahl von Selbstinspektionsvorgängen eine Lösung für das beschriebene Problem findet. Die gewählte Populationsgröße von 10 und die geringe Anzahl der Generationen (ebenfalls 10) geben dem Verfahren genug Spielraum um anwendbare Lösungen für das gegebene Problem zu entwickeln.

Es zeigt sich weiter, dass die Formalisierung der Lernaufgabe (dem Ziel des Agenten) ein Prozess ist, der mit Sorgfalt durchzuführen ist. Ein Fehler in diesem Prozess kann den Lernerfolg des Agenten mindern oder verhindern. Die meisten anderen Parameter des genetischen Verfahrens (Populationsgrößen, Selektions- und Kreuzungsstrategien) sind zwar ebenso relevant und deshalb korrekt zu wählen, beeinflussen jedoch oft eher die Performanz des Lernprozesses als seinen grundsätzlichen Erfolg.

Das dokumentierte System stellt ein in C++ implementiertes Framework für die Erstellung von adaptiven, verhaltensbasierten Agenten dar. Die Lernkomponente ist dabei durch eine definierte Schnittstelle (dem Kommunikationsmodul), vom Rest des Agenten getrennt und so austauschbar. Da alle Komponenten in C++ realisiert sind und bis auf die API's der Roboterbasis und des genetischen Optimierers nur wenige plattformabhängige Funktionen verwendet werden, sollte durch den Einsatz entsprechender Cross Compiler die Portierung auf andere Plattformen mit geringem Aufwand realisierbar sein.

7.2 Kritische Würdigung & Aussichten

Die genetischen Verfahren formalisieren die Wechselbeziehung zwischen biologischen Lebewesen und ihrer Umwelt. Der dokumentierte Versuch implementiert diese formalisierten Wechselbeziehungen auf Basis von verkörperten Agenten. Die Aufgabenstellung war in ihrer Komplexität jedoch stark begrenzt, um den Aufwand der Implementierung gering zu halten. Eine beliebige Erweiterung der Aufgabenstellung durch die Hinzunahme weiterer Sensoren, Motoren und Aktionen ist durch die zeitintensive Qualitätsbewertung begrenzt.

Das implementierte System (im speziellen die virtuelle Maschine und ihre Interpretersprache) ist für die Optimierung des Verhaltens von reaktiven Agenten ausgelegt. Eine Erweiterung um eine modellbasierte Verhaltensbildung ist zwar durchaus realisierbar, würde jedoch eine erhebliche Modifikation der entsprechenden Objekte nach sich ziehen.

Eine mögliche Erweiterung des Systems könnte die Portierung der Evolutionsmaschine auf den Roboter beinhalten. Die Infrarotkommunikation wäre dann auf das Austauschen der genetischen Informationen und ihres Qualitätswertes zwischen verschiedenen Agenten beschränkt. Die Agenten einer Population würden den eigenen Qualitätswert versenden und den der anderen Empfangen. Die Auswahl des Kreuzungspartners würde dann von den Agenten selbständig durchgeführt werden. Es würde ein verteiltes Optimierungsverfahren entstehen, bei dem das "Fortpflanzungs-Verhalten" der einzelnen Agenten den Optimierungsprozess determinierte (vergleiche Interaktionistische Wissensrepräsentation in Kapitel 2.2)

Die genetischen Verfahren stellen ein Klasse von Optimierungsverfahren dar, deren Grenzen nur schwer abschätzbar sind. Obwohl sich die Anwendungen dieser Verfahren heutzutage auf einfache, sehr spezielle Aufgaben begrenzt zeigt sich, das das Prinzip der Evolution in der Lage ist erstaunliche Adaptionsleistungen zu erbringen.

Bei einer entsprechend detailierten Modellierung der Problemstellung und einer ausreichenden Anzahl von Ressourcen (Zeit, Speicher und Prozessorleistung) kann wahrscheinlich jedes formalisierbare Problem mit einem solchen Optimierer gelöst werden, vorrausgesetzt, es steht eine Möglichkeit zur Selbstinspektion zur verfügung.

Es gilt zu prüfen, ob sich bei Einbettung eines Agenten in eine komplexe Umwelt, wie etwa natürliche Systeme sie darstellen, allgemein gültige Fitnessfunktionen für komplexe Aufgabenstellungen formulieren lassen. Im insbesonderen ist die Modellbildung, die bei genetischen Verfahren durch die Definition des Genomes und dem die Genominformation interpretierenden Interpreter gegeben ist, ein Prozess, der abhängig von der gegebenen Aufgabenstellung eine erhebliche Vorleistung des Programmierers benötigen kann.

Abbildungsverzeichnis

1 N.A.S.A. Roboter Sojourner 6

1.1 Geheimagent 007 9
1.2 Taxonomie der Agenten 12

2.1 Die Populationsüberführung und ihre Teilprozesse 20
2.2 Genominformationen in einem DNA-Strang 24
2.3 Die Urpopulation streut die Lösungsansätze 25
2.4 Das Selektionsverfahren verwirft die ”Verlierer” 26
2.5 Die selektierten Genome bilden die neue Population 27

4.1 Der Versuchsaufbau 30
4.2 Grobkonzept des Agentensystem 31
4.3 Informationsfluss in der virtuellen Maschine 32
4.4 Sequenzdiagramm der Bewertung einer Regelbasis 40
4.5 Genom des Programmes +|EEF+|NNL 42
4.6 Unterbaumkreuzung 45
4.7 Swap node Mutation 46

5.1 Der RCX Baustein 50
5.2 Die verwendete Roboterkonfiguration 51
5.3 Der Versuchsaufbau 52
5.4 Ausnahmebehandlung nach Kollision 59
5.5 Die Objekte der GAlib 63
5.6 Die 3 Nachrichten Typen 73
5.7 Sequenzdiagramm der Infrarotkommunikation 78

6.1 Stammbaum der beiden besten Genome des 1.Versuch 88

Versicherung über Selbständigkeit

Hiermit versichere ich, daß ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbständig verfaßt und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift des Studenten

Literaturverzeichnis

1 *Rover Telecommunications Link Analysis* - URL: http://mars.jpl.nasa.gov/MPF/rovercom/itworks.htmlrover-radio - Zugriffsdatum: 01.09.2002

2 *Frauenhofer Institut für Autonome Intelligente Systeme: Unsere Roboter* - URL: http://www.ais.fhg.de/ - Zugriffsdatum: 01.09.2002

3 *Robocup 2002 Homepage* - URL: http://www.robocup.org/ - Zugriffsdatum: 01.09.2002

4 OSHERENKO, Alexander: *Design und Implementierung eines Multiagentensystems zur Robotsteuerung* - URL: http://www.informatik.fh-hamburg.de/ robots/papers.html -Zugriffsdatum: 01.09.2002

5 BURKHART, H.-D. : *Software-Agenten* in: GÖRZ, G.; ROLLINGER, C.-R. ; SCHNEEBERGER, J. (Hrsg.): *Handbuch der Künstlichen Intelligenz* , S. 981 - 986. 3. Auflage, Oldenbourg Wissenschaftsverlag, München, 2000 - ISBN 3-486-25049-3

6 WROBEL, Stefan : *Maschinelles Lernen* in: GÖRZ, G.; ROLLINGER, C.-R. ; SCHNEEBERGER, J. (Hrsg.): *Handbuch der Künstlichen Intelligenz*, S. 519.3. Auflage, Oldenbourg Wissenschaftsverlag, München, 2000 - ISBN 3-486-25049-3

7 DILGER, Prof. Dr. Werner : *Repräsentation von Wissen und Annahmen* in: *Multiagentensysteme*, Vorlesungsskript zum Wintersemester 2001/2002, S. 5. Technische Universität Chemnitz

8 BROOKS,R.A. *Intelligence without reason* in: *Proceedings IJCAI-91* , S. 569-595, 1991.

9 BROOKS, R. A. (1991) *Intelligence without representation* in: *Artificial Intellgence* , S. 139-159, 1991.

10 VOLMER, Prof. Dr. Reinhard : *Entwicklung und Anwendung von Künstlichen Neuronalen Netzen* - URL: http://www.fh-muenster.de/ZWEI/knn/knn_96_3.html - Zugriffsdatum: 01.09.2002

11 BROOKS, R.A.: *How to build complete creatures rather than isolated cognitive simulators* in: K. VanLehn (ed.): *Architectures for Intelligence*, S. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991.

12 BAUM, Stephen; GASPERL, Michael; HEMPEL, Ralph; VILLA, Luis: *Extreme Mindstorms(an Advanced Guide to Lego Mindstorms). Berkley (CA), USA: APress 2000. - ISBN 1-893115-84-4*

13 *The C++ Resources Network* - URL: http://www.cplusplus.com - Zugriffsdatum: 01.09.2002

14 WALL, Matthew : *A C++ Library of Genetic Algorithm Components* - URL: http://lancet.mit.edu/ga/ - Zugriffsdatum: 01.09.2002

15 LugNet Community *RCX Programming* - URL: http://news.lugnet.com/robotics/rcx/ - Zugriffsdatum: 01.09.2002

16 ASH, Mike : *LegOS Network Protocol* - URL: http://legos.sourceforge.net/HOWTO/x405.html - Zugriffsdatum: 01.09.2002

Anhang A

Versuchs Logs

Die vom genetischen Optimierer generierten Logdateien stellen die Grundlage der Verfahrensanalyse dieser Arbeit dar. Die den Versuchsdurchführungen zugeordneten Logdateien befinden sich im *logfiles* Verzeichnis auf der beigefügten CD. Die Logdateien dokumentieren die in den Populationen enthaltenen Individuen, die ihnen zugeordnete Qualität sowie die vom Optimierer durchgeführten Transformationsvorgänge. Das folgende Beispiel dokumentiert die dabei verwendete Strukturierung der Logdateien:

Die Regelbasen einer Population werden bewertet, wobei der hinter der -› Zeichenkette stehende Wert die Fitness des Musters darstellt:

Test new Pattern : +|EE›+&EE‹+!EE‹+ -› 0
Test new Pattern : +|NE‹+&EEF+|EE‹+ -› 20
Test new Pattern : +|EEL+&EN‹+&EE›+ -› 12
Test new Pattern : +|EE‹+!NEL+|NN‹+ -› 5
Test new Pattern : +|ENL+!NNL+&EN‹+ -› 35
Test new Pattern : +&EN›+|NE›+|EE›+ -› 0
Test new Pattern : +|NNR+&EEL+&NER+ -› 65
Test new Pattern : +!EE‹+&EE›+!NEF+ -› 0
Test new Pattern : +|EEF+&ENF+|NE›+ -› 0
Test new Pattern : +&EE‹+|NER+|NN‹+ -› 51

Die Erstellung der Generation mit dem am Ende des Satzes spezifizierten Index wird durchgeführt.

forging next Generation which is number: 0

Die Transformation der zur Kreuzung herangezogenen Genome wird durchgeführt

++
Crossover initialized
++

Die beiden zur Kreuzung verwendeten Individuuen werden dokumentiert.

****
Individual A: +|EE‹+!NEL+|NN‹+
****
Individual B: +|NNR+&EEL+&NER+
****

Die zur Kreuzung verwenden Knoten der beiden Individuuen werden dokumentiert

* Node 10 with 5
****

Die Mutation der durch die Kreuzung entstandenen Kinder wird dokumentiert

****
Mutating childs of Crossover
****
Child: +|EE‹+!NEL+&EEL+&NER+
****

* swapping from 9to9

* swapping from 0to15

* swapping from 13to13

* swapping from 13to8

* swapping from 14to14
****

Anhang B

Sourcecode

Der Sourcecode des Implementierten Systems, ist in der beigefügten CD enthalten. Das Verzeichniss *sourcecode* enthält die Unterverzeichnisse *virtuelleMaschine* und *evolutionsMaschine* , indem die entsprechenden Sourcecode-Dateien enthalten sind.

Die Dateien können nur im Zusammenhang mit der entsprechenden Entwicklungsumgebung kompiliert werden. Das bedeutet, das die Dateien des Verzeichniss *virtuelleMaschine* eine funktionierende *LegOs* Entwicklungsumgebung und die Dateien des *evolutionsMaschine* Verzeichniss die Bibliotheken der GAlib benötigen. Für nähere Informationen über die Installation dieser Fremdkomponenten sei auf die auf der CD enthaltene Dokumentation verwiesen.