Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Timo Storjohann

Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Timo Storjohann

Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Gunter Klemke

Abgegeben am 23. August 2005

Timo Storjohann

Thema der Diplomarbeit

Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters

Stichworte

Maschinelles Lernen, genetische Algorithmen, mobiler Roboter, Simulator

Kurzzusammenfassung

Die vorliegende Arbeit dokumentiert die Erstellung einer lernfähigen Software, die zur Steuerung eines mobilen Roboters eingesetzt wird. Die Adaptionsfähigkeit, in Hinsicht auf das zu erlernende Ziel, wurde dabei mit Hilfe eines genetischen Verfahrens realisiert und durch Versuchsreihen auf ihre Tauglichkeit überprüft. Zur Ermittlung der evolutionären Rahmenparameter wurde im Vorfeld ein Simulatorsoftware erstellt.

Timo Storjohann

Title of the paper

Application of genetic learning methods for the programming of a mobile robot

Keywords

Machine based learning, genetic algorithm, mobile robots, simulation

Abstract

This thesis documents the development of an adaptive software, used for controlling of a mobile robot. The ability to adapt was realized with genetic learning methods and examined for its fitness by test series.

To determine the evolutionary parameters a simulator software was created.

Danksagung

Prof. Dr. Kai von Luck für sein Engagement, seine Unterstützung und die Leidenschaft für seine Arbeit.

Prof. Dr. Gunter Klemke für seine Bereitschaft auch am Freitag Abend noch Kolloquien abzunehmen.

Lars Brandt für die Unterstützung bei der Bluetooth Anbindung des Roboters und das engagierte Zusammenlöten von kleinen Teilen.

Gunther Lemm für den nächtlichen Linux Support.

Ingo Boersch von der FH Brandenburg, für seine Unterstützung bei der RVISION-Plattform und für viele interessante Anregungen.

INHALTSVERZEICHNIS

1 EINLEITUNG

Computertechnologie ist mittlerweile in fast allen Bereichen unseres Lebens vorzufinden. Für immer wiederkehrende und gleichbleibende Aufgaben können mit den traditionellen Programmiertechniken bereits gute Ergebnisse erzielt werden. So gehören Robotersteuerungen mit festen Abläufen, wie sie z.B. in der Fertigungsindustrie eingesetzt werden, zum Alltag. Der Wunsch des Menschen immer komplexere Aufgaben auf Maschinen zu übertragen, fordert jedoch neue Ansätze zur Programmierung. Was sind solche komplexeren Aufgaben?

Zu den komplexeren Aufgaben gehören z.B. Systeme, die sich selbstständig an sich verändernde Umweltbedingungen anpassen müssen, oder Systeme, von denen eine Interaktion mit der Umwelt gefordert wird. Für diese Aufgaben muss ein System eine adaptive Leistung erbringen können, um sich in veränderbaren Bezugssystem zurechtzufinden.

Eine konkrete Anwendung für diese Art von Aufgaben sind mobile autonome Roboter. Überall auf der Welt wird an diesen Systemen geforscht. Hier gibt es mittlerweile eine Vielzahl von Lösungen, die von eher einfachen Aufgaben – wie Hindernisse umgehen, Landkarten eines Raumes zeichnen oder einem Licht folgen – bis zu Staubsaugerrobotern, Roboterhunden mit Interaktions-möglichkeiten, oder Roboter die auf zwei Beinen laufen reichen. Interessant ist hierbei zu beobachten, dass sich immer mehr dieser Systeme aus den Forschungslabors und Universitäten in den kommerziellen Handel bewegen und dort zu erschwinglichen Preisen gekauft werden können. Hier ist besonders der Trend zu Unterhaltungsrobotern und Staubsaugerrobotern zu beobachten. Zu den bekanntesten bereits im Handel verfügbaren Systemen gehören hier wohl die Produkte von Sony:

Abbildung 1.1: Sony Roboterhund Aibo

(Quelle: [SONY AIBO])

Der Roboterhund Aibo beherrscht bereits eine Gesichts- und Stimmenerkennung und kann einfache Kunststücke auf Befehl ausführen. Der bipedale Roboter QRIO stammt ebenfalls aus dem Hause Sony und ist in der Lage sich auf zwei Beinen fortzubewegen. Beide Systeme sind als reine Unterhaltungsroboter konzipiert.

Abbildung 1.2: Sony QRIO
(Quelle: [SONY QRIO])

Der Roboter „Sir Arthur“ wurde von Frank Kirchner, der als Wissenschaftler im GMD-Institut für Systementwurfstechnik forscht, entwickelt. Die sechs Beine des Roboters sind paarweise über drei in der Mitte befindliche Segmente verbunden. Dadurch besitzt „Sir Arthur“ 16 Freiheitsgrade, die durch eine Software koordiniert werden müssen. Aufgrund der Komplexität wurde die Bewegungskoordination mit Hilfe eines autonomen, selbstlernenden Systems vorgenommen. Das Lernverfahren basiert auf einer hierarchischen Version des Q-Learnings (HQL).

Abbildung 1.3: Sir Arthur – Sechsbeiniger Roboter

Der von Michael Manger im Rahmen einer Diplomarbeit entwickelte Fußballroboter hat den Anstoß für diese Arbeit gegeben. Die Besonderheit dieses Roboters liegt in seinem omnidirektionalen Antrieb. Durch diese Antriebsart ist der Roboter in der Lage sich frei in alle Richtungen zu bewegen, ohne sich dabei um die eigene Achse drehen zu müssen. Dies wird durch drei Antriebsrollen ermöglicht, die gleichzeitig auch seitliche Bewegungen zulassen.

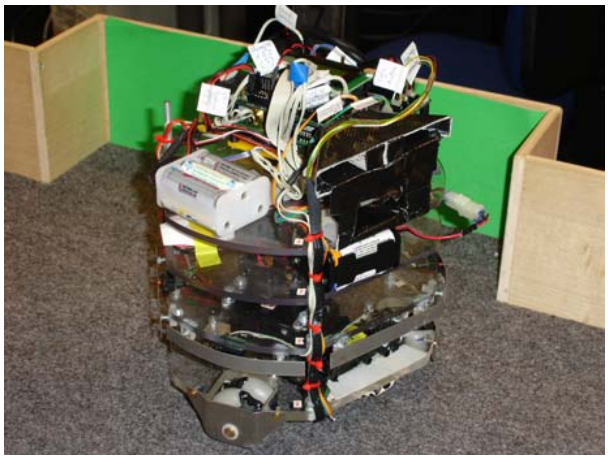


Abbildung 1.4: Omnidirektionaler Roboter - M. Manger

Erste Ansätze zur Steuerung des omnidirektionalen Antriebs des Roboters wurden von Michael Manger bereits programmiert.

Da die besondere Konstruktion des Antriebs eine handkodierte Programmierung sehr komplex gestaltet, kam die Idee auf, bestimmte Aufgaben mit Hilfe maschinellen Lernens zu ermöglichen. Die Aufgabestellung dieser Arbeit sieht vor, dass der Roboter selbstständig lernt sich auf eine Linie mit einem Ball und dem Zieltor zu begeben.

Anhand dieser Beispiele ist gut zu erkennen, dass eine fest vorgegebene Abfolge von Instruktionen keine befriedigenden Ergebnisse erzielen würde, oder zumindest mit einem programmiertechnisch hohen Aufwand verbunden wäre. Für diese Art von Aufgaben ist es wünschenswert, dass das System in der Lage ist selbstständig zu lernen, um so auf intelligente Weise mit der Umwelt interagieren zu können.

Für das maschinelle Lernen gibt es verschiedene Ansätze und Methoden. Dabei zeichnen sich die unterschiedlichen Methoden durch ihre verschiedenen Fähigkeiten aus. Zum einen bestehen zwischen den Verfahren Unterschiede, was die Komplexität des Wissens das gelernt werden kann angeht, zum anderen wie gut eine Methode in der Lage ist unscharfe oder ungenaue Umweltdaten zu handhaben. Zu den bekanntesten Lernverfahren gehören z.B. die Neuronalen Netze, die sich am menschlichen Gehirn orientieren und versuchen dieses in einer stark vereinfachten Form nachzuahmen. Ein weiteres Verfahren ist die

Induktion von Entscheidungsbäumen. Hier werden einfache Regeln gelernt, die in Form von Konjunktion von Tatsachen ausgedrückt werden. Eine weitere Lernmethode sind die genetischen Verfahren. Diese können nicht nur auf die Probleme des maschinellen Lernens angewendet werden, sondern auch auf viele andere Suchprobleme. Der Ansatz dieser Lernverfahren lehnt sich an die Darwinsche Evolutionstheorie und dem Prinzip des „survival of the fittest“ an. Von Generation zu Generation passen die Individuen eines Systems sich ihrer Umwelt an und verbessern so Ihre Überlebenschancen. Wie in der Darwinschen Theorie, werden die Mechanismen der Vererbung, Mutation und Selektion auf Computerprogramme angewendet, um bessere Lösungen zu finden. Die vorliegende Arbeit beschäftigt sich mit den verschiedenen genetischen Verfahren. Hier soll insbesondere die Frage geklärt werden, ob eine Steuerung des Roboters mit Hilfe genetischer Algorithmen möglich ist und ob komplexes Verhalten mit diesen Methoden erlernbar ist.

Im Folgenden wird nach einer kurzen Vorstellung der technischen Voraussetzungen der Roboterplattform eine Einführung in die Theorie der genetischen Lernverfahren gegeben. Dabei wird evaluiert, welche Verfahren für die gegebene Aufgabenstellung am besten geeignet sind. Darauf basierend wird dann das Grunddesign der zu erstellenden Software und die zugrundeliegenden Modelle entwickelt. Zur Vorbereitung des eigentlichen Roboterversuchs, wird im Vorfeld eine stark vereinfachte Umweltsimulation, zur Ermittlung der evolutionären Rahmenparameter, erstellt werden. Im letzten Teil folgt der eigentliche Roboterversuch, in welchem die weiterentwickelte und an die Hardware angepasste Simulatorsoftware, mit den gewonnen Rahmenparametern zu Anwendung kommt.

2 DIE PROBLEMSTELLUNG UND ZIELSETZUNG

Mit Hilfe eines genetischen Verfahrens soll ein autonomer Roboter mit omnidirektionalem Antrieb in die Lage versetzt werden, selbständig eine vorgegebene Choreografie zu erlernen. Die Umwelt des Roboters besteht dabei aus einer begrenzten, rechteckigen Fläche, an deren Stirnseiten eine rote und ein grüne Fläche (die Tore) montiert ist. Auf der „Spielfläche“ befindet sich ein Infrarotball. Die Aufgabe des Roboters soll es nun sein, sich in möglichst kurzer Zeit auf einer Linie mit dem Ball und einer farbigen Stirnseite (dem Tor) zu positionieren. Die Startposition des Roboters wird dabei zufällig gewählt.

Die elementare Frage ist, lässt sich mit Hilfe eines genetischen Verfahrens eine geeignete Lösung zur Steuerung des Roboters generieren? Dies soll auf experimentellem Wege herausgefunden werden.

3 TECHNISCHE VORAUSSETZUNGEN

In diesem Kapitel möchte ich die technischen Vorraussetzungen, d.h. die verwendete Hard- und Software näher vorstellen.

3.1 Roboterplattform

Wie bereits in der Einleitung erwähnt, wird der Versuch auf der von Michael Manger entwickelten Roboterhardware durchgeführt. Zur Erkennung der Umwelt verfügt der Roboter über verschiedene Sensoren:

- Sharp Infrarot Sensoren zur Abstandsmessung
- Microschalter für das Bumpersystem
- Beaconsensoren zur Erkennung der Torbarken
- Infrarot Ballsensoren zur Erkennung eines IR-Balles

Der Aufbau des Roboters teilt sich in vier Ebenen auf.

Auf der untersten Ebene befindet sich der omnidirektionale Antrieb, der sich aus 3 Getriebemotoren und den Antriebsrädern zusammensetzt. Die Grundform des Roboters beschreibt ein gleichmäßiges Sechseck, mit einem „Durchmesser“ von 22 cm. Diese Bauform ermöglicht eine extreme Wendigkeit, d.h. der Roboter ist in der Lage jede Himmelsrichtung ohne vorheriges Drehen anzufahren.

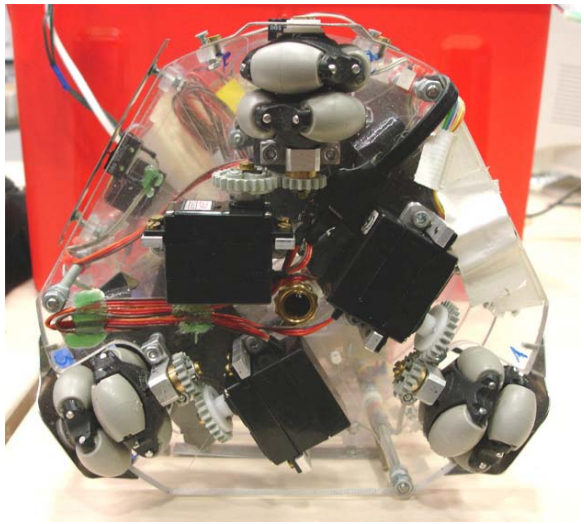


Abbildung 3.1: Omnidirektionale Antriebseinheit

Auf der zweiten Ebene befinden sich zwei Infrarot-Ballsensoren, die über einen Servomotor bewegt werden können. Die Sensoren sind, wie auf der Abbildung zu erkennen, auf einem beweglichen Arm angebracht. Der Einsatz von zwei Sensoren ermöglicht ein schnelleres Erkennen des Infrarotballes, da jeder Sensor mit einer Drehung von lediglich 180° den gesamten Raum abdeckt. Die Sensoren erkennen einfache Infrarotstrahlung, wie sie von den speziellen Infrarotbällen der Firma Wiltronics ausgestrahlt werden. Um die Position des Infrarotballs möglichst genau feststellen zu können, ist der Einlass für das Licht durch zwei Röhrchen begrenzt.

Abbildung 3.2: IR-Ballerkennungssensor

Der Roboball MK2 der Firma Wiltronics besteht aus einem Kunststoffkugelgehäuse, in dem einfache Infrarot-LEDs verbaut sind. Der Roboball hat einen Durchmesser von 8 cm und wiegt 105 g. Der Ball wird durch eine wiederaufladbare Batterie betrieben, die ca. für 25 - 30 Minuten Betriebszeit ausreicht.

Abbildung 3.3: IR-Ball Fa. Wiltronics

Auf der dritten Ebene sind 4 Infrarot-Abstandssensoren[1] auf einer beweglichen Sensorplattform installiert. Dabei decken drei Sensoren den vorderen Bereich des Scanners ab und ein Sensor gleichzeitig den hinteren Bereich. Die Sensoren arbeiten mit einer Infrarotabstandsmessung, die im Bereich von 8 - 60 cm zuverlässig Objekte und Hindernisse erkennt. Die von der Firma Sharp produzierten Sensoren geben einen zur gemessenen Entfernung proportionalen Spannungspegel aus. Für den Fall, dass die Abstandsmessung doch einmal versagen sollte, ist auf der gleichen Ebene ein rundum laufendes Bumpersystem, bestehend aus 6 Mikroschaltern installiert. So kann im Notfall eine Kollision festgestellt werden.

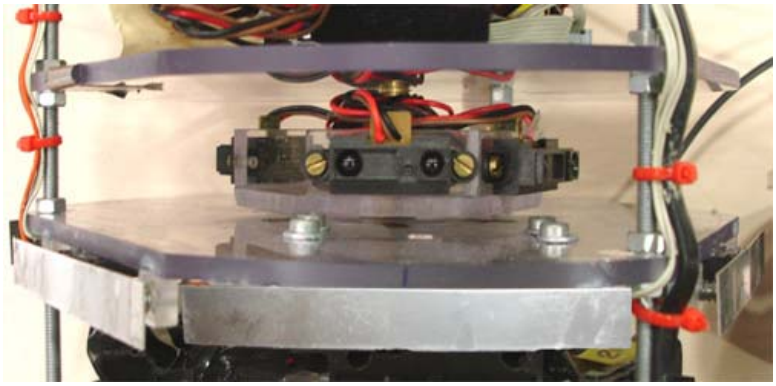


Abbildung 3.4: IR-Abstandsmessung - Sensoren Fa. Sharp

Auf der vierten und letzten Ebene befindet sich das RVISION System zur Steuerung des Roboters, sowie ein Sensor zur Torerkennung. Der Torerkennungssensor arbeitet ebenfalls mit Infrarotstrahlung. Spezielle Torbarken senden auf Infrarotbasis ein 40 kHz Signal, mit aufmodulierten 100 Hz oder 125 Hz aus. Eine Elektronik im Torsensor wertet die 40 kHz Frequenzen aus und gibt die aufmodulierten Impulse an das Aksen-Board weiter. So kann zwischen den zwei Torbarken unterschieden werden. Der Sensor besteht aus drei Infrarot-Dioden, mit einer nachgeschalteten Elektronik zur Auswertung der 40 kHz Frequenz. Jede der Dioden liefert ein Signal an das Controllerboard. So ist eine relativ gute Messung des Winkels zwischen Roboterfront und zu erkennendem Tor möglich. Im Gegensatz zu den beiden äußeren IR-Empfangsdioden, ist die mittlere Diode mit einer verkleinerten Lichteinfallsöffnung ausgestattet. Der dadurch resultierende verkleinerte Erkennungskegel des mittleren Sensors, macht genauere Aussagen über die Stellung des zu erkennenden Tores möglich. Der mittlere Sensor reagiert dadurch hauptsächlich bei einer direkten Gegenüberstellung zum Zieltor. Jeweils eine Torbake ist über den Toren positioniert, um eine Unterscheidung zu ermöglichen.

[1] Der Motor zum Antrieb der Sharpsensorplattform wurde von mir durch einen Servomotor ersetzt, um genauere Angabe über die Sensorstellung machen zu können. Die Stellung des Sensors kann jetzt gradgenau gemessen werden.

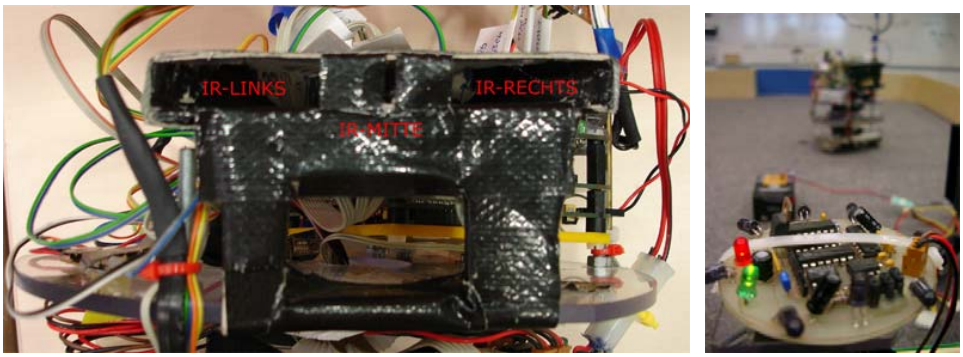


Abbildung 3.5: Links: IR-Torerkennungssensoren - Rechts: IR-Torbarke

3.2 Controllersystem RVISION

Zur Steuerung des Roboters wird das an der Universität Brandenburg entwickelte RVISION System[2] verwendet. Das RVISION System setzt sich aus drei Hauptplatinen zusammen.

- Aksen-Board
- VIO-Board
- CPU-Board

Wie im Bild zu erkennen, sind die Komponenten übereinander verbaut. Ganz oben befindet sich das Aksen-Board zur Ansteuerung der Sensorik und Aktorik. Darunter sitzt das VIO-Board für die Bildaufnahme mittels Kamera. Den Abschluss bildet das leistungsfähige CPU-Board, quasi das Gehirn des Systems, welches leistungsfähig genug ist, eine Bildverarbeitung durchzuführen. Das CPU- und das VIO-Board bilden zusammen das sogenannte VIO-Modul. Das Aksen-Board und das VIO-Modul sind über einen CAN-Bus miteinander verbunden.

[2] Das ursprünglich verwendete MIT 6.270 Board wurde durch die leistungsfähigere RVISION Plattform ersetzt.

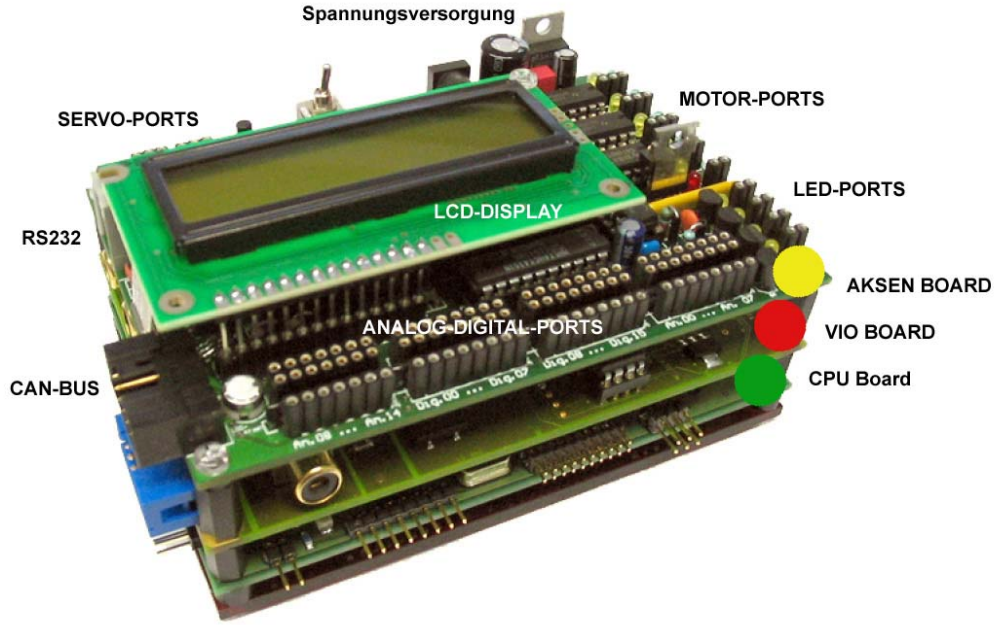


Abbildung 3.6: Controllersystem - RVISION

Wie in der Abbildung zu erkennen, ist das Aksen-Board mit diversen Schnittstellen ausgestattet. Für die Motoransteuerung stehen vier Standard-Motorports, die mit jeweils einem Ampere belastet werden können, zur Verfügung. Des weiteren verfügt das Board über vier spezielle Servo-Ports, die mit Hilfe einer Pulsweitenmodulation gesteuert werden können. Für den Anschluss diverser Sensoren stehen sowohl analoge, als auch digitale Ein- und Ausgänge zur Verfügung. Ein 8-Bit Mikrokontroller aus der 8051-Familie stellt das Herz des Aksen-Boardes dar, der trotz seiner relativ geringen Leistung ein rudimentäres Betriebssystem mit Multitaskingeigenschaften ermöglicht.

Die Fähigkeiten des VIO-Boards werden nicht benutzt, da keine Bilderkennung für das Experiment vorgesehen ist. Eine Torerkennung auf Basis der von einer Kamera aufgenommenen Bilder wird zur Zeit von meinem Kollegen Lars Brandt im Rahmen seiner Diplomarbeit [BRANDT 2005] untersucht. Der Vollständigkeit halber möchte ich trotzdem kurz die Fähigkeiten des VIO-Boards vorstellen, da sie Teil des RVISION Systems ist.

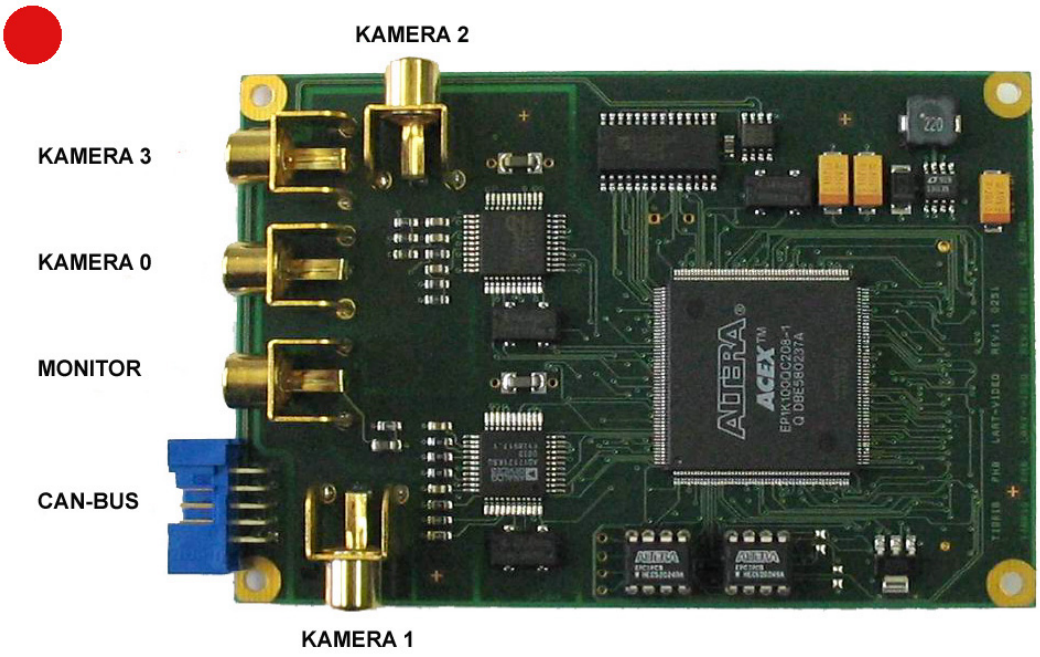


Abbildung 3.7: VIO-Board

An das VIO-Board können maximal vier PAL Kameras angeschlossen werden. Eine C-Mos Kamera liegt dem RVISION System standardmäßig bei. Zwischen den Kameraquellen kann im Multiplexingverfahren, mit einer Umschaltzeit von 40 Millisekunden, gewechselt werden. Das Board dient hauptsächlich zur Verarbeitung der Kamerasignale und kann im weitesten Sinne als Digitalisierer für die angeschlossenen Bildquellen betrachtet werden. Die gewonnenen Bilddaten werden über eine spezielle Schnittstelle dem CPU Board zur Weiterverarbeitung zur Verfügung gestellt. Für Kontrollzwecke, können die aktuellen Kamerabilder auch über einen vorhandenen F-Bas Monitoranschluss ausgegeben werden.

Die Bildverarbeitung findet auf dem CPU-Board statt. Ein leistungsstarker StrongARM SA1100 Prozessor, wie er auch in PDA´s Verwendung findet, sorgt dabei für ausreichend Rechenleistung. Das RVISION System bietet somit umfangreiche Möglichkeiten für die Steuerung reaktiver Systeme.

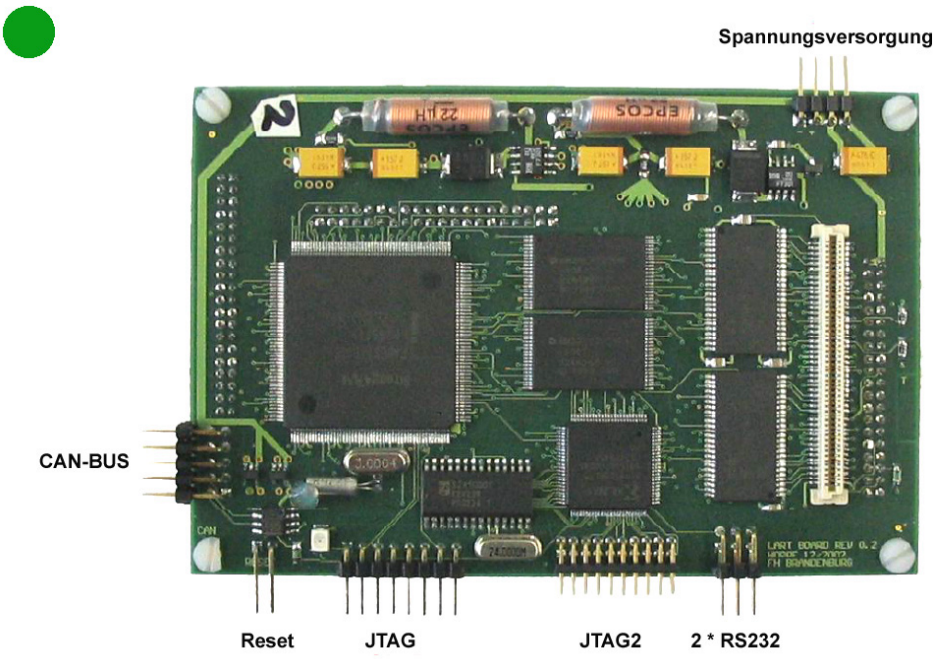


Abbildung 3.8: CPU-Board

Das CPU-Board verfügt für die Programmierung über zwei serielle Schnittstellen, die über ein Terminalprogramm bedient werden können. Ein auf dem Board installiertes embedded Linux dient als Betriebssystem. Das Linux wird beim Einschalten des Boards in einen flüchtigen Speicher geladen und ausgeführt. Die mit einem Small Devices C-Compiler generierten Programme können auf eine Ramdisk, die von Linux verwaltet wird, übertragen und gestartet werden. Da es sich bei der Ramdisk ebenfalls um einen flüchtigen Speicher handelt, muss für die dauerhafte Speicherung von Daten auf die serielle Schnittstelle für eine Datenübertragung zurückgegriffen werden. Ein bereits in der Entwicklung begriffenes Speichermodul auf Compact Flash Basis stand zum Zeitpunkt dieser Diplomarbeit noch nicht zur Verfügung. Die Stromversorgung aller Boards wird zentral über einen Anschluss, der sich auf dem Aksen-Board befindet, vorgenommen.

3.3 Software

Für die Erstellung der benötigten Software stehen im HAW Roboterlabor verschiedene Tools zur Verfügung, die unter zwei zur Auswahl stehenden Betriebssystemen benutzt werden können. Für die geplante Simulatorsoftware soll die unter Windows XP installierte Entwicklungsumgebung Microsoft Visual Studio .NET 2003 zum Einsatz kommen. Hierbei soll der in der Entwicklungsumgebung verfügbare C-Compiler und der integrierte Editor benutzt werden.

Die Programmierung des RCUBE-Systems soll mit dem unter der Suse 9 Linux Distribution verfügbaren Small Device C-Compiler [SDCC] stattfinden. Hierbei ermöglichen speziell an die RCUBE-Architektur angepasste Bibliotheken die Programmierung der Zielplattform. Zum Erstellen des C-Codes kommt der frei verfügbare Jedit Editor [JEDIT] zum Einsatz. Die für das Aksen-Board und das Lart-Board zu erstellenden Programme können unter Linux kompiliert und von dort direkt auf die Zielsysteme übertragen werden. Das Aksen-Board wird über ein spezielles Flashtool mit den erstellten Programmen beschrieben. Das Lart-Board, auf dem ein angepasstes Linux läuft, lässt sich über ein Terminalprogramm ansprechen. Die erstellten Programme können mit Hilfe eines Terminalzugangs in den Speicher des Lart-Boards übertragen und dort ausgeführt werden. Die Verbindung kommt in beiden Fällen über die, auf den Boards installierten, seriellen Ports zu Stande.

4 GENETISCHE LERNVERFAHREN

4.1 Einführung

Die genetischen Lernverfahren sind an die Darwinsche Evolutionstheorie angelehnt und greifen die von Darwin beobachteten Mechanismen auf. In der Natur überleben hauptsächlich diejenigen biologischen Individuen, die sich erfolgreich an Ihre Umgebung anpassen können. Diese Individuen können sich folglich auch am häufigsten reproduzieren („survival of the fittest“).

In der Biologie spricht man von natürlicher Selektion. Individuen, die sich nicht oder nur unzureichend an ihre Umwelt anpassen können, sterben aus. Die überlebensfähigen Individuen kreuzen Ihr Erbmaterial durch (sexuelle) Rekombination und erzeugen so neues, evtl. verbessertes Erbmaterial. Das zusätzliche gelegentliche Auftreten von Mutationen des Erbmaterials verhilft der Natur zu einer erfolgreichen Methode biologische Individuen an ihre Umwelt anzupassen.

Eine elementare Frage im Bereich des maschinellen Lernens ist: Wie kann ein Computer selbstständig lernen Probleme zu lösen, ohne ihn expliziert für das Problem programmiert zu haben? Ein Ansatz solche Probleme zu lösen, sind die genetischen Algorithmen und die erweiterte Untermenge, die genetische Programmierung.

Die Methoden der Selektion, Kreuzung und Mutation werden bei den genetischen Verfahren analog zum biologischen Vorbild zur Lösungssuche eingesetzt. Der Erfolg eines Individuums kann in der Natur anhand seiner Fitness gemessen werden, also der Anpassungs- und somit Überlebensfähigkeit in einer gegebenen Umgebung. Die Fitness eines Individuums spielt nicht nur in biologischer Hinsicht eine zentrale Rolle, da sie ja direkt das Überleben des Individuums sichert, sie spielt auch eine zentrale Rolle bei den genetischen Algorithmen, da sie das Qualitätskriterium des laufenden Prozesses darstellt. Um die verwendeten Datenstrukturen und dynamischen Prozesse der genetischen Verfahren zu beschreiben, hat man sich bewusst der Begriffe des darwinschen Evolutionsmodells bedient.

4.2 Die Grundidee

Wie läuft ein genetischer Algorithmus ab? Die Grundidee ist, eine Anzahl von zufällig erzeugten Symbolketten zu bilden und diese durch einen Interpreter in eine Folge von Aktionen umzusetzen, wobei der Interpreter die erzeugten Ketten

nach den Eingabesymbolen durchsucht und die entsprechenden Aktionen ausführt. Die Aktionsfolgen, die sich aus jedem Genom und den Eingabesymbolen ergeben, werden hinsichtlich ihrer Fitness bewertet und durch die Methoden der Selektion, Kreuzung und Mutation in eine neue Population überführt.

Dabei werden hauptsächlich die Symbolketten mit einer eher hohen Qualität in die neue Population überführt. Die Fitness stellt hierbei die Bewertung der Aktionsfolge dar. Wie gut also die Abfolge der Aktionen der Aufgabenstellung entspricht. Mit der neu entstandenen Population von Genomen wird der Vorgang so lange wiederholt, bis eine gewünschte Qualität erreicht ist, oder eine bestimmte Anzahl von Generationen erzeugt wurde.

4.3 Das Evolutionsmodell

Im Folgenden werden die wichtigsten Begriffe der Evolutionstheorie und Ihre Bedeutung für die genetischen Algorithmen näher erklärt.

4.3.1 Das Genom

Aus biologischer Sicht:

Das Genom enthält alle Informationen zur Entwicklung eines Lebewesens. In ihm sind die gesamten Bau- und Leistungsmerkmale in Form von Basensequenzen verschlüsselt. Aus den 4 Basen werden die in der DNA aufgereihten Gene kodiert. Es gibt mittlerweile einige hundert bekannte Gensequenzen, deren Funktion entschlüsselt ist.

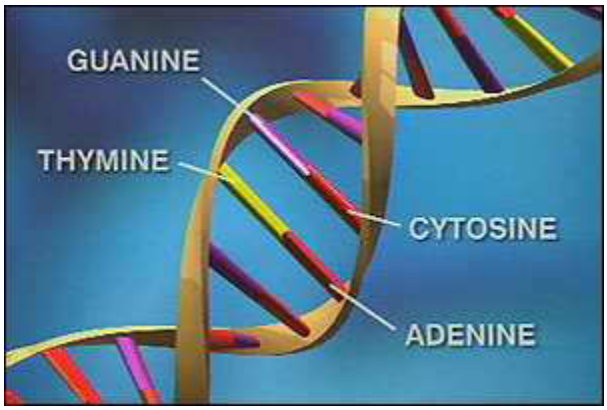


Abbildung 4.1: Das menschliche Genom
(Quelle: [WIKI])

Bedeutung in der Informatik:

Ein Genom der genetischen Algorithmen besteht aus einem Vektor mit konstanter Länge. In ihm werden die Variablen des Programms gespeichert. Ein einzelnes Gen beschreibt dabei eine abgeschlossene Informationseinheit innerhalb des Genoms. Die strukturelle Integrität der einzelnen Gene muss sichergestellt sein. Hierauf muss bei den verschiedenen Transformationsmethoden geachtet werden.

Die Informationen in der Zeichenkette können aus einfachen Zahlen oder Symbolen bestehen. Die Sprache des Interpreters wird dabei auf das Genom abgebildet. An dieser Stelle unterscheiden sich die „einfachen“ genetischen Algorithmen von der genetischen Programmierung. Bei der genetischen Programmierung haben die Genome keine feste Länge. Außerdem wird die Funktion, also die sinnvolle Verbindung der Variablen innerhalb des Genoms, nicht vorgegeben, sondern mit Hilfe des Algorithmus gefunden. Die Genome werden dabei in Form von hierarchischen Baumstrukturen gespeichert.

4.3.2 Die Population und seine Individuen

Aus biologischer Sicht:

Eine Population ist eine Gruppe von Individuen, die durch ihren Entstehungsprozess miteinander verbunden sind. Eine Population zeichnet sich in der Natur durch sexuell kompatible Individuen aus, die auf einem gemeinsamen Territorium zusammenleben. Je nach Anpassungsfähigkeit an die Lebensbedingungen, Verfügbarkeit von Nahrungsmitteln und anderen Faktoren, vergrößert oder verkleinert sich die Population.

Bedeutung in der Informatik:

Genau wie im biologischen Vorbild ist eine Population eine Gruppe von Individuen. Individuen bestehen dabei aus den Parametervariablen oder Programmen, die miteinander konkurrieren. Die Populationen werden unter zu Hilfenahme der Selektions-, Kreuzungs- und Mutationsmethoden in neue Populationen überführt. Dies geschieht in der Regel so lange, bis ein Individuum mit ausreichender Qualität erzeugt wurde.

4.3.3 Genotyp / Phänotyp

Aus biologischer Sicht:

Der Genotyp repräsentiert die Gesamtheit der vorhandene Gene eines Genoms im Zellkern eines Lebewesens. Der Phänotyp hingegen repräsentiert die Ausprägung eines Organismus, wie z.B. Größe, Augenfarbe, Gewicht, Form, Schnelligkeit usw. Wie bei eineiigen Zwillingen gut zu beobachten ist, muss ein exakt gleicher Genotyp nicht zwingend zu einem exakt gleichen Phänotyp

führen. Die Familie der Zwillinge z.B. kann mit Sicherheit kleine aber feine Unterschiede zwischen den beiden feststellen.

Bedeutung in der Informatik:

Der Genotyp repräsentiert, genau wie sein biologisches Pendant, die vorhandenen Gene eines Genoms. Der Phänotyp wird sich in der Regel bei gleichem Genotyp, anders als in der biologischen Variante, immer gleich verhalten. Das bedeutet, die in dem Genom kodierten Merkmale und das produzierte Verhalten, sind immer reproduzierbar.

4.3.4 Die Fitness eines Individuums

Aus biologischer Sicht:

Wie in Kapitel 4.1 besprochen, hängt das Überleben eines biologischen Organismus von der Anpassungsfähigkeit an seine Umwelt, also der Fitness eines Individuums ab. Nur diejenigen Organismen, die fit genug sind zu überleben, werden ihr Erbmaterial weitergeben können und so weitere Generationen gründen. Das Erbmaterial wird dabei durch die benannten Mechanismen in den meisten Fällen weiter verbessert.

Bedeutung in der Informatik:

Die Fitness stellt ein wichtiges Konzept der genetischen Algorithmen dar. Die Fitness eines Individuums wird mit Hilfe einer Fitnessfunktion festgestellt. Sie hat die Aufgabe quantitativ festzustellen, wie gut ein Genom geeignet ist das gegebene Problem zu lösen. In ihr ist das Ziel der Aufgabe kodiert. Das Ziel in Zahlen und Formeln zu kodieren ist eine der schwierigen Aufgaben, die man im Zusammenhang mit genetischen Algorithmen lösen muss. Diesen Umstand bemängeln auch die meisten Kritiker. Die Implementierung der Fitnessfunktion variiert stark von Problem zu Problem. Wie sich später noch zeigen wird, hängt von ihrer Modellierung das Gelingen oder Scheitern des Algorithmus ab.

4.3.5 Die Selektion

Aus biologischer Sicht:

Bei der natürlichen Selektion wird die vorhandene Population durch das Sterben einzelner Individuen ausgedünnt. Einzelne Individuen sterben in der Regel, weil sie sich nicht ausreichend an die gegebenen Umstände anpassen können. Das bedeutet: Fehlendes Durchsetzungsvermögen gegen stärkere Individuen, oder Krankheit in Folge von Kampf um Ressourcen oder nicht ausreichend starkem Immunsystem. So überleben hauptsächlich die stärksten Individuen mit der besten Fitness („survival of the fittest“). So trifft die Natur eine natürlich Auslese, um bessere Individuen erzeugen zu können.

Bedeutung in der Informatik:

Die Selektion wird analog zur biologischen Vorgehensweise angewendet. Diejenigen Genome mit der höchsten Fitness werden durch Kreuzung, Mutation und direkte Emigration in die neue Generation übernommen. Hierbei wird die Häufigkeit der Anwendung der einzelnen Methoden in der Regel durch einen Wahrscheinlichkeitsfaktor angegeben. Diese Parameter sind ein weiterer wichtiger Faktor, von deren Gewichtung die Erfolgsquote des Algorithmus stark abhängt.

4.3.6 Die Kreuzung des Erbmaterials

Aus biologische Sicht:

Biologische Organismen, die sich als überlebensfähig herausgestellt haben, kreuzen ihr Erbmaterial durch sexuelle Reproduktion und erzeugen so in den meisten Fällen anpassungsfähigere Exemplare mit einer besseren Fitness.

Bedeutung in der Informatik:

Analog zur Biologie beschreibt die Kreuzungsmethode das Verfahren zur Erzeugung neuer Individuen. Durch Kreuzung zweier erfolgreicher Genome entstehen neue Genome für die Nachfolgegeneration, mit der Möglichkeit eine bessere Fitness zu erzielen. Hierfür werden definierte Gensequenzen zweier Genome gezielt ausgetauscht. Dabei darf die strukturelle Integrität der Genome nicht verletzt werden. Das bedeutet, die Stellen an denen die Genome aufgetrennt und neu kombiniert werden, müssen wieder abgeschlossene logische Einheiten innerhalb des Genoms bilden.

4.3.7 Die Mutation des Erbmaterials

Aus biologischer Sicht:

Bei der Entstehung neuer Individuen, die aus der Selektion und Kreuzung des Erbmaterials entstehen, hat die Natur durch das Zufallsprinzip Mutation einen weiteren wichtigen Faktor für eine erfolgreiche Anpassung an die Umgebung hervorgebracht. Auf diese Weise können neue, zufällige Konzepte in das neue Individuum einfließen. Diese müssen nicht immer von Vorteil sein, bieten dem Organismus aber eine Chance sich auf neuen Entwicklungsbahnen zu bewegen und letztendlich eine höhere Entwicklungsstufe zu erreichen.

Bedeutung in der Informatik:

Die Mutationsmethode beschreibt in der Informatik den Vorgang, der auf die durch Selektion oder Kreuzung gewonnenen Genome angewendet wird. Die Mutationsmethode ändert den Inhalt einer definierten Anzahl von Genen (abgeschlossene logische Informationseinheiten innerhalb des Genoms) durch zufällige neue Inhalte. Die strukturelle Integrität darf dabei, wie auch schon bei

der Kreuzungsmethode, nicht verletzt werden. Genau wie im biologischen Vorbild ist die Mutation ein wesentlicher Faktor für die Entstehung von neuen Lösungsansätzen und die Durchquerung des Lösungsraums. Die Mutationsmethode verhindert effektiv das „Steckenbleiben“ in lokalen Maxima des Lösungsraumes.

4.4 Genetische Algorithmen

Die auf der Darwinschen Evolutionstheorie basierenden Modelle der künstlichen Evolution wurden in den sechziger Jahren zeitgleich aber unabhängig voneinander an verschiedenen Orten der Welt entwickelt. Zu den bedeutendsten Entwicklern dieser Modelle gehören zweifellos Ingo Rechenberg von der Universität Berlin, sowie John Holland von der Universität Michigan USA. Ingo Rechenberg ist Begründer der Evolutionsstrategien, John Holland wiederum Begründer der genetischen Algorithmen. Beide „Strömungen“ wurden in den darauffolgenden Jahren weiterentwickelt und verbessert. Lange Zeit wurde über die Richtigkeit der einzelnen Vorgehensweisen und Methoden, sowie vor allem der „richtigen“ Kodierung der Genome gestritten. Diese Dogmen wurden in den darauffolgenden Jahren glücklicherweise aufgeweicht und so eine flexiblere Sicht auf die Anwendungen der einzelnen Strategien möglich. Die wesentlichen Unterschiede liegen in der Kodierung der Genome, sowie in der unterschiedlichen Ausprägung und Anwendung der in Kapitel 4.3 beschriebenen Methoden. Während die Anhänger der Evolutionsstrategien die Kodierung der Genome mit reellen Zahlen vorziehen, bevorzugen die Anhänger der genetischen Algorithmen die Kodierung der Genome in binärer Form.

4.4.1 Ablauf Genetischer Algorithmen

Im folgenden Kapitel möchte ich näher auf die angesprochenen Verfahrensweisen eingehen. Da sich beide Verfahrensweisen sehr ähnlich sind und die essenziellen Ideen der Evolution von beiden Verfahren benutzt werden, möchte ich mich in dieser Arbeit auf die genetischen Algorithmen und die darauf aufgebaute genetische Programmierung konzentrieren. Zur Anwendung kommt eine Mischung der verschiedenen Verfahren. Die folgende Abbildung zeigt das Grundprinzip eines genetischen Algorithmus.

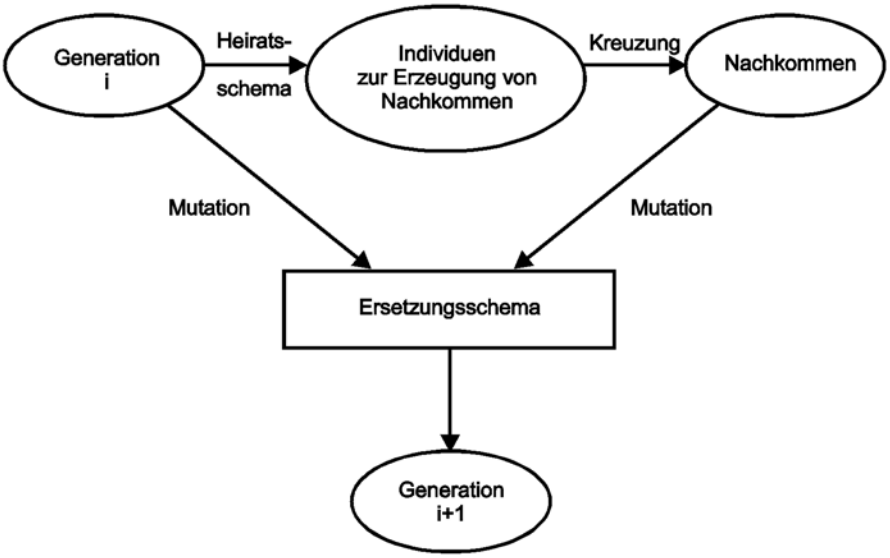


Abbildung 4.2: Ablaufschema - Genetischer Algorithmus
(Quelle: Genetische Algorithmen und Evolutionsstrategien)

Der Ablauf zur Erstellung und Durchführung eines genetischen Algorithmus ist im folgenden Pseudocode näher beschrieben. Dieser Ablauf ist quasi die Grundstruktur, die jeden genetischen Algorithmus auszeichnet. Da es mittlerweile viele verschiedene Varianten der einzelnen Methoden gibt (Heiratsschema / Selektionsverfahren, Ersetzungsschema, Mutationsmethode, Rekombinationsmethode usw.) möchte ich hier nur auf die bekanntesten eingehen.

Pseudocode Genetischer Algorithmus:

0. Wähle eine geeignete Kodierung der Chromosomen
1. Initialisiere eine zufällige Population von Chromosomen und nenne die Ausgangspopulation Generation 0

   Wiederhole bis Fitness zufriedenstellend oder Abbruchbedingung (z.B. Generation > 1000) erreicht ist
2. Bewerte alle Elemente der aktuellen Generation gemäß Fitnessfunktion
3. Selektiere Paare (oder größere Subpopulation) gemäß Heiratschema und erzeuge mittels Rekombination (Crossover) Nachkommen der aktuellen Generation
4. Mutiere die Nachkommen
5. Ersetze Elemente der aktuellen Generation durch die Nachkommen gemäß Ersetzungsschema und erzeuge so eine neue Generation („survival of the fittest“)
6. Aktualisiere Abbruchbedingung (z.B. Generationszähler)

(Quelle: Genetische Algorithmen und Evolutionsstrategien)

Nachdem die Urpopulation mit Hilfe eines Zufallgenerators erstellt wurde (Generation 0), wird die Fitness der in der Population befindlichen Individuen berechnet. Zur Ermittlung der Fitnesswerte kommen verschiedene Methoden zum Einsatz. Da die Fitnessfunktion das eigentliche Optimierungskriterium und somit das Ziel festlegt, ist auf besondere Sorgfalt bei der Erstellung dieser zu achten. Die gebräuchlichsten Funktionen sind die proportionale, lineare und gleichverteilte Fitness. Auf die für diesen Versuch eingesetzte Fitnessfunktion wird in Kapitel 6.2.5 später noch detaillierter eingegangen.

4.4.2 Heiratsschemata

Durch das Heiratsschema (auch Selektionsverfahren genannt), wird festgelegt welche Elemente der bereits bewerteten Population zur Erzeugung neuer Genome herangezogen werden. Das für die genetischen Algorithmen klassische Heiratsschema „Roulette“ z.B. wählt die Elemente mit einer Wahrscheinlichkeit aus, die proportional zu Ihrer Fitness ist. Daraus folgt, dass Elemente mit einer guten Fitness häufiger in die neue Population gelangen, als solche mit einer schlechten. Die ausgewählten Elemente werden durch Rekombination, also der Kreuzung zweier Elemente, in die neue Population übernommen. Dadurch soll bezweckt werden, die Fitness der Individuen mit fortschreitender Generation kontinuierlich zu verbessern.

Roulette Methode:

Pseudocode:

1. Bestimme die Fitneß f_i jedes Individuums i aus der aktuellen Population.

2. Bestimme die Gesamtfitneß F der aktuellen Population:

$$F = \sum_{i=1}^{n} f_i$$

3. Bestimme die Auswahlwahrscheinlichkeit p_i jedes Individuums i der aktuellen Generation: $p_i = f_i / F$

4. Bestimme die kumulative Auswahlwahrscheinlichkeit q_i jedes Individuums i der aktuellen Generation:

$$q_i = \sum_{j=1}^{i} p_j$$

5. Wiederhole die folgenden Anweisungen n mal:

a. Erzeuge eine reelle Zufallszahl r aus dem Intervall [0, 1].

b. Falls $r < q_1$, dann wähle Individuum 1, sonst wähle Individuum i mit $q_{i-1} < r <= q_i$.

Die Roulette Methode ist das Standard Heiratsschema bei den genetischen Algorithmen. Die Auswahl der Elemente kann man sich bildlich als eine Kreisscheibe vorstellen, auf der n Segmente der Länge p_i aufgezeichnet sind.

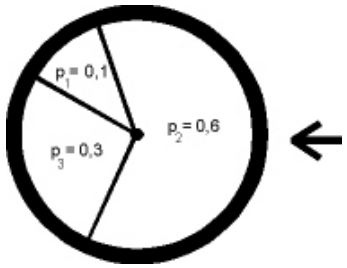


Abbildung 4.3: Roulette Methode

Die Scheibe wird zur Auswahl der Elemente m-mal gedreht und jeweils das Element ausgewählt, auf dem der Zeiger zum Stillstand kommt. Durch die Roulette Methode wird sichergestellt, dass jedes Individuum eine zu seiner Fitness proportionale Chance erhält seine Gene weiterzugeben. Auch Individuen mit einer geringeren Fitness erhalten so eine Chance weiterverwendet zu

werden, wenngleich die Chance zur Übernahme geringer ist, als bei denjenigen Elementen mit einer höheren Fitness.

Ranking Methode:

Bei der Ranking Methode werden die Individuen einer Population nach ihrer Fitness in absteigender Reihenfolge sortiert und nummeriert. Die Auswahlwahrscheinlichkeit für ein Individuum berechnet sich aus:

$$p_i = q - r * (Rang(i) - 1)$$

q: ist eine Konstante aus dem Intervall [1/n, 2/n]
n: ist die Populationsgröße
r: $r = 2(qn - 1) / (n(n - 1)$

Der Selektionsdruck ist mittels der Konstanten q steuerbar. Die eigentliche Auswahl erfolgt wie beim Roulette-Prinzip.

Schöpfkellen Methode:

Zunächst wird wie bei den anderen Methoden auch die Fitness jedes Individuums in der aktuellen Population festgestellt. Durch den Parameter S wird die Größe der Schöpfkelle festgelegt. Zur Selektion der zu verwendenden Individuen werden n mal zufällig jeweils S Individuen aus der aktuellen Population ausgewählt. Wobei aus jeder Schöpfkelle nur das Individuum mit der besten Fitness zur Kreuzung und Mutation herangezogen wird. Je größer die Schöpfkellengröße S ist, desto kleiner ist die Wahrscheinlichkeit, dass Individuen geringer Fitness zur Weiterverarbeitung ausgewählt werden.

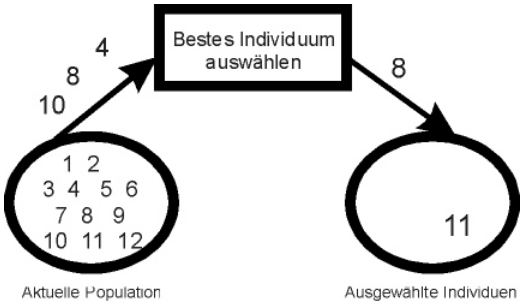


Abbildung 4.4: Schöpfkellen Methode mit S = 3

Wie in der Abbildung zu sehen ist, werden einige der neu entstandenen Individuen mit der Mutationsmethode bearbeitet, um eine zufällige Streuung der Suche zu erreichen. Für die Mutation gibt es verschiedenste Schemata. Die Standardvariante stellt die gleichverteilte Mutation dar, bei der einzelne Gene durch neue Zufallswerte besetzt werden.

Nachdem mit Hilfe des Heiratsschemas Individuen ausgewählt und neue Individuen durch Kreuzung und Mutation erstellt wurden, wird mit Hilfe des Ersetzungsschemas bestimmt, was mit den Individuen der aktuellen Population geschehen soll.

4.4.3 Ersetzungsschemata

Die folgenden Ersetzungsschemas gehören zu den bekanntesten und sollen hier kurz vorgestellt werden.

Generational Replacement:

Bei dem Generational Replacement Schema wird die aktuelle Population vollständig durch seine Nachkommen ersetzt. Diese Ersetzungsmethode birgt die Gefahr in sich, dass ein Genom mit der bisher besten gefundenen Fitness verloren geht, oder die durchschnittliche Fitness schlechter ist, als die der Elterngeneration. Um zu verhindern, dass ein bereits gutes Genom verloren geht, bietet sich die nächste Methode an.

Elitismus:

Bei dem Schema Elitismus werden die n besten Elemente, also quasi die Elite der aktuellen Population, in die nachfolgende Population übernommen. Ein Absinken der schon einmal gewonnenen besten Fitness wird so verhindert. Allerdings birgt diese Vorgehensweise die Gefahr in sich, dass die übernommene Elite die nächsten Generationen dominiert. Dies liegt daran, dass wie in Kapitel 4.4.2 beschrieben, die besten Elemente die größte Chance haben Nachkommen zu bilden. Dieser Dominanzeffekt kann zu einer frühzeitigen Stagnation des Evolutionsprozesses führen.

Schwacher Elitismus:

Um das beim Schema Elitismus genannte Problem der frühzeitigen Stagnation zu verhindern, kann mit dem Schema „Schwacher Elitismus“ gearbeitet werden. Hierbei werden wiederum n beste Individuen in die neue Population übernommen, allerdings mit dem Unterschied, dass diese vor der Übergabe mutiert werden. So wird sichergestellt, das trotz Übernahme der elitären Individuen eine Streuung stattfindet.

Delete-n mit Elitismus:

Dieses Schema ersetzt n zufällig gewählte Elemente der aktuellen Population durch n Nachkommen der Vorgängergeneration. Die Elite der Generation bleibt dabei geschützt. Dies hat zur Folge, dass die besten Individuen sich in jedem Fall reproduzieren können, die Population durch die zufällige Auswahl der Elemente aber trotzdem gut durchmischt ist.

4.4.4 Das Abbruchkriterium

Die Individuen werden von Generation zu Generation weiterentwickelt, bis eine bestimmte geforderte Fitness erreicht ist oder ein bestimmtes Abbruchkriterium zutrifft. Da in der Regel am Anfang einer Reihe von Versuchen noch keine brauchbaren Aussagen über die benötigte Fitness gemacht werden können, wird als Abbruchkriterium oft die Anzahl der zu generierenden Generationen gewählt.

Das Abbruchkriterium definiert somit eine Bedingung für die Beendigung des Suchvorgangs. In der Regel wird als Abbruchkriterium ein gewünschter Qualitätsgrad des zu erzeugenden Programms benutzt. Da die Qualität des Programms anhand der Fitness des Genoms gemessen wird, bestimmt man hier einen Wert, bei dem man von einer ausreichenden Qualität ausgeht. Als weiteres Abbruchkriterium kommt eine definierte Anzahl von Generationen in Frage. Dies ist insbesondere dann hilfreich, wenn noch kein ausreichendes „Gefühl“ für die Fitnesswerte vorhanden ist.

4.5 Genetische Programmierung

Die genetische Programmierung ist eine Weiterentwicklung der genetischen Algorithmen und der Evolutionsstrategien. 1992 stellte John R. Koza [KOZA 1992] von der Universität Stanford mit seinem Buch „Genetic Programming – On the Programming of Computers by Means of Natural Selection“ die genetische Programmierung vor. Die Grundidee der genetischen Programmierung ist die Anwendung der evolutionären Methoden auf Computerprogramme. Das heißt, der Computer entwirft selbstständig einen Computercode, oder vielmehr eine Population von Computerprogrammen die im evolutionären Prozess entwickelt werden. Diese Programme passen sich im Laufe der Evolution, wie bei den anderen Verfahren auch, an die in der Fitnessfunktion gegebene Zielsetzung an.

Mit Hilfe der genetischen Programmierung ist es also nicht nur möglich einzelne Parameter eines definierten Problems durch Approximation zu lösen, vielmehr ist man nun auch in der Lage ein hierarchisches Programm zu finden, das ein gegebenes Problem löst. Im Gegensatz zu den genetischen Algorithmen, die mit fixen linearen Vektoren zur Speicherung der Parameter arbeiten, werden bei der genetischen Programmierung flexible Baumstrukturen eingesetzt. Dies ist neben der oben genannten Erweiterung ein Hauptmerkmal der genetischen Programmierung. Ein weiterer Unterschied besteht in der Erstellung der Nachfolgepopulationen. Wie in dem Chart zu erkennen, werden die Methoden der Mutation und Kreuzung in einem Schritt benutzt, während bei den genetischen Algorithmen beide Methoden nacheinander benutzt werden.

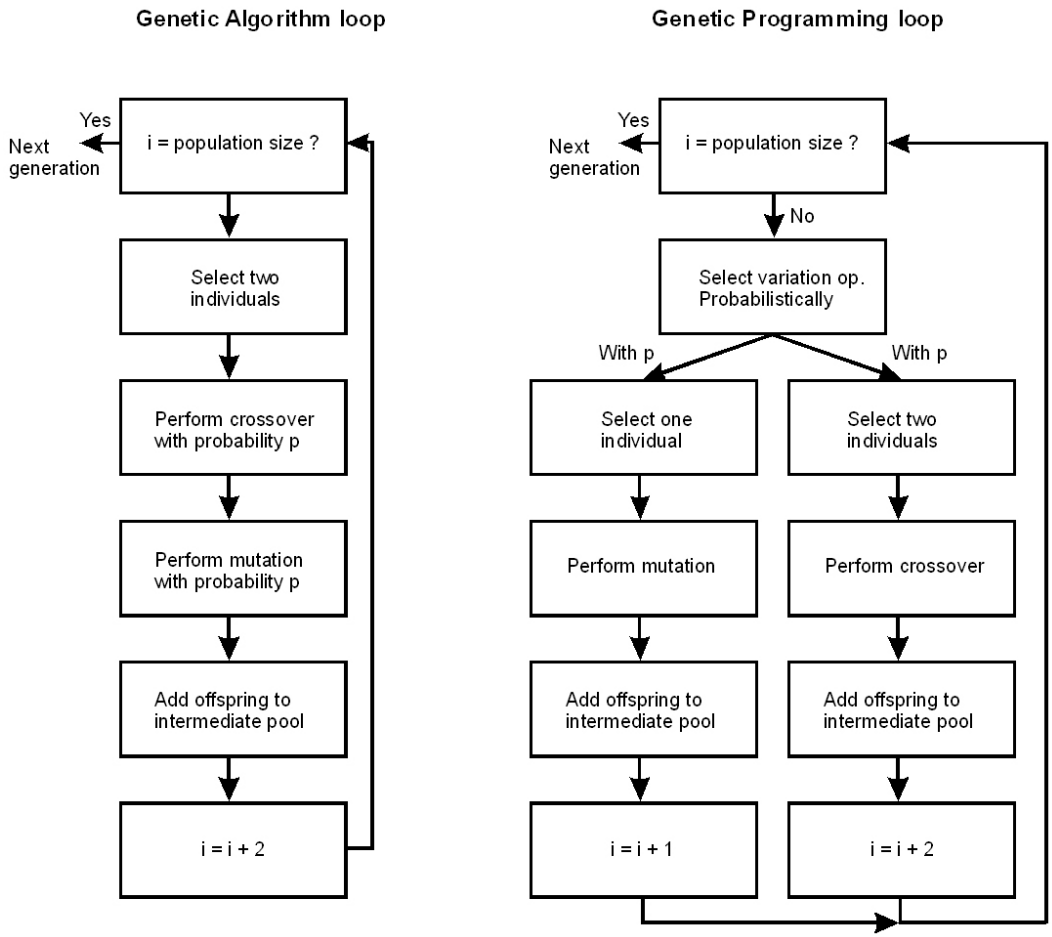


Abbildung 4.5: Vergleich GA – GP Loop
(Quelle: [EIBEN 2003])

Im folgenden Chart ist der Ablauf der genetischen Programmierung, wie er von Koza entwickelt wurde, zu sehen. Vergleicht man diesen mit dem Ablauf der genetischen Algorithmen, ist ein Unterschied zu erkennen. Während bei den genetischen Algorithmen grundsätzlich zwei Individuen selektiert werden und mit Hilfe der Kreuzungs- und Mutationsmethode weiterverarbeitet werden, gibt es bei der genetischen Programmierung drei verschiedene Wege zur Auswahl. Welcher Weg genommen wird, ist über Wahrscheinlichkeitsparameter angegeben. Um eine neue Generation zu erzeugen werden also je nach eingestellter Wahrscheinlichkeit die Reproduktion, Kreuzung oder Mutation ausgewählt. Für die Auswahl welches Individuum aus der Ursprungspopulation zur Weiterverarbeitung verwendet wird, kommen die gleichen Strategien zum Einsatz die im Kapitel 4.4 „Genetische Algorithmen“ beschrieben wurden. Bei der Reproduktion wird einfach ein aus der Vatergeneration ausgewähltes Individuum in die neue Generation kopiert. Bei der Kreuzungsmethode werden aus zwei selektierten Individuen durch Kreuzung zwei neue Individuen erstellt und eingefügt. Während bei den genetischen Algorithmen die Mutationsmethode lediglich als Nachfolgeschritt einer Kreuzung eingesetzt wird, ist die Mutation eines kopierten Individuums aus der Vatergeneration bei der GP sofort möglich.

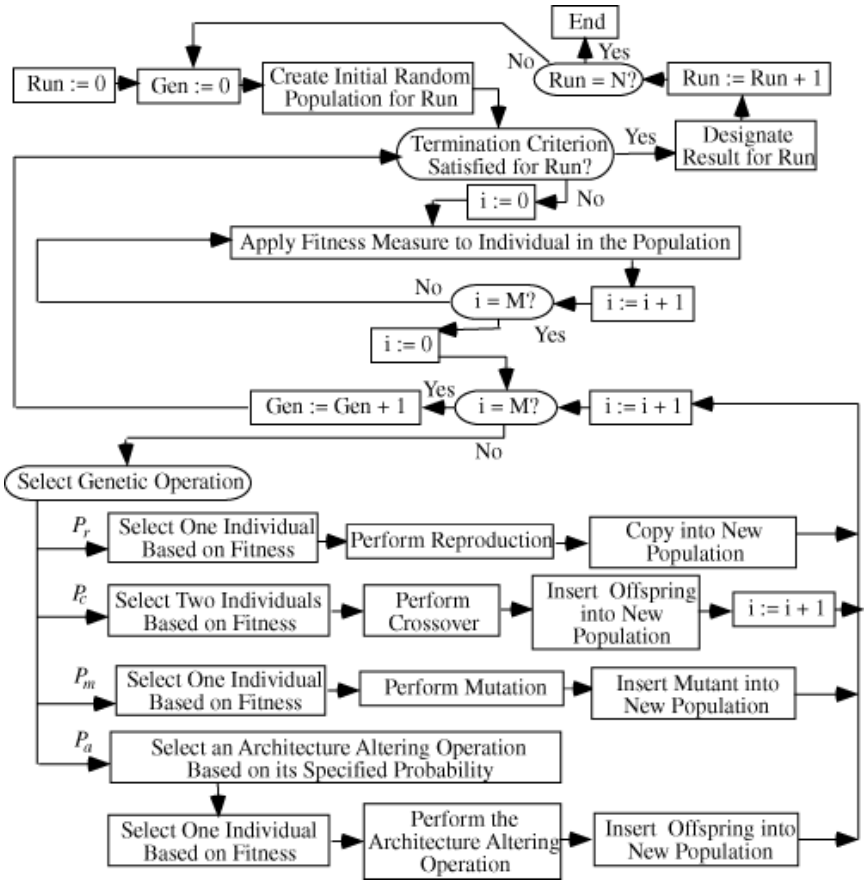


Abbildung 4.6:Ablaufchart Genetische Programmierung
(Quelle: [Koza 1992])

Während bei den evolutionären und genetischen Algorithmen die Chromosomen aus Vektoren, gefüllt mit reellen Zahlen oder binären Code mit fester Länge bestehen, werden die Chromosomen bei der genetischen Programmierung durch hierarchische Baumstrukturen repräsentiert. Ein Chromosom setzt sich aus einer Funktionsmenge und einer Terminalmenge zusammen. Die Terminalmenge umfasst alle für die Funktionen der Funktionsmenge erlaubten Eingaben. Die Funktionsmenge wiederum kann z.B. aus arithmetischen, trigonomischen, logischen, exponentiellen und anderen Funktionen bestehen. Außerdem sind die für Computerprogramme oft benötigte Zuweisungsoperatoren, Verzweigungen oder Iteration ebenso erlaubte Funktionen. Die Terminalmenge kann neben den Eingabevariablen auch Zustandsvariablen, Zahlenwerte oder andere Konstanten beinhalten. Wie bereits erwähnt wird für die Darstellung der Chromosomen die Baumstruktur verwendet. Die Funktionen bilden dabei die inneren Knoten und die Terminals die Blätter des Baumes. Die Gesamtmenge aller ausführbaren Programme setzt sich aus allen möglichen Kombinationen der gewählten Terminals und Non-Terminals zusammen. Diese Menge bildet den Suchraum in dem nach einem passenden Programm gesucht wird. Um ein gegebenes Problem lösen zu können, muss die Funktions- und Terminalmenge ausreichend groß gewählt werden, um daraus überhaupt eine Lösung bilden zu können. Die Auswahl einer geeigneten Funktions- und Terminalmenge fordert wie bei den anderen Verfahren auch ein gewisses Vorwissen der Lösung. Wird die Menge der Funktionen und Terminals unnötig groß gewählt, vergrößert sich gleichzeitig

der Suchraum unnötig, was unter Umständen eine längere Suche zur Findung einer Lösung nach sich zieht.

GP Chromosomen (Programme) werden durch S-Expressions (Prefix-Notation[3]) dargestellt:

z.B. : eine Even-2-Parity Funktion

als S-Expression: (OR (AND (NOT D0) (NOT D1)) (AND D0 D1))

und als Baumansicht:

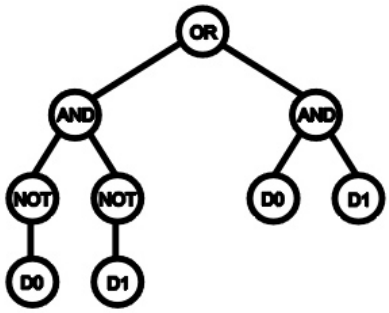


Abbildung 4.7:Genom - Baumstruktur

4.6 Die Kreuzungs- und Mutationsmethode

Je nach verwendetem Datentyp werden die Mutation und Kreuzung der Genome strukturbedingt unterschiedlich durchgeführt. Im Folgenden möchte ich die Vorgehensweise anhand eines Beispiels für vektorbasierte Genome (reelle Zahlen) und baumorientierte Genome wie sie in der genetischen Programmierung verwendet werden näher vorstellen.

4.6.1 Mutationsmethode

Die Mutationsmethode ermöglicht kleine zufällige Veränderungen innerhalb des Genoms. Die Anzahl der Gene und die Stellen die mutiert werden sollen können fest vorgeben oder durch einen Zufallsgenerator bestimmt werden. Als Standardschema gilt die gleichverteilte Mutation. Hierbei wird das Genom

[3] Prefix-Notation: Operatoren werden vor die Ausdrücke gestellt

schrittweise durchlaufen und für jedes Gen mit Hilfe einer gleichverteilten Zufallszahl bestimmt, ob eine Mutation an der aktuellen Stelle stattfinden soll. Ein weiteres geläufiges Mutationsschema ist die Positionsmutation. Hierbei wird für jede Genposition innerhalb des Genoms eine eigene Mutations-wahrscheinlichkeit festgelegt. Eine andere Möglichkeit ist, festgeschriebene Genpositionen innerhalb eines Genoms pro Vorgang zu mutieren. Die folgenden Abbildungen verdeutlichen noch einmal den Vorgang der Mutation für vektorbasierte und baumorientierte Genome. Die Mutation findet in der Regel während der Überführung in die Nachfolgegeneration statt.

Mutation bei Genom als Vektor:

Die folgende Abbildung zeigt die Mutation zweier im voraus festgelegter Genpositionen. In diesem Beispiel werden durch die Mutationsmethode zwei fest vorgegebene Positionen innerhalb des Genoms bei der Überführung in die Nachfolgepopulation durch einen Zufallswert neu belegt.

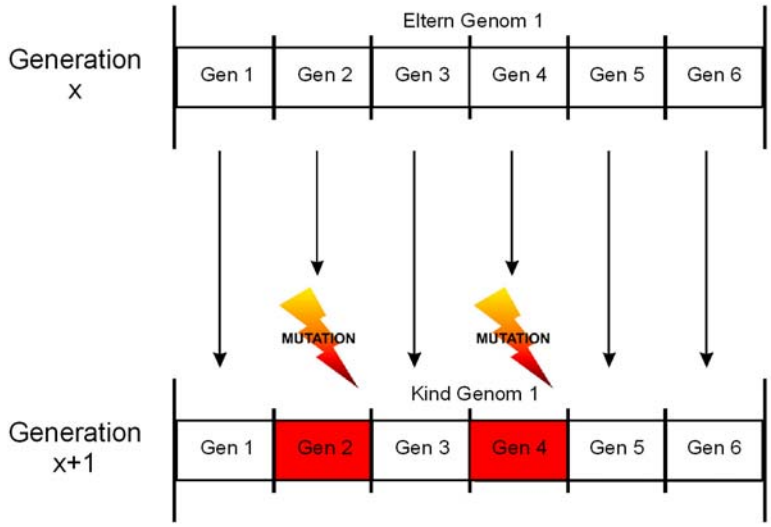


Abbildung 4.8:Mutation - Genom als Vektor

Mutation bei Genom als Baumstruktur:

Die folgende Abbildung zeigt die Mutationsmethode angewendet auf die baumorientierte Genomstruktur. Da es anders als bei den Genomen, die durch lineare feste Vektoren dargestellt werden, keine feste Struktur mit konstanter Länge gibt, können keine festen Stellen zur Mutation innerhalb des Genoms angegeben werden. Als Mutationsparameter ist daher eine prozentuale Angabe der Mutationswahrscheinlichkeit innerhalb eines Genoms, in Abhängigkeit von der Anzahl der enthaltenen Gene sinnvoll. Je nach dem, ob ein Blatt oder Knoten des Baumes mutiert werden soll, ist auf die Neubesetzung durch Terminals oder Non-Terminals zu achten.

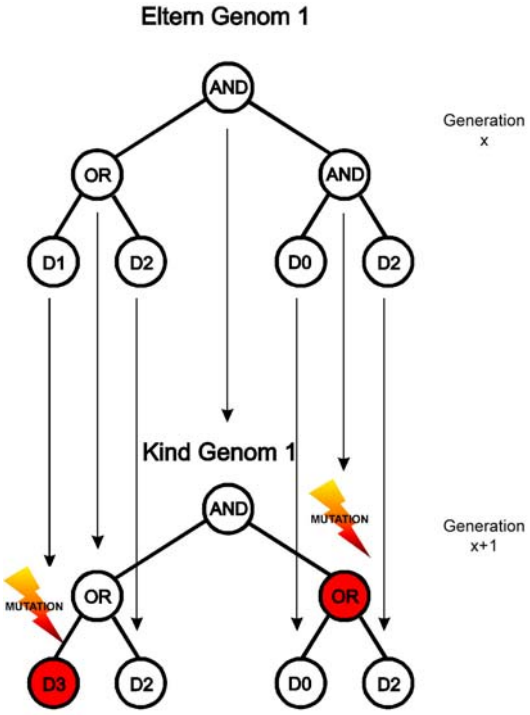


Abbildung 4.9: Mutation – Genom als Baumstruktur

4.6.2 Kreuzungsmethoden

Anders als die Kreuzungsmethode, die einen eher zufälligen Faktor in den Evolutionären Prozess bringt, dient die Kreuzungsmethode dazu, den Suchraum durch gezielte „Crossing Over“ Verfahren effizient zu durchschreiten. Zu den bekanntesten „Crossing Over“ Schemata gehören die „one point crossover“, „two point crossover“ und die „Zufalls Schablone“. Bei diesen Verfahren wird mittels einer natürlichen Zufallszahl ein oder mehrere „Crossing Over“ Punkte im Genom bestimmt. An den zufällig gewonnen Punkten wird das Genom dann gekreuzt. Da es diverse Rekombinationsverfahren gibt, möchte ich an dieser Stelle nicht näher darauf eingehen und verweise auf die Literatur.

Genom als Vektor:

In der folgenden Abbildung ist ein Beispiel für das „Crossing Over“ von zwei linearen Vektoren mit zwei zufällig zum Tausch ausgewählten Genen zu sehen. Aus zwei Ausgangsgenomen entstehen durch das „Crossing Over“ Verfahren zwei neue gekreuzte Genome für die Nachfolgegeneration.

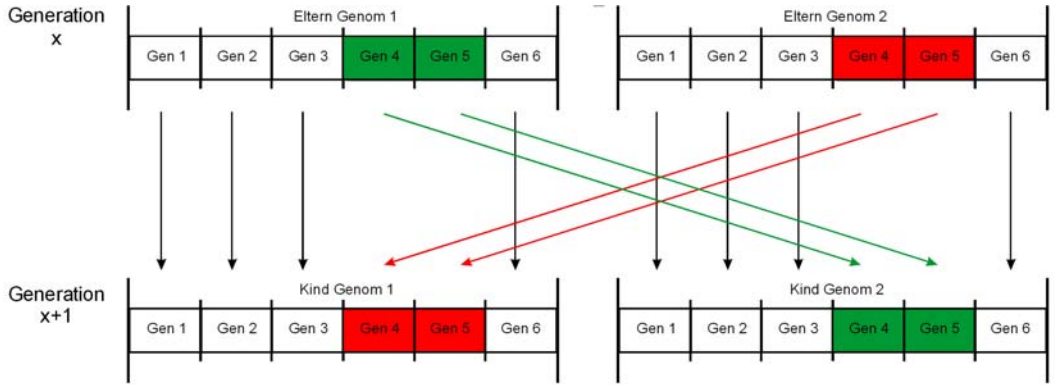


Abbildung 4.10: „Cross Over“ Genome als Vektoren

Genom als Baumstruktur:

Das „Crossing Over“ bei Genomen, die als Baumstrukturen gespeichert werden, verhält sich ähnlich wie bei Genomen aus linearen Vektoren. Einzelne Unterbäume werden durch einen Zufallsalgorithmus aufgetrennt und in der Nachfolgegeneration durch Kreuzung wieder zusammengesetzt. Wie bei dem Mutationsverfahren ist auch hier darauf zu achten, dass die Blätter der Knoten lediglich Terminals enthalten dürfen und die Knoten aus Non-Terminals bestehen müssen.

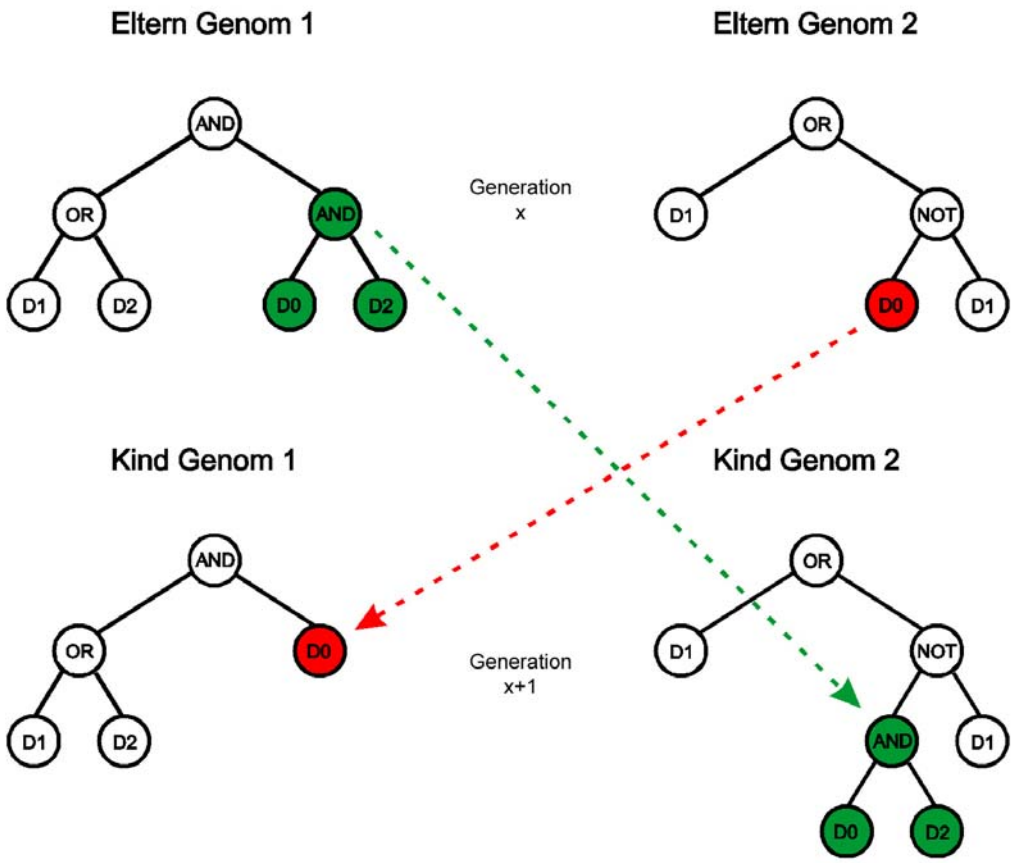


Abbildung 4.11: „Cross Over“ – Genome als Baumstrukturen

4.7 Der Lösungsraum

Die genetischen Algorithmen gehören zur Gruppe der heuristischen Suchverfahren. Die heuristische Suche wird insbesondere dann eingesetzt, wenn eine systematische Suche mit einem zu hohen Zeitaufwand zur Berechnung einer Lösung verbunden ist. Mit der heuristischen Suche werden zumeist keine optimalen, für das Problem aber ausreichend gute Lösungen gefunden.

In der folgenden Abbildung ist beispielhaft ein Lösungsraum für zwei zu findende Variablenwerte als dreidimensionales Chart dargestellt. Auf der X- und Y-Achse sind alle möglichen Variablenkombinationen abgebildet. Auf der Z-Achse sind die dazugehörigen Fitnesswerte für die möglichen Kombinationen aufgetragen. Natürlich sind bei den meisten zu lösenden Problemen erheblich mehr Variablen nötig. Die Anzahl der Dimensionen erhöht sich dann dementsprechend. Die Abbildung gleicht einem Terrain mit unterschiedlichen Höhenausprägungen. In den Gegenden wo die gebildete Landschaft höher als die umgebende ist, befinden sich Variablenkombinationen mit einer guten Fitness. Ziel ist es nun diese lokalen Optima oder sogar das globale Optima zu finden. Das globale

Optima ist der Punkt, der am höchsten herausragt und somit den besten Fitnesswert darstellt.

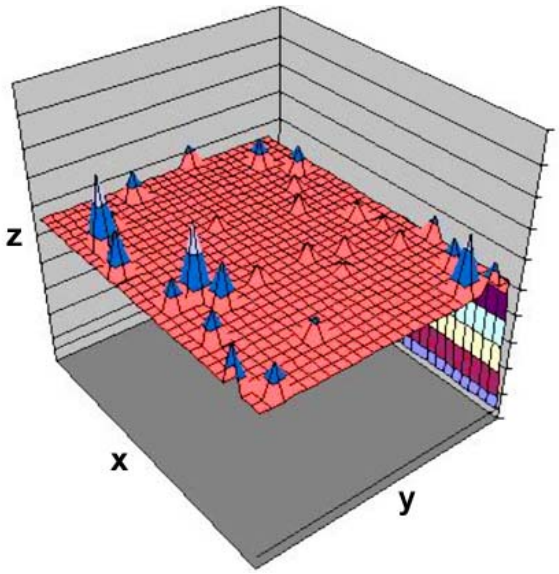


Abbildung 4.12: Lösungsraum

Wie arbeitet sich ein genetischer Algorithmus durch den Suchraum?
Die Urpopulation wird durch einen Zufallsgenerator gebildet. Die Startpunkte im Suchraum werden somit per Zufall gesetzt. Ausgehend von diesen Punkten wird die Umgebung nach besseren Fitnesswerten durchsucht. Dies geschieht mit den in Kapitel 4.4 beschriebenen Methoden der Selektion, Kreuzung und Mutation.

Die Selektionsmethode sorgt dafür, dass Genome, also Punkte im Suchraum, mit einer guten Fitness in die nächste Population übernommen werden. Die „Cross Over“ Methode sorgt dafür, dass auch die Räume zwischen zwei lokalen Maxima durchsucht werden. Evtl. bessere Lösungen in den Zwischenräumen können dadurch gefunden werden. Durch die Mutationsmethode wird sichergestellt, dass der Algorithmus nicht in einem lokalen Maximum stecken bleibt, wie es z.B. bei dem Hill Climbing Algorithmus der Fall ist. Qualitativ bessere Lösungen können so erreicht werden. Die Mutationsmethode führt so eine zufällige Streuung in die Suche ein.

4.8 Anwendung als Lernverfahren für mobile Roboter

In den vorherigen Abschnitten wurden die verschiedenen evolutionären Verfahren als eine Arte heuristisches Suchverfahren vorgestellt. Wie aber kann diese Technik nun zur Wissensbildung eingesetzt und speziell auf das vorgegeben Problem angepasst werden?

Für die autonome Steuerung eines Roboters gibt es verschiedene Ansätze, die je nach Anforderung eingesetzt werden können. Die übliche Vorgehensweise ist dabei folgende: In der Regel nimmt ein autonomes System Informationen über seine Umwelt mit Hilfe von Sensoren auf, verarbeitet diese und trifft dann eine Entscheidung welche Aktion ausgeführt werden soll, um das gewünschte Ziel zu erreichen. Dabei kann man bei der Entscheidungsfindung, welche Aktionen auszuführen sind, grob zwischen einer modellbasierten und einer reaktiven Verarbeitung unterscheiden. Bei der modellbasierten Entscheidungsfindung wird neben den eingehenden Sensordaten auch ein internes Weltbild, welches bereits vollzogene Erfahrungen speichert, zu Rate gezogen. Diese Vorgehensweise führt in den meisten Fällen zu komplexen Berechnungen, um das externe und interne Weltbild, sowie das gewünschte Ziel zu verarbeiten und daraus eine neue Aktion zu gewinnen. Diese Variante der Entscheidungsfindung ist recht komplex und somit nicht für jedes Problem angemessen und notwendig. Die zweite Möglichkeit ist die reaktive Entscheidungsfindung, die ein reflexartiges Handeln ermöglicht. Diese ist insbesondere bei wenigen Sensordaten sinnvoll, auf die eine schnelle Reaktion folgen soll. Hierfür wird die Wissensverarbeitung auf ein einfaches Matchingverfahren begrenzt, das eingehende Sensorpattern direkt auf Aktionen abbildet. Mit Hilfe eines regulären Automaten, welcher die erzeugten Sensorpattern in Form von Symbolen verarbeiten kann, sind dann einfache Handlungsabfolgen realisierbar. Die diversen Zustände werden dabei in Form von Look-Up Tables verarbeitet.

Betrachtet man die genetischen Algorithmen, so bestehen die Genome aus einer Folge von Zahlen oder Parametern. Eine erste Idee war nun, die Genome mit einer zufälligen Reihe von Motorbefehlen zu füllen und diese über einen Interpreter in die entsprechenden Motoranweisungen umzuwandeln. Zur Fitnessbestimmung eines Genoms sollten dann die verfügbaren Sensoren herangezogen werden, um festzustellen ob die ausgeführte Abfolge von Motorbefehlen den Roboter auf einer Linie mit Ball und Tor gebracht hat. Diese Idee wurde aber schnell wieder verworfen, als klar wurde, dass diese Vorgehensweise in Bezug auf die Ausgangsstellungen des Roboters sehr unflexibel ist. Mit der Änderung der Startposition, müsste die Abfolge der Motorbefehle neu angepasst werden, um das anvisierte Ziel zu erreichen. Mit anderen Worten, nach jeder Änderung der Startposition müsste mit Hilfe des genetischen Prozesses eine neue Abfolge von Motorbefehlen gefunden werden. Dies ist auch in Bezug auf den Zeitverbrauch unbefriedigend. Um den Roboter gezielt steuern zu können, also auch die von den Sensoren erfassbaren Umweltdaten nutzen zu können, ist es notwendig diese auf bestimmte Motorbefehle abzubilden. Dadurch wäre man in der Lage das vorgegeben Verhalten einmal über den genetischen Prozess zu erlernen und dieses Wissen dann für verschiedenste Startpositionen zur Steuerung des Roboters zu nutzen. Aus diesem Grund wurde entschieden, die Wissensrepräsentation in Form von einfachen Regeln vorzunehmen. Das Genom soll dabei aus einer Abfolge von „Wenn Dann Regeln“ bestehen, die bestimmte Sensorpattern direkt auf eine Aktion abbilden. Mit Hilfe des genetischen Algorithmus können die Regelbasen an das gewünschte Ziel angepasst werden. Eine reaktive Entscheidungsfindung scheint für das gegebene Problem angemessen zu sein.

5 GRUNDKONZEPT

Wie in der Aufgabenstellung beschrieben, soll der Roboter selbstständig eine Bewegung durchführen, um sich auf eine Linie mit einem definierten Tor und einem Infrarotball zu bewegen. Dafür muss der Roboter ein bestimmtes Verhalten erlernen. Der grundlegende Ansatz hierfür geht von der Modifikation zufällig generierter Verhaltensmuster aus, die in Form von Regelsätzen in den Genomen gespeichert sind und von einem genetischen Verfahren auf Ihre Zielsetzung hin modifiziert werden. Um dieses Ziel zu erreichen wird die gewünschte Position des Roboters relativ zu Tor und Ball in einer Fitnessfunktion kodiert. Die zufällig erzeugten Regelsätze werden durch einen Regelinterpreter nach Auswertung der Sensoren in die entsprechenden Motorbefehle umgesetzt. Die Fitnessfunktion bewertet nach jeder Bewegung des Roboters die Qualität der aktuellen Regelbasis. Die gewonnenen Fitnesswerte dienen dann als Qualitätsmerkmal, anhand dessen die Auswahl der geeigneten Regelbasen zur weiteren genetischen Optimierung ausgewählt werden.

Das Experiment soll nach Evaluierung der einzelnen Verfahren auf Basis der genetischen Programmierung von John R. Koza [KOZA 1992] durchgeführt werden. Da sich der Suchraum bei der genetischen Programmierung durch die Verwendung der Baumstrukturen und der sich daraus ergebenden Möglichkeiten erheblich vergrößert, habe ich mich entschieden mit linearen Vektoren fester Größe zu arbeiten. Da die Komplexität der zu lösenden Aufgabe recht hoch ist, möchte ich mich auf die Findung der geeigneten Parameter für das Regelwerk beschränken. Daraus ergibt sich für die Suche ein erheblicher Zeitvorteil, da nicht zusätzlich zu den gesuchten Parametern die Funktion der Lösung gesucht werden muss. Der funktionelle Zusammenhang der Parameter soll bereits in der Software vorgegeben werden. Dies bezieht sich insbesondere auf die Kombination der verschiedenen Sensordaten.

Aus den oben beschriebenen Verfahren wird ersichtlich, dass zur Durchführung einige wichtige Rahmenparameter bestimmt werden müssen. Dies sind im einzelnen die Populationsgröße, die Anzahl der zu erzeugenden Generationen, sowie die Anwendungswahrscheinlichkeit für die Selektions-, Mutations- und Kreuzungsmethode. Des weiteren ist eine geeignete Selektionsstrategie zu wählen und die Fitnessfunktion, in der das Ziel der zu lösenden Aufgabe kodiert ist, zu formalisieren. Was aber sind geeignete Parameter? Die Auswahl geeigneter Parameter ohne Vorwissen über ein gegebenes Problem ist sehr schwierig. Da es sich bei einer Robotersteuerung um eine Realtimeanwendung handelt, ist der Zeitbedarf ein kritischer Faktor. Um in einer angemessenen Zeit zu einem Ergebnis zu kommen, müssen die Rahmenparameter zur Ausführung des Algorithmus sinnvoll ausgewählt werden. Um ein „Gefühl“ für die geeignete Einstellung der Rahmenparameter zu bekommen, habe ich mich entschieden hierfür einen softwarebasierten Simulator zu entwickeln. Mit Hilfe des Simulators

können sinnvolle Rahmenparameter innerhalb kurzer Zeit herausgefunden werden. Dies ist möglich, da auf die langsame mechanische Aktorik und Sensorik verzichtet werden kann.

5.1 Die Systemarchitektur

Die zu erstellende Software soll in Form einer 3-Tier-Architektur aufgebaut werden. Die 3-Tier-Architektur modelliert das Gesamtsystem in drei Stufen. Diese sind, wie in der Abbildung zu erkennen, die User-Interface-Schicht, die Application-Schicht und die Persistence-Schicht.

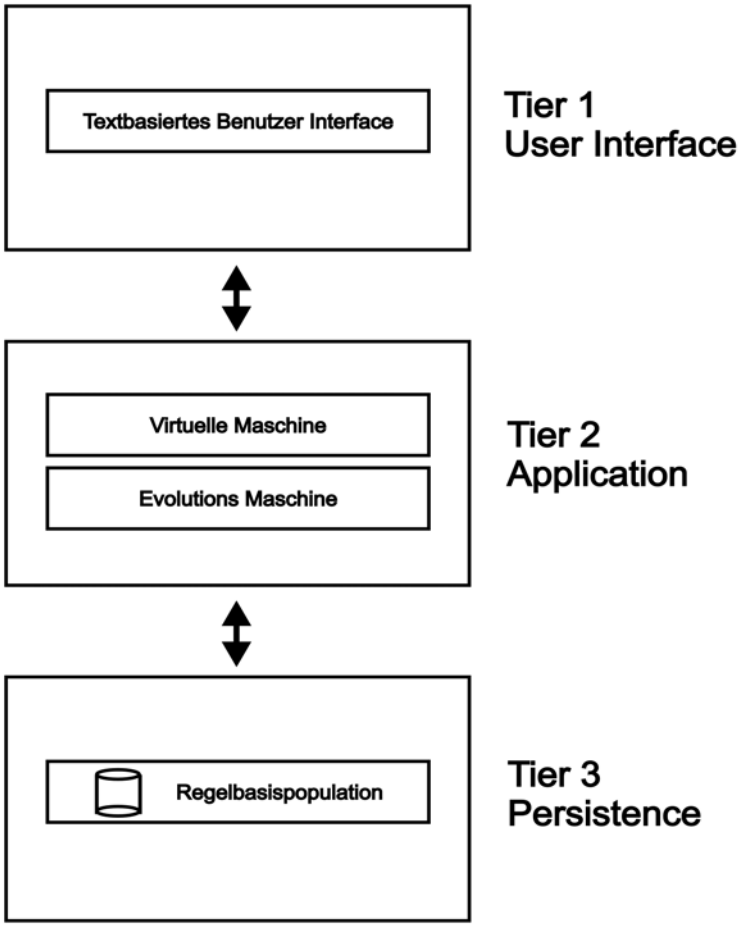


Abbildung 5.1: 3-Tier-Architektur

Die User-Interface-Schicht ist für die Interaktion mit dem Anwender zuständig und regelt die Eingabe der notwendigen Parameter, sowie die Ausgabe der errechneten Ergebnisse. Das User-Interface ist als textbasierte Schnittstelle geplant, da die Hardware nur über begrenzte Ausgabemöglichkeiten verfügt. In der Application-Schicht befindet sich die Anwendungslogik, welche die zwei

Hauptkomponenten beinhaltet: Die Virtuelle Maschine und die Evolutionsmaschine. In der Persistence-Schicht werden die benötigten Daten in Form der Regelbasispopulation vorgehalten. Eine klare Trennung der Schichten, sowie die Verwendung von Komponenten innerhalb der Schichten, soll bei der Portierung der Anwendung von der PC-Hardware (Simulator) auf die Realanwendung (Roboter) hilfreich sein. Die geplante Architektur soll die Wiederverwendbarkeit des Codes gewährleisten und somit den Programmieraufwand verringern.

Die beiden Hauptkomponenten, die Virtuelle Maschine[4] und die Evolutionsmaschine, bilden den Kern der verteilten Anwendung. Sie sind wie bereits erwähnt in der Application-Schicht angesiedelt.

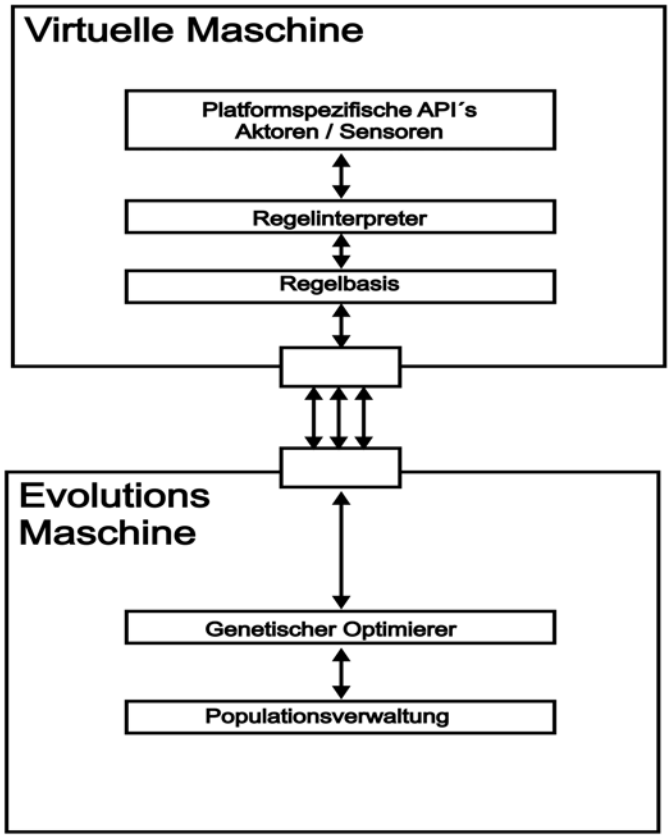


Abbildung 5.2: Hauptkomponenten

In der Abbildung sind die beiden Hauptkomponenten und ihre internen Funktionalitäten zum besseren Verständnis noch einmal abgebildet. In der Virtuellen Maschine sind die Ablaufsteuerung, der Regelinterpreter, die Fitnessfunktion, sowie die aktuell zu testende Regelbasis integriert. In der

[4] Eine Virtuelle Maschine ist allgemein ein Modell eines Prozesses und der dazugehörigen Systemarchitektur. Die Rechenweise wird dabei unabhängig von der technischen Realisierung beschrieben.

Evolutionsmaschine befindet sich der genetische Optimierer, sowie eine Populationsverwaltung, welche mit Hilfe einer Zufallsfunktion die Urpopulation erzeugt und während des evolutionären Prozesses die Population verwaltet. Die Regelbasenpopulation selbst wird dabei in der Persistence-Schicht gehalten. Beide Komponenten sind durch eine Schnittstelle miteinander verbunden. Über die Schnittstelle werden die zu testenden Genome und die Fitnesswerte, sowie einige Rahmenparameter ausgetauscht. Die Evolutionsmaschine ist somit die Komponente, auf welcher der evolutionäre Prozess stattfindet. Die weiter oben beschriebenen Methoden und Vorgehensweisen zur genetischen Optimierung werden auf Basis der von der Virtuellen Maschine zurückgegebenen Fitnesswerte durchgeführt. Im folgenden Abschnitt möchte ich die Funktionen der Virtuellen Maschine näher beschreiben.

5.2 Die Virtuelle Maschine

Die Virtuelle Maschine prüft die von der Evolutionsmaschine gesendeten Genome auf ihre Tauglichkeit. Der Regelinterpreter wertet dabei die durch die Genome repräsentierten Regelbasen aus und steuert so das Verhalten des Roboters. Der Regelinterpreter sorgt also für die Umsetzung der in der Regelbasis vorhandenen Anweisungen in die plattformspezifischen Motorbefehle. Über eine ebenfalls in die Virtuelle Maschine integrierte Fitnessfunktion wird die Fitness (Qualität) der aktuellen Regelbasis ermittelt. Die Virtuelle Maschine kann somit als verhaltensbildende Komponente gesehen werden, da in Ihr die Qualitätsbewertung des aktuellen Genoms stattfindet.
Die Auswertung der gewonnenen Fitnesswerte wird in der Evolutionsmaschine vorgenommen. In der Abbildung 5.3 ist der Informationsfluss innerhalb der Virtuellen Maschine noch einmal genauer dargestellt. Neben den zu testenden Genomen und den Fitnesswerten, werden außerdem einige Randparameter wie Zykluslänge und Ausführungsdauer zwischen den Komponenten ausgetauscht.

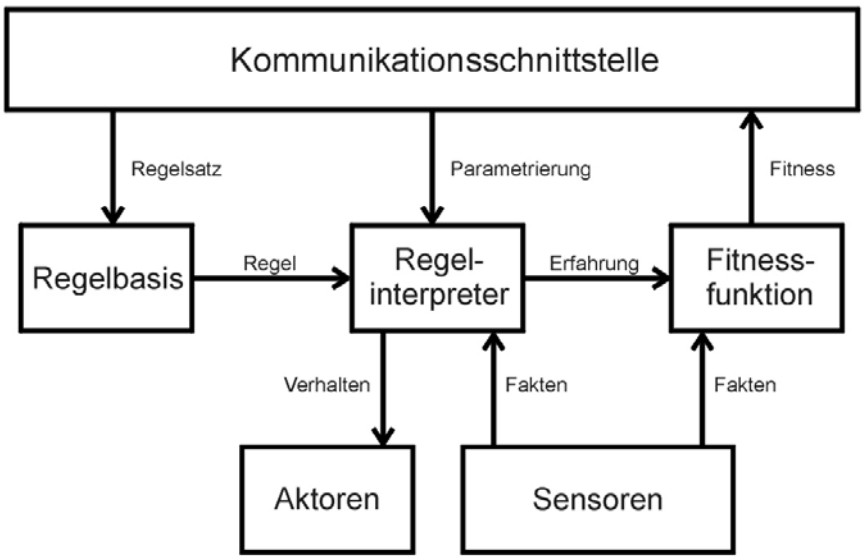


Abbildung 5.3: Informationsfluss - Virtuelle Maschine

5.2.1 Die Regelbasis

In der Regelbasis wird das Verhalten des Roboters gespeichert. Der Roboter kann auf Basis der vorhandenen Regeln auf Sinnesreize, die durch die Sensoren geliefert werden, reagieren. Zustände werden bei dieser Vorgehensweise nicht gespeichert. Die Handlungen des Roboters werden somit „reflexartig“ durch die Sinnesreize ausgelöst. Die Regelbasis wird als eine geordnete Menge von Implikationsregeln repräsentiert. Die Implikationsregeln werden aus den Prämissen, also der symbolischen Repräsentation der Sensorwerte und den Symbolen für die auszuführenden Aktionen zusammengesetzt.

Prämissen:	repräsentieren Sensorzustände
Aktionen:	repräsentieren Antriebszustände

Die Verknüpfungsfunktion der einzelnen Variablen wird vorgegeben, um den Suchraum zu verkleinern und eine zeitnahe Suche zu ermöglichen. Daraus folgt, dass Werte von verschiedenen Sensoren logisch miteinander verknüpft werden und ein eigenes Prämissensymbol zugeordnet bekommen. Alle zur Lösung des Problems sinnvollen Sensorverknüpfungen müssen dafür aufgestellt werden. Die Umwandlung der Sensordaten und der sich ergebenden Kombinationen in die entsprechenden Prämissen wird in einer Sensorfunktion, die Teil der Virtuellen Maschine ist, von einem Prämissenencoder vorgenommen.

Hier ein Beispiel zum besseren Verständnis: Die auf dem beispielhaften Roboter installierten Sensoren können eine Wand erkennen und dies durch Rückgabe eines TRUE oder FALSE Wertes signalisieren. Brauchbare, sich aus den Sensorkombinationen ergebende Prämissen, sind hier aufgeführt: Werden beide Sensoren ausgelöst, wird das Prämissensymbol BUMP zurückgeliefert. Das

Symbol BUMP steht in diesem Fall für eine Kollision mit der Wand. Registriert Sensor1 oder Sensor2 die Wand, kann eine Aussage getroffen werden, ob sich die Wand rechts (WALL RIGHT) oder links vom Roboter (WALL LEFT) befindet. Natürlich sind auch andere Verknüpfungsoperatoren möglich. Sinnvolle Kombinationen müssen im Vorfeld herausgefunden werden.

Sensor 1:	Logischer Operator:	Sensor 2:	Prämisse:
TRUE	AND	TRUE	BUMP
TRUE	AND	FALSE	WALL LEFT
FALSE	AND	TRUE	WALL RIGHT
FALSE	AND	FALSE	NO WALL

5.2.2 Das Genom

Die Regelbasen werden in den Genomen in Form von konstanten Vektoren gespeichert. Die Summe der vorhandenen Prämissen und der dazugehörigen Aktionen stellen die Anzahl der kleinsten teilbaren Einheiten (Gene) dar. Die einzelnen Implikationsregeln sind somit abgeschlossene Einheiten, die auch bei den Methoden der evolutionären Verfahren (Kreuzung, Mutation) erhalten bleiben müssen, um die strukturelle Integrität nicht zu verletzen. Zur Kreuzung und Mutation werden daher nur die in sich abgeschlossenen Gene herangezogen. Im Folgenden ist der generelle Aufbau eines Genoms dargestellt.

Genom (Regelbasis) in Form eines Vektors:

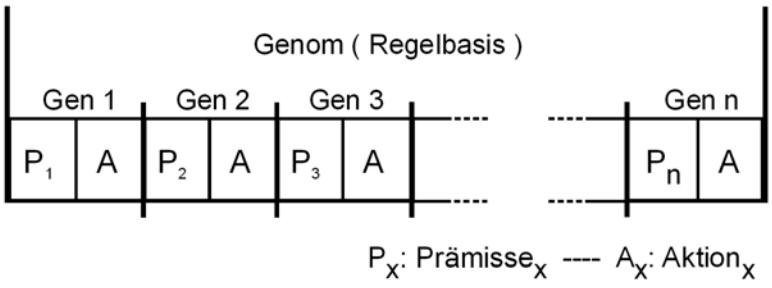


Abbildung 5.4: Genom - als linearer Vektor

5.2.3 Der Regelinterpreter

Der Regelinterpreter ist ebenfalls Teil der Virtuellen Maschine und verarbeitet die von der Evolutionsmaschine übergebenen Regelbasen. In ihm entsteht praktisch

das reaktive Verhalten des Roboters. Der Regelinterpreter liest die angeschlossenen Sensoren zyklisch aus und durchsucht das Regelwerk nach den von den Sensoren zurückgegebenen Prämissen. Ist die passende Prämisse gefunden, wird die entsprechende Aktion festgestellt und durch einen Decoder[5] in die plattformspezifischen Motorbefehle umgewandelt. Nach jedem Zyklus bewertet die Fitnessfunktion die Qualität der aktuellen Reaktion. Ist der Roboter dem gewünschten Ziel näher gekommen, oder hat sich die Situation eher verschlechtert? Zur Bewertung zieht die Fitnessfunktion wiederum die aktuellen Sensordaten heran. Die Kodierung der Sensordaten in die entsprechenden Prämissen wird von dem Prämissenencoder, der in einer Sensorfunktion integriert ist, übernommen. In der folgenden Abbildung ist der Ablauf der Regelinterpretation noch einmal verdeutlicht.

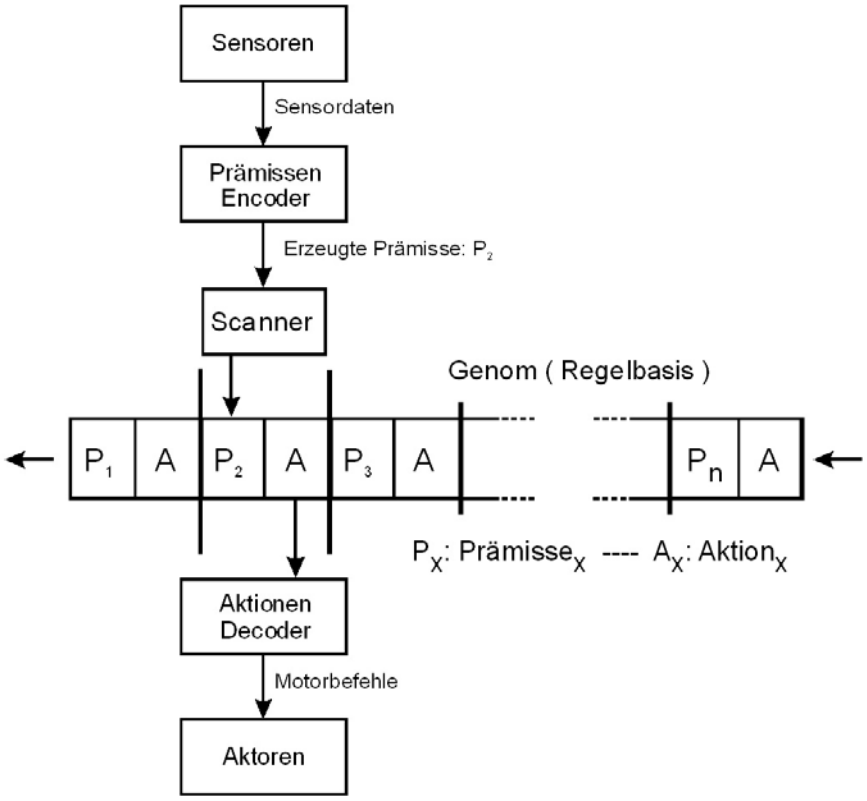


Abbildung 5.5: Interpretationsablauf - Regelinterpreter

Neben den zu testenden Regelbasen in Form der Genome, werden zwei weitere Parameter an den Regelinterpreter übergeben. Diese sind die Zyklusanzahl und die Ausführungsdauer jedes Zyklus. Die Zyklusanzahl bestimmt wie lange eine Regelbasis getestet werden soll. Dabei wird eine Rundenzahl an den Regelinterpreter übergeben, welche die Wiederholungen für den in Abbildung 5.5 gezeigten Vorgang angibt. Die Ausführungsdauer wiederum gibt an für welchen Zeitraum die gefundene Aktion ausgeführt werden soll. Die Ausführungsdauer

[5] Für die Decodierung der Aktionssymbole steht eine entsprechende Decoderfunktion zur Verfügung, welche die Symbole in die plattformspezifischen Motorbefehle umsetzt.

spielt für den Simulator keine Rolle, da der Weg welchen der Roboter bei Ausführung einer Aktion zurücklegt, durch die simulierte Welt vorgegeben ist. In der realen Roboteranwendung verhält sich dies anders. Hier gibt die Ausführungsdauer die Distanz pro Zyklus vor, die vom Roboter zurückgelegt werden kann.

Die Fitnessmessung findet nach jeder Runde statt. Die gewonnenen Fitnesswerte werden dabei aufsummiert und nach Fertigstellung eines kompletten Zyklus als Durchschnittswert zurückgegeben. Dieser Fitnesswert wird an die Evolutionsmaschine übertragen und dient dort dem genetischen Optimierer als Bewertungsgrundlage der aktuell durch die Virtuelle Maschine getesteten Regelbasis. Da sich die Fitnessfunktionen für den realen Roboterversuch und den Simulator stark unterscheiden, möchte ich erst in den entsprechenden Kapitel näher darauf eingehen. Die folgende Formel ist eine verallgemeinerte Formel zur Fitnessbestimmung.

$$\textit{Fitness (Genom}_x\textit{) = Fitness (Formel mit Summe über Fitness)}$$

Für den Fall, dass der Roboter während eines Zyklus an eine Arealbegrenzung stößt, sind zusätzliche Vorkehrungen nötig. Das Berühren einer Arealbegrenzung ist unerwünscht und lässt auf eine unbrauchbare Regelbasis schließen. Aus diesem Grund muss es innerhalb des Regelinterpreters einen übergeordneten Mechanismus geben, der dieses Verhalten bestraft und die Ausführung der aktuellen Regelbasis beendet. Für die Realtimeanwendung ist daher eine Subsumptionsarchitektur vorgesehen. Diese überwacht das Verhalten des Roboters und beendet bei Auslösen eines Bumpers den aktuellen Zyklus. Die Regelbasis, die sich als unbrauchbar erwiesen hat, wird dabei mit einem Fitnesswert von 0 abgestraft. Die Simulationssoftware überwacht die Arealbegrenzungen ebenfalls und bestraft Übertretungen der Arealgrenze auf gleiche Weise. Da bei der Simulation keine physikalische Welt vorhanden ist, ist eine Subsumptionsarchitektur hier nicht nötig.

5.2.4 Die Vorganssteuerung VM

Durch die Vorgangssteuerung wird der Ablauf zur Qualitätsbewertung der empfangenen Genome innerhalb der Virtuellen Maschine gewährleistet. Sie führt die von der übergeordneten Vorgangssteuerung der Evolutionsmaschine gesendeten Befehle aus und regelt die internen Abläufe. Die Vorganssteuerung sorgt außerdem für die Rückübertragung der erzeugten Fitnesswerte an die Evolutionsmaschine. Die Kommunikation wird über die bereitgestellten Kommunikationsmethoden ermöglicht.

Der Ablauf zur Bewertung einer empfangenen Regelbasis sieht folgendermaßen aus:

1. Warten auf neue Regelbasis
2. Warten auf neue Rahmenparameter (Zykluslänge, Anzahl der Zyklen usw.)
3. Regelbasis an den Regelinterpreter übergeben
4. Regelbasis für n Zyklen mit Ausführungsdauer t bewerten
5. Roboter in Ausgangsstellung bewegen
6. Erzeugte Fitnesswerte an Evolutionsmaschine senden

Die Vorgangssteuerung regelt somit den Betrieb der Virtuellen Maschine und führt die von der Evolutionsmaschine empfangenen Befehle aus.

5.3 Die Evolutionsmaschine

Die Evolutionsmaschine stellt die zweite wichtige Komponente dar, die ebenfalls in der Application-Schicht angesiedelt ist. In ihr sind ein genetischer Optimierer, sowie die Verwaltung der Genompopulation untergebracht. Die Evolutionsmaschine erzeugt mit Hilfe der integrierten Methoden eine zufällige Population von Regelbasen, überträgt diese einzeln an die Virtuelle Maschine und wertet die zurückgegebenen Fitnesswerte aus. Die zurückgegebenen Fitnesswerte dienen dem integrierten genetischen Optimierer als Grundlage für den Lernprozess. Auf Basis der Fitnesswerte wird bestimmt welche Individuen für den Optimierungsprozess in Frage kommen. Durch die bereits angesprochenen Selektions-, Kreuzungs-, und Mutationsvorgänge werden so Nachfolgegenerationen erzeugt. Die Evolutionsmaschine sorgt somit für den evolutionären Übergang von der Urpopulation in die Nachfolgepopulationen.

5.3.1 Die Urpopulation

Die Urpopulation für das genetische Verfahren wird mit Hilfe eines Zufallsgenerators, der in der Evolutionsmaschine integriert ist, erzeugt. Das zu erzeugende Genom ist dabei, was die einzelnen Genpositionen betrifft, fest vorgeschrieben. Jeder möglichen Sensorkombination für die eine Prämisse im Genom existiert, wird per Zufallsgenerator aus der Menge der definierten Aktionen ein Aktionssymbol zugewiesen. Wie sich später zeigen wird, ist die Qualität der Urpopulation ein wichtiger Faktor für den genetischen Prozess.

5.3.2 Das Abbruchkriterium

Das Abbruchkriterium entscheidet wann der evolutionäre Prozess beendet werden soll. Diese Entscheidung kann über einen vorgegebenen Fitnesswert oder die Anzahl der bereits erzeugten Generationen getroffen werden. Den Abbruch des Algorithmus über einen definierten Fitnesswert vorzunehmen kann dazu führen, dass der Vorgang in manchen Fällen sehr lange läuft, ohne ein entsprechendes Ergebnis zu finden. Da die Urpopulationen zufällig erzeugt

werden, birgt dies die Gefahr in sich, dass einige unbrauchbare Urpopulationen unverhältnismäßig lange laufen, bevor dies erkannt wird. Die Ausführungsdauer ist somit nicht voraussagbar. Weiterhin ist es schwierig einen Fitnesswert zu bestimmen, der einer ausreichenden Qualität zur Problembewältigung entspricht. In den meisten Fällen gibt es nicht genug Vorwissen um den Fitnesswert in Relation zum benötigten Ergebnis zu stellen. Da eine zeitliche Voraussage über die Ausführungsdauer gerade bei physischen Robotern in Anbetracht des Verschleißes der Bauteile eine Rolle spielt, habe ich mich für eine definierte Anzahl von Generationen als Abbruchkriterium entschieden. Eine festgelegte Anzahl von Generationen und somit eine definierte Ausführungsdauer führt zwar nicht immer zum Erfolg, verhindert aber effektiv, dass sich der Algorithmus durch eine schlechte Ausgangslage verrennt und bringt zumindest eine angenäherte Lösung hervor.

5.3.3 Der Optimierungsprozess

Der Optimierungsprozess stellt die Kernfunktion der Anwendung dar. Die Optimierung der Regelbasen erfolgt nach dem Grundschema von Koza [KOZA 1992]. Der Ablauf ist noch einmal im folgenden Pseudocode dargestellt:

1. Erzeugung einer n großen Urpopulation gefüllt mit zufälligen Aktionen
2. Prüfen ob Abbruchkriterium erfüllt ist (Generation Counter = X)
3. Stelle die Fitness für jedes Individuum in der Population fest (Virtuelle Maschine)
4. Zeichne Fitnessverlauf auf und sichere das soweit beste Genom
5. Erzeuge Nachfolgepopulation mit Hilfe der Transformationsmethoden gemäß Heiratsschema
6. Inkrementiere Generation Counter und mach weiter mit 2.

Nachdem eine Urpopulation erzeugt und nach Ihrer Fitness bewertet wurde, wird gemäß dem gewählten Heiratsschema und den Transformationsmethoden eine Nachfolgepopulation erzeugt. Das beste bis dahin gefundene Genom wird nach jedem Durchlauf gespeichert. Außerdem wird für die spätere Auswertung der beste Fitnesswert jeder Generation aufgezeichnet. Der Vorgang der Bewertung, Aufzeichnung und Generierung der Nachfolgepopulationen wird so lange wiederholt, bis das Abbruchkriterium zutrifft. Als Abbruchkriterium wird aus bereits genannten Gründen die Anzahl der Generationen herangezogen. Als Heiratsschema, zur Auswahl der Genome die für die Transformationsmethoden als Basis dienen, soll die Schöpfkellenmethode angewendet werden. Die Individuen der Nachfolgepopulation werden gemäß den definierten Wahrscheinlichkeiten der Transformationsmethoden gebildet.
Dies sind im Detail durch einfache Selektion in die Nachfolgepopulation kopierte, sowie durch Mutation und Kreuzung erzeugte Genome. Für den Fall, dass die Elitarismus-Funktion aktiviert ist, wird jeweils noch das bis dahin beste Genom in die Nachfolgegeneration übernommen. Ob das gewählte Heiratsschema und die gewählten Transformationsverfahren geeignet sind, soll sich im Simulatorexperiment herausstellen.

5.3.4 Die Transformationsmethoden

Die Transformationsmethoden und deren unterschiedliche Ausprägungen wurden bereits im Kapitel 4.6 vorgestellt. Da ich mit Genomen in Form von konstanten Vektoren arbeite, möchte ich hier noch einmal die ausgewählte Vorgehensweise für die Transformationsverfahren näher beschreiben.

Die Mutationsmethode:

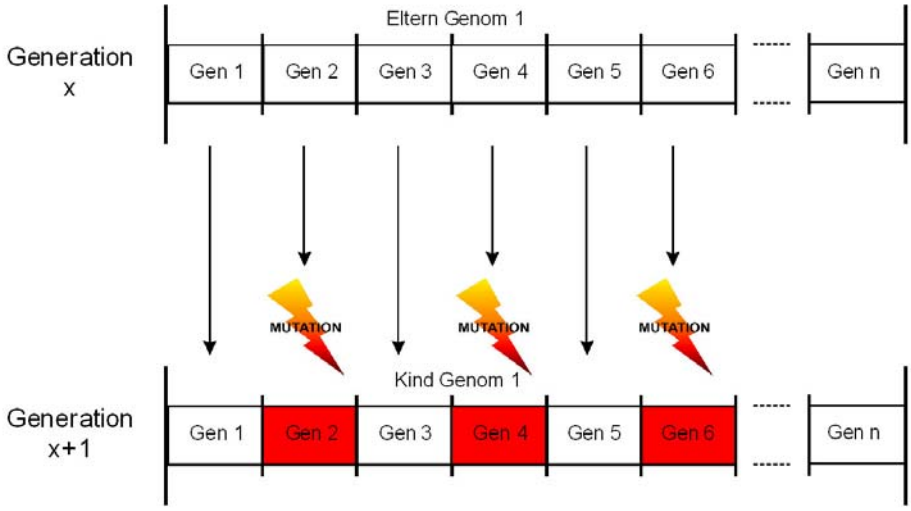


Abbildung 5.6: Schema - Mutationsmethode

Die Mutationsmethode soll ein ausgewähltes Genom dahingehend verändern, dass jeweils jedes zweite Gen innerhalb des Genoms mutiert, d.h. durch eine Zufallsaktion neu besetzt wird. Um eine größere Streuung der Zufallswerte zu erreichen, wird die Mutation wechselweise bei dem ersten und bei dem zweiten Gen vorgenommen.

Die Kreuzungsmethode:

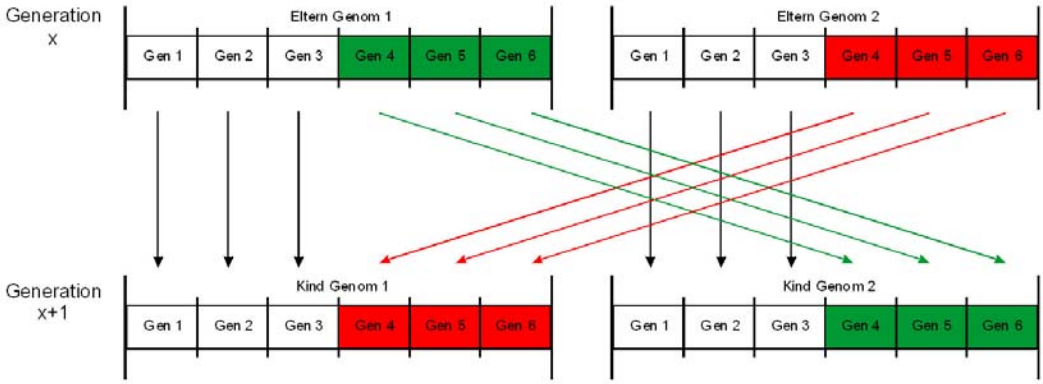


Abbildung 5.7: Schema - Kreuzungsmethode – Beispiel mit 6 Genen

Die Kreuzungsmethode soll nach folgendem Schema durchgeführt werden: Die zwei zu kreuzenden Genome werden exakt in der Mitte geteilt und eine Hälfte des ersten Genoms wird mit einer Hälfte des zweiten Genoms vertauscht. So entstehen zwei Kindgenome, die jeweils eine Hälfte der Elterngenome und eine unveränderte Hälfte enthalten.

5.3.5 Die Vorgangssteuerung EM

Die Vorgangssteuerung innerhalb der Evolutionsmaschine lenkt den Prozess für die genetische Optimierung und hält gleichzeitig die Kontrolle über die Vorgangssteuerung der Virtuellen Maschine. Die Virtuelle Maschine wird Befehle über die vorhandene Schnittstelle entgegen nehmen. Die Vorgangssteuerung der Evolutionsmaschine ist als übergeordnete Instanz geplant.

6 DER SIMULATOR

Die für den evolutionären Prozess nötigen Rahmenparameter sind, ebenso wie eine geeignete Fitnessfunktion, die ausschlaggebenden Faktoren für das Gelingen oder Scheitern des genetischen Algorithmus. Da es zu der gegebenen Problemstellung keine empirischen Daten gibt, ist die Parametrierung nur schwer vorzunehmen. Es gibt noch keine Erfahrungswerte, die für eine geeignete Parametrierung nötig wären. Da der Zeitverbrauch, insbesondere bei „langsamen“ Realtimeanwendungen ein nicht zu vernachlässigender Faktor ist, sollen geeignete Parameter mit Hilfe eines Simulators herausgefunden werden. Zeitverbrauch und Verschleiß spielen bei der Simulation keine große Rolle, so können aufgrund der geringeren Kosten mehr Versuche in kürzerer Zeit durchgeführt werden.

6.1 Die Evolutionsparameter

Die für den evolutionären Prozess nötigen Rahmenparameter und deren Bedeutung sind im Folgenden näher beschrieben:

- **Populationsgröße:**

  Gibt die Größe der Urpopulation an und bestimmt die Ausgangssituation für den genetischen Prozess. Eine zu klein gewählte Populationsgröße birgt die Gefahr in sich, dass sich kein brauchbares Genommaterial in der Population befindet. Eine zu groß gewählte Population kann eine sehr lange Ausführungsdauer zur Folge haben, ohne das ein merkbarer Qualitätsvorteil gegenüber der Ausgangspopulation zu spüren ist. Durch die zufällige Generierung der Urpopulation ist es schwer eine quantitative Aussage zu treffen, welche Populationsgröße sinnvoll ist, da die Qualität von Population zu Population höchstwahrscheinlich stark schwanken wird.

- **Anzahl der Generationen:**

  Die Anzahl der zu erzeugenden Nachfolgegenerationen legt das Abbruchkriterium für den genetischen Prozess fest. Ein zu klein gewählter Wert führt höchstwahrscheinlich zu keinem befriedigenden Ergebnis, da eine gewisse Ausführungsdauer benötigt wird, bis der evolutionäre Prozess greift. Ein zu groß gewählter Wert führt gerade bei der Realanwendung zu einer maßgeblichen Verlängerung der Ausführungsdauer und somit zu einer starken Belastung der Mechanik. Da die Generationenanzahl in der Simulation keine so starken

Auswirkungen auf den Zeitbedarf hat, soll hier ein günstiger Wert herausgefunden werden, bei dem der evolutionäre Prozess bereits greift.

- **Mutationswahrscheinlichkeit:**

  Die Mutationswahrscheinlichkeit gibt die Wahrscheinlichkeit für die Anwendung des Mutationsverfahrens für die Überführungen von einer Population in die Nachfolgepopulation an. Die Mutations-wahrscheinlichkeit muss behutsam gewählt werden, da ein zu klein oder zu groß gewählter Wert die Qualität des Evolutionsprozesses stark beeinflussen kann. Ein zu klein gewählter Wert kann unter Umständen nicht verhindern, dass der Evolutionsprozess in einem lokalen Maximum stecken bleibt und so evtl. bessere Lösungen nicht gefunden werden. Ein zu groß gewählter Wert hingegen kehrt den positiven Effekt der gelegentlichen Mutation um und kann die systematische Durchkreuzung des Lösungsraumes behindern, da zu viele willkürliche Sprünge unternommen werden.

- **Crossoverwahrscheinlichkeit:**

  Die Crossoverwahrscheinlichkeit gibt die Wahrscheinlichkeit für die Anwendung des Kreuzungsverfahrens während der Überführung von einer Population in die Nachfolgepopulation an. Dieser Parameter bestimmt, wie viele Individuen in der neuen Population durch Kreuzung erzeugt werden sollen. In der Regel wird ein Grossteil (ca. 80 % – 90 %) der neu zu erzeugenden Individuen durch die Kreuzungsmethode erstellt.

- **Selektionswahrscheinlichkeit: (Direkte Migration)**

  Die Selektionswahrscheinlichkeit gibt die prozentuale Wahrscheinlichkeit für die direkte Migration einzelner Individuen aus der aktuellen Population in die Nachfolgepopulation an. Durch die Selektionsmethode ausgewählte Individuen werden eins zu eins in die Nachfolgepopulation übernommen, ohne dass diese eine Veränderung des Genoms erfahren.

- **Elitism:**

  Dieser Parameter gibt an, ob das beste Genom, also das Genom mit der besten Fitness einer Generation, unverändert in die Nachfolgepopulation übernommen werden soll. Durch diese Option kann sichergestellt werden, dass die Fitness von Generation zu Generation in ungünstigen Fällen nicht mehr stagnieren kann.

- **Anzahl der Lebenszyklen pro Genom:**

  Gibt die Anzahl der Lebenszyklen an, für die ein Genom im Regelinterpreter getestet wird. Dieser Wert gibt dem Roboter indirekt eine Reichweite innerhalb seiner Welt vor. Für jeden Zyklus ist der Roboter in der Lage sich ein Stück zu bewegen. Zu klein gewählte Werte können evtl. verhindern, dass eine Aussage über die Qualität der aktuellen

Regelbasis getroffen wird, da sie sich zeitbedingt nicht bewähren konnte. Ein zu groß gewählter Wert hingegen könnte den evolutionären Prozess unnötig in die Länge ziehen, da ungeeignete Regelbasen frühzeitiger erkannt werden könnten.

6.2 Design und Implementierung

Wie im vorausgegangenen Kapitel besprochen, sollen vor dem Hardwareversuch mit Hilfe eines Simulators praktikable Rahmenparameter für den Algorithmus herausgefunden werden. Da die Simulation der benutzten Roboterplattform ohne Vorwissen über das genaue Verhalten nicht möglich ist, soll zur Evaluierung der geeigneten Parameter ein Linefollow-Roboter simuliert werden. Der Softwareroboter ist mit drei Helligkeitssensoren ausgestattet, die an seiner Front installiert sind. Des weiteren verfügt er über einen Bumpersensor, der das unerwünschte Verlassen der gegebenen Spielfläche erkennt. Die Aufgabe des Roboters besteht darin, einer vorgegebenen Kreisbahn in Form einer dunklen Markierung zu folgen. Über die drei Helligkeitssensoren kann er dabei die vorgegebene Linie erkennen.

Um den Code möglichst portabel zu halten, wird der Simulator in der Zielsprache der Roboterplattform, der Programmiersprache C geschrieben. Die Portierung der Virtuellen Maschine und des Evaluators auf das spätere Realsystem soll so ohne größeren Aufwand ermöglicht werden. Die Kommunikation zwischen den Komponenten wird durch Parameterübgaben realisiert.

6.2.1 Die simulierte Umwelt

Die simulierte Umwelt besteht aus einem Raster, das 10 * 10 Einheiten groß ist. Der Roboter belegt auf dem Spielraster genau eine Einheit. In der folgenden Abbildung ist das Areal aus Sicht des simulierten Roboters und seine definierte Startposition abgebildet. Anhand der Position des Roboters und der dazugehörigen Sensoren ist die aktuelle Ausrichtung des Roboters zu erkennen. Es soll immer in Blickrichtung Norden gestartet werden.

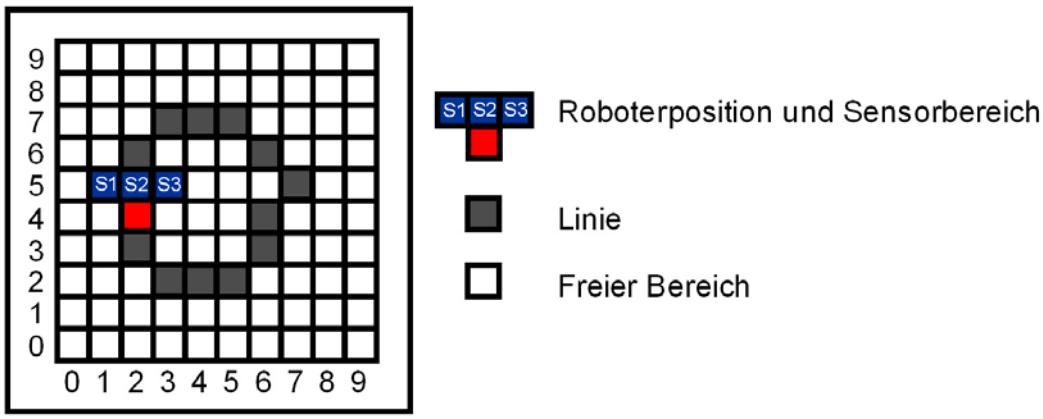


Abbildung 6.1: Simulierte Umwelt

6.2.2 Die Sensoren und Aktoren

Der Softwareroboter verfügt genau wie die reale Roboteranwendung über einen omnidirektionalen Antrieb. Das bedeutet, dass auch er in der Lage ist, ohne sich explizit drehen zu müssen, jede gewünschte Richtung anfahren zu können. Mit jedem Fahrkommando bewegt sich der Roboter ein Feld in die angegebene Richtung. Aufgrund der Beschränkungen des Spielrasters stehen folgende Freiheitsgrade zur Verfügung.

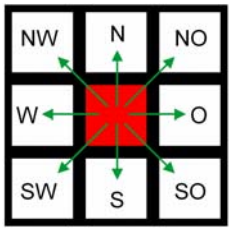


Abbildung 6.2: Freiheitsgrade Roboter

Aus diesen Freiheitsgraden ergeben sich somit acht mögliche Fahrtrichtungen, die an die Funktion "Fahre" übergeben werden können. Die Fahrtrichtungen werden dabei relativ zur Ausrichtung des Roboters angegeben. Die Kodierung der Motorbefehle wird bewusst in Form von reellen Zahlen vorgenommen, um den Aufwand für die Programmierung des Regelinterpreters zu minimieren. Die Kodierung der Genome kann so einfach in Form von Vektoren, gefüllt mit reellen Zahlen vorgenommen werden. Für die Regelbasis werden die Fahrtrichtungen auf folgende Aktionssymbole abgebildet:

Fahrtrichtung	Aktionssymbol
Gerade aus	0
Schräg rechts vorwärts	1
Rechts	2
Schräg rechts rückwärts	3
Rückwärts	4
Schräg links rückwärts	5
Links	6
Schräg links vorwärts	7

Tabelle 6.1: Simulator Aktionssymbole

Die simulierte Umgebung und die Zustandsvariablen werden in der Anwendung von einer Matrix gehalten, die über verschiedene Funktionen manipuliert werden kann. Folgende Informationen werden in der Matrix gespeichert.

- Verlauf der Linie
- Roboterposition
- Ausrichtung des Roboters

Das Bewegen innerhalb des Spielrasters wird über eine Funktion „Fahre(Fahrtrichtung)“ realisiert. Über eine Funktion „ZeichneMatrix()“ kann das Spielraster auf der Konsole ausgegeben werden. Die Funktion „Fahre()“ gibt im Falle einer Kollision mit der Arealbegrenzung eine Meldung zurück. Diese Bumperinformation wird im Regelinterpreter ausgewertet und unterbricht den aktuellen Bewertungszyklus. Bei Auslösung des Bumpersignals wird das aktuelle Genom mit einem Fitnesswert Null bestraft.

Die Sensoren des simulierten Roboters können über eine vorhanden Funktion „Sensor()“ ausgelesen werden. Die Funktion liest die aktuelle Position und Richtung des Roboters in der Matrix aus und gibt drei Sensorwerte S1 – S3 zurück. Je nach Stellung des Roboters werden dabei die drei umliegenden Felder berücksichtigt. Als Sensordaten wird einfach die in der Matrix angegebene, relativ zu Ausrichtung und Position des Roboters befindliche, Linie zurückgegeben. Je nach Ausrichtung des Roboters werden die sich in Fahrtrichtung befindlichen Sensoren S1 – S3 ausgelesen.

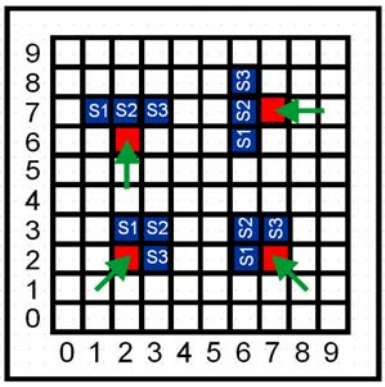


Abbildung 6.3: Simulator - Roboter Sensoranordnung

6.2.3 Der Prämissenencoder

Der Prämissenencoder sorgt für die Kodierung der 3 Sensorwerte in die entsprechenden Prämissensymbole und ist in die Sensorfunktion eingebettet. Die Sensorenfunktion ermittelt aus der Matrix, je nach dem ob eine Linie erkannt wurde oder nicht, die Werte True oder False. Daraus ergeben sich 2^3 = 8 mögliche Sensorkombinationen, die durch Prämissen abgebildet werden müssen.

Die folgende Tabelle legt die Prämissen für jede möglich Sensorkombination fest:

Sensor 1	Sensor 2	Sensor 3	Prämisse	Beschreibung
TRUE	TRUE	TRUE	1	Roboter steht im 90° Winkel vor Linie
TRUE	TRUE	FALSE	2	Roboter steht schräg vor Linie
TRUE	FALSE	TRUE	3	Roboter steht auf Rundung der Linie
TRUE	FALSE	FALSE	4	Linie befindet sich Links vom Roboter
FALSE	TRUE	TRUE	5	Roboter steht schräg vor Linie
FALSE	TRUE	FALSE	6	Roboter direkt auf Linie
FALSE	FALSE	TRUE	7	Linie befindet sich Rechts vom Roboter
FALSE	FALSE	FALSE	8	Keine Linie in Sicht

Tabelle 6.2: Simulator Sensorsymbole

6.2.4 Das Genom (Regelbasis)

Die Regelbasen setzen sich aus den möglichen Prämissen und den auszuführenden Aktionen zusammen. Aus der Anzahl der Prämissen leiten sich

somit 8 Gene innerhalb des Genoms ab, die mit zufälligen Aktionen besetzt werden. Die Regelbasenpopulation wird dabei in Form von mehrdimensionalen Arrays gespeichert.

6.2.5 Die Fitnessfunktion

Der Roboter soll erlernen, sich in seinem Parkcour auf der vorgegebenen Linie fortzubewegen. Um dieses Ziel zu erreichen, muss er die von den Helligkeitssensoren zurückgegebenen Informationen auswerten und in sinnvolle Aktionen in Form von Motoranweisungen umsetzten. Bei der Suche nach einer brauchbaren Verhaltensweise bildet die Fitnessfunktion die Bewertungsgrundlage, die das aktuelle Verhalten quantitativ feststellt. Die Fitnessfunktion ist somit ein zentrales Element, von deren Qualität eine erfolgreiche Lösungsfindung abhängt.
Damit der genetische Algorithmus erfolgreich sein kann, muss das Ziel ausreichend genau in der Fitnessfunktion kodiert werden. Um eine brauchbare Fitnessfunktion zu kodieren, ist einiges Vorwissen über die Problemstellung nötig. Die Abbildung des zu erreichenden Zieles auf quantitative Werte ist in der Regel nicht einfach, da kein ausreichendes Vorwissen und kaum Erfahrungswerte vorhanden sind. Wie nun kann das Linefollow Problem kodiert werden?

Für die Bewertung der vorgenommenen Aktion sollen die aktuellen Sensordaten herangezogen werden. Dabei können die aufgenommenen Sensordaten in gewünschte und eher unerwünschte Sensordaten eingeteilt werden. Oder anders ausgedrückt, die aktuelle Position des Roboters in Relation zur Linie auf dem Boden wird als gut oder weniger gut bewertet. Da ein PC mit den Begriffen gut oder weniger gut nicht umgehen kann, sollen den einzelnen Situationen, die als bewertungswürdig in Betracht kommen, Werte zugeordnet werden. Dabei wird festgelegt, dass desto erfolgreicher eine Aktion war, desto höher ist der erreichte Fitnesswert und umgekehrt.

Folgende Sensorwerte und die daraus resultierenden Roboterstellungen relativ zur Linie, sollen zur Bewertung herangezogen werden:

S1	S2	S3	Qualitative Bewertung	Quantitative Bewertung	Beschreibung
0	1	0	Sehr gut	100	Roboter befindet sich genau auf der Linie
1	0	0	Gut	50	Roboter befindet sich rechts neben der Linie
0	0	1	Gut	50	Roboter befindet sich links neben der Linie
1	1	1	Ausreichend	20	Roboter steht direkt vor der Linie
0	1	1	Schlecht	5	Roboter steht schräg vor der Linie
1	1	0	Schlecht	5	Roboter steht schräg vor der Linie
1	0	1	Schlecht	5	Roboter steht direkt vor einer Krümmung der Linie
0	0	0	Sehr schlecht	0	Der Roboter hat die Linie aus der Sicht verloren

Tabelle 6.3: Simulator Bewertungsschema

Die Bewertung einer Regelbasis geht über mehrere Zyklen. Nach jedem Zyklus wird die aktuelle Situation bewertet. Die so gewonnenen Fitnesswerte werden aufsummiert. Der endgültige Fitnesswert einer Regelbasis ergibt sich dann aus dem durchschnittlichen Verhalten über die Anzahl der Zyklen.

Overall Fitness = Summe der Fitnesswerte / Anzahl der Zyklen

Beispiel:

Regelbasis wird für fünf Zyklen bewertet:

Zyklus 1:	Fitness:	100
Zyklus 2:	Fitness:	5
Zyklus 3:	Fitness:	50
Zyklus 4:	Fitness:	100
Zyklus 5:	Fitness:	20
Zyklus Ende:	Fitnesssumme:	275

Daraus ergibt sich für die getestete Regelbasis eine Fitness von:

Overall Fitness: 275 / 5 = 55

Mit dieser Vorgehensweise kann jetzt die Fitness einer Regelbasis quantitativ festgestellt werden. Allerdings gibt es noch einen Ausnahmefall, der außerhalb der Regelinterpretation abgedeckt werden muss. Für den Fall, dass eine

Regelbasis komplett unbrauchbar ist und der Roboter die Linie verlässt und vor Ablauf des Zyklus gegen eine Bande fährt, müssen weitere Maßnahmen getroffen werden. Die Randbegrenzungen der Welt können bei Kollision durch einen Bumper, der in der Robotersimulation integriert ist, erkannt werden. Die Bumperfrage nimmt zum Schutz des Roboters eine übergeordnete Rolle ein und befindet sich somit außerhalb des Wirkungsbereichs der Regelinterpretation. Dieses übergeordnete Verhalten ist direkt in der Virtuellen Maschine implementiert.

Fährt der Roboter während einer Bewertungsphase gegen die Randbegrenzung, kann davon ausgegangen werden, dass die aktuelle Regelbasis unbrauchbar ist. Die Bumperauslösung führt somit zum Abbruch der aktuellen Regelinterpretation. Die unbrauchbare Regelbasis wird mit einem Fitnesswert 0 abgestraft.

Ob und wie weit das Problem ausreichend in der Fitnessfunktion kodiert ist, werden die ersten Durchläufe in der Simulation ergeben. Werden sich durch die Vorgaben in der Fitnessfunktion die gewünschten Verhaltensweisen bilden? Eine evtl. Verbesserung der Fitnessfunktion ist eingeplant.

6.2.6 Die Benutzerschnittstelle

Um eine möglichst hohe Portabilität des Codes in Hinsicht auf das Robotersystem zu erlangen, wurde bewusst auf ein programmiertechnisch aufwendiges, grafisches Interface verzichtet. Die Rahmenwerte für den evolutionären Prozess sollen der Anwendung per Parameterübergabe zur Verfügung gestellt werden. Die Ausgabe des Verlaufs und der erzeugten Ergebnisse werden ebenfalls über die Konsole erfolgen. Um den genauen Verlauf der genetischen Optimierung auswerten zu können, ist geplant die Ausgaben der Anwendung in eine Textdatei umzuleiten[6]. Folgende Optionen und Parameter können über die Kommandozeile an das Programm übergeben werden:

[6] Dies ist unter Microsoft Windows mit Hilfe des Umleitungsoperators „<“ möglich.

Option:	Mögliche Parameter:	Beschreibung:
-Help	Keine	Gibt einen Hilfebildschirm mit den möglichen Optionen und Parametern aus.
-PopSize	5 – 1000	Gibt die Populationsgröße an. Sie muss zwischen 5 und 1000 gewählt werden.
-LifeCycles	1 – 70	Gibt die Anzahl der Lebenszyklen die jedem Genom zur Verfügung stehen an. Sie muss zwischen 1 und 70 gewählt werden.
-TimeToLife	10 ms – 5000 ms	Gibt die Ausführungsdauer der gefundenen Aktion im Regelinterpreter an. Dieser Parameter kommt in der Simulation nicht zum Einsatz.
-Mutation	1% - 98%	Gibt die Mutationswahrscheinlichkeit in Prozent an.
-Crossover	1% - 98%	Gibt die Kreuzungswahrscheinlichkeit in Prozent an.
-Generations	5 – 1000	Gibt die Anzahl der zu erzeugenden Generationen an und dient als Abbruchkriterium.
-Elite	1 oder 0	Gibt an ob die Elitarismusfunktion aktiviert werden soll.
-Fitness	1 oder 0	Wählt eine von zwei möglichen Fitnessfunktionen aus

Tabelle 6.4: Simulator Software Optionen

Hier ein Beispiel für die Parametrierung der Simulationssoftware:

gpopt.exe –PopSize 15 –LifeCycles 15 –Generations 100 –Elite 1 –Mutation 10 –Crossover 80

Die Gesamtwahrscheinlichkeit für die Auswahlverfahren wird aus den Einzelwahrscheinlichkeiten der Kreuzungs-, Mutations- und Selektionsmethode zusammengesetzt. Die Angaben zur Wahrscheinlichkeit für das Auftreten der Mutations- und Kreuzungsmethode dürfen zusammen genommen 99% nicht überschreiten, da die Restwahrscheinlichkeit der Selektionsmethode zugeschrieben wird. Auf obiges Beispiel angewendet bedeutet dies für die einzelnen Wahrscheinlichkeiten:

- Mutationswahrscheinlichkeit 10 %
- Kreuzungswahrscheinlichkeit 80 %
- Rest aus Selektion 10%

Ergibt zusammen eine Gesamtwahrscheinlichkeit von 100%.

Abbildung 6.4: Simulator Hilfebildschirm

In der Abbildung ist der Hilfebildschirm der Anwendung zu sehen. Alle Parameter können über die Eingabezeile an das Programm weitergereicht werden. Zur Auswertung eines Durchlaufes gibt die Anwendung verschiedene Parameter aus. Diese sind im einzelnen:

- **Parameterübsicht genetischer Prozess:**

  Zur Dokumentation und Übersicht werden alle eingestellten Rahmenparameter noch einmal ausgegeben, um eine komplette Versuchsdokumentation zu erhalten. (Populationsgröße, Anzahl der Generationen, Lebenszyklen pro Genom, Mutationswahrscheinlichkeit, Kreuzungswahrscheinlichkeit, Direkte Emigration, Elitarismus Option, Verbesserte Fitnessfunktion)

- **Urpopulation:**

  Die durch den Zufallsgenerator erzeugte Urpopulation, die als Basis für den evolutionären Prozess dient, wird für Kontrollzwecke komplett ausgegeben.

- **„Best of Fitness“ für jede Generation:**

  Die beste Fitness einer jeden Generation wird aufgezeichnet und als Liste ausgegeben.

- **Fitnessverlauf der gesamten Evolution:**

  Alle Fitnesswerte die ermittelt werden, werden aufgezeichnet und als Liste ausgegeben.

- **Verhalten der Regelbasen:**

  Jede Regelbasis durchläuft innerhalb der Virtuellen Maschine den Prüfungsprozess zur Fitnessbestimmung. Die dabei ausgeführten Motorbefehle werden zu Kontrollzwecken für jede Regelbasis ausgegeben. Hierbei werden auch evtl. untaugliche Regelbasen durch „-Bumper ausgelöst-„ kenntlich gemacht. Auf eine Ausgabe der Spielfläche wird hierbei bewusst verzichtet, da je nach Populationsgröße und Generationenanzahl eine zu große Datenmenge für eine praktikable Handhabung der Daten zur Auswertung entstehen würde.

- **Ergebnis:**

  Nachdem das Abbruchkriterium erreicht ist, wird das Ergebnis des Durchlaufs festgestellt. Hierzu wird das beste gefundene Genom und dessen Fitnesswert ausgegeben. Um das Ergebnis beurteilen zu können, wird das Genom noch einmal getestet und dessen Verhalten als Bewegung auf der Spielfläche ausgegeben. Hier kann jetzt das Verhalten genauer betrachtet und beurteilt werden.

Einige exemplarische, von der Simulatorsoftware generierte Ausgaben, sind im Anhang dieser Arbeit zu finden.

6.3 Versuchsdurchführung

Mit Hilfe der Simulationssoftware sollen auf experimentellem Wege geeignete Rahmenparameter (siehe Kapitel 6.1) für den evolutionären Prozess herausgefunden werden. Dies ist eines der primären Ziele der Simulationssoftware. Die ermittelten Parameter sollen im Roboterversuch zum Einsatz kommen. Diese Vorgehensweise minimiert die kritischen Kostenfaktoren Zeit und Verschleiß auf Roboterseite und erhöht somit gleichzeitig die Chancen, dass der Algorithmus in einer angemessenen Zeit zu einer brauchbaren Lösung führt. Weiterhin soll ermittelt werden, ob die Zielvorgabe des simulierten Roboters, sich auf der vorgegebenen Kreisbahn zu bewegen, ausreichend genau in der erstellten Fitnessfunktion kodiert ist. Im Laufe des Versuchs wird sich zeigen, ob diese evtl. verfeinert werden muss.

Die Versuchsreihe teilt sich in 4 Versuchsdurchläufe auf. Die Versuche unterscheiden sich in den eingestellten Rahmenparametern. Mit Hilfe der von der Software erstellten Logdateien, sollen die unterschiedlichen Parameter bewertet werden. Hier ist im einzelnen zu klären, wie sich die unterschiedliche Verteilung der anzuwendenden Kreuzungs-, Mutations- und Selektionsverfahren auswirkt und ob die gewählte Fitnessfunktion geeignet ist.

6.3.1 Der 1. Versuch

Die Versuchsparameter:

Der erste Versuch soll mit einer hohen Mutations- und Kreuzungswahrscheinlichkeit, jeweils 40%, für die zu erzeugenden Nachfolgegenerationen gestartet werden. Auf die direkte Selektion entfallen somit lediglich 20% in den Nachfolgegenerationen. Da es primär um die Findung der Kreuzungs- und Mutationsparameter geht, sollen die Populationsgröße und die Anzahl der zu erzeugenden Generationen in jedem der Versuche gleich groß gewählt werden. So ist eine bessere Beurteilung möglich, welche Auswirkungen die unterschiedlichen Gewichtungen auf die Mutations- und Selektionsmethode haben. Die Elitarismusfunktion, also die Übernahme des besten gefundenen Genoms in die Nachfolgegeneration, soll in allen Versuchen eingeschaltet sein, um eine Stagnation der Fitnesswerte zu verhindern.

Die Parameter des 1. Versuchs:

Parameter:	**Wert:**
Fitnessfunktion mit Streckenmessung	Aus
Populationsgröße	15
Anzahl der Generationen	100
Elitarismusfunktion	Ein
Lebenszyklen pro Genom	20
Kreuzungswahrscheinlichkeit	40%
Mutationswahrscheinlichkeit	40%
Direkte Selektion	20%

Tabelle 6.5: Simulation Parameter Versuch 1

Ergebnisse des ersten Versuchs:

In der folgenden Abbildung ist die in Kapitel 6.2.1 eingeführte simulierte Umwelt des Roboters abgebildet. Die grauen Kästchen bilden die vom Roboter nachzufahrende Strecke, die roten Kästchen wiederum zeigen an welche Strecke von der gefundenen Regelbasis zurückgelegt wurde. Der Startpunkt des Roboters ist mit „S“ markiert. Die aufsteigenden Zahlen zeigen den vom Roboter zurückgelegten Weg an. Sollten einzelne Felder doppelt angefahren werden, wird dies durch einen grünen Pfeil kenntlich gemacht.

Verhalten der gefundenen Regelbasis mit einem Fitnesswert von 40:

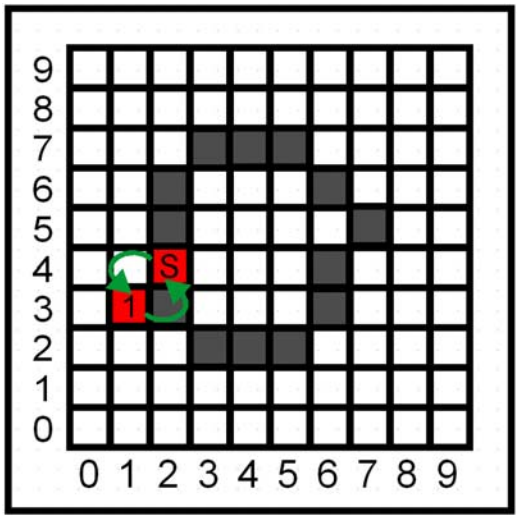


Abbildung 6.5: Versuch 1 - Bewegungsmuster des besten Genoms

Wie in der Abbildung 6.5 zu erkennen ist, zeigt die im ersten Versuch gefundene Regelbasis, in Bezug auf das gewünschte Verhalten der Linie zu folgen, eine mangelhafte Ausprägung. Der Roboter wechselt über die Dauer der zu durchlaufenden Zyklen zwischen dem Startpunkt „S“ und dem mit „1“ markiertem Feld, dass sich neben der zu befahrenden Linie befindet. Das Ziel der Linie zu folgen wurde somit nicht erreicht.

Analyse der Versuchsergebnisse:

In dem nachfolgenden Chart ist eine grafische Analyse des Fitnessverlaufes über die erzeugten Generationen zu sehen:

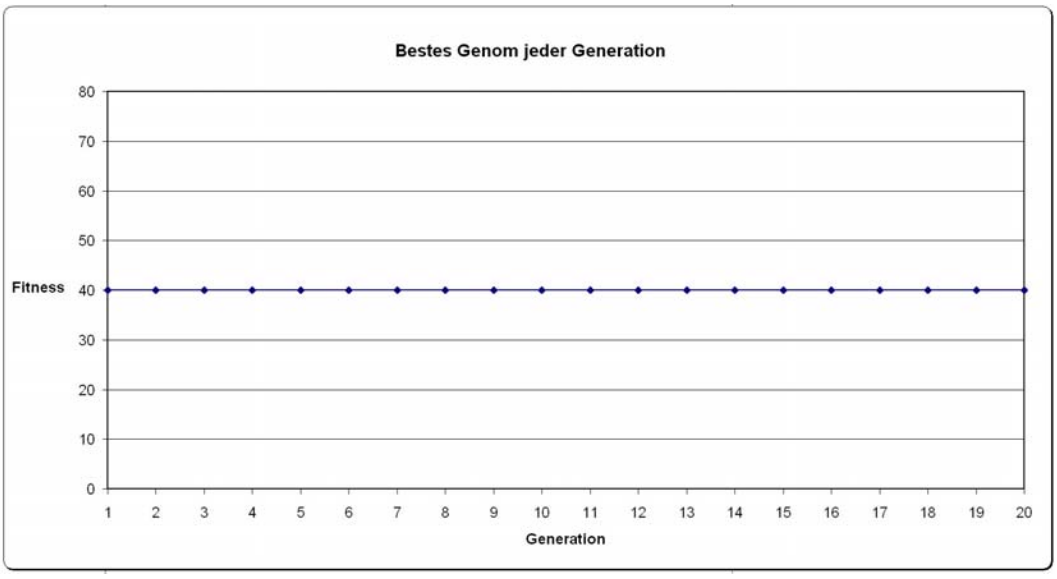


Abbildung 6.6: Versuch 1 - Bestes Genom jeder Generation

Wie in dem Chart zu erkennen, hat sich die Fitness der Individuen, trotz voranschreitender Generationenzahl, nicht verbessert. Das beste Genom wurde in der zufällig erzeugten Urpopulation gefunden. Es hat lediglich einen Fitnesswert von 40 erreicht. Die Analyse der Urpopulation und des besten gefundenen Genoms zeigt, dass das gefundene Genom direkt aus der Urpopulation stammt. Da sich die Fitness der neu erzeugten Individuen nicht über 40 Fitnesspunkte hinausheben konnte, wurde das beste Genom aus der Urpopulation durch den Elitemechanismus in die Nachfolgegenerationen übernommen und später als Ergebnisgenom präsentiert. Da sich die Fitnesswerte über die Generationen nicht verbessert haben, muss davon ausgegangen werden, dass der genetische Prozess nicht gegriffen hat.

In dem folgenden Chart ist eine grafische Analyse der durchschnittlichen Fitness aller Genome pro Generation aufgeführt:

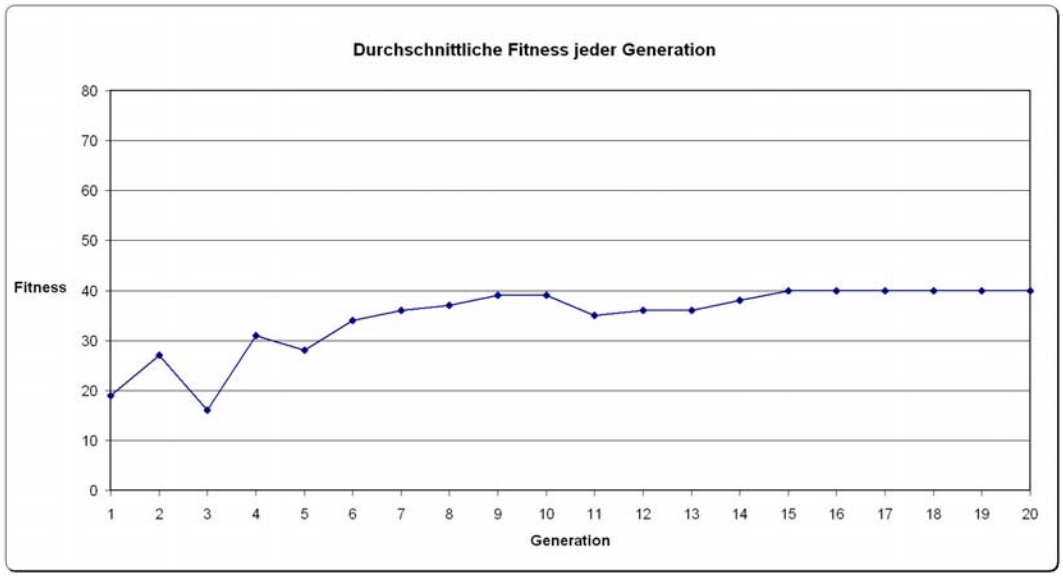


Abbildung 6.7: Versuch 1 - Durchschnittliche Fitness

In den ersten sechs Generationen ist eine gewisse Varianz der durchschnittlichen Fitnesswerte zu erkennen. Ab der sechsten Generation steigt die durchschnittliche Fitness wieder leicht an und erreicht im Schnitt einen Fitnesswert von 40 Punkten. Ab der 15 Generation ist keinerlei Varianz mehr zu erkennen, der Algorithmus hat sich „festgefahren“.

Die Analyse des besten gefundenen Genoms zeigt, dass die Aktionen zur Lösung des Problems (der Linie zu folgen), nicht ausreichend genau auf die Prämissensymbole abgebildet wurden.

Das gefundene beste Genom:

Gen Nr:	Prämissensymbol:	Aktionssymbol:
0	0	4
1	1	4
2	2	5
3	3	1
4	4	4
5	5	3
6	6	4
7	7	3

Tabelle 6.6: Simulator Versuch 1 - Bestes Genom

Schlussfolgerung:

Eine Verbesserung der aus der Urpopulation gewonnenen Genome konnte nicht beobachtet werden. Aus den in der Analyse gewonnenen Daten lässt sich schließen, dass der genetische Prozess mit den gewählten Einstellungen nicht „greifen“ konnte. Evtl. ist einer der Parameter (Mutationswahrscheinlichkeit oder Kreuzungswahrscheinlichkeit) zu hoch gewählt. Um festzustellen, ob diese These zutrifft, soll in den nächsten Versuchen einer der Parameter niedriger gewählt werden.

6.3.2 Der 2. Versuch

Die Versuchsparameter:

Der 1. Versuch hat ergeben, dass der genetische Prozess bei einer 40% Wahrscheinlichkeit für die Mutations- und Kreuzungsmethode nicht zu dem gewünschten Ergebnis geführt hat. Um das Problem näher eingrenzen zu können, soll im nächsten Schritt die Kreuzungswahrscheinlichkeit auf 5% verringert werden. Alle anderen Parameter sollen, um einen Vergleich zu ermöglichen, unverändert bleiben.

Die Parameter des 2. Versuchs:

Parameter:	**Wert:**
Fitnessfunktion mit Streckenmessung	Aus
Populationsgröße	15
Anzahl der Generationen	100
Elitarismusfunktion	Ein
Lebenszyklen pro Genom	20
Kreuzungswahrscheinlichkeit	5%
Mutationswahrscheinlichkeit	40%
Direkte Selektion	55%

Ergebnisse des 2. Versuchs:

Das im 2. Versuch gefundene Genom zeigt ein leicht komplexeres Verhalten, als das Genom aus dem 1. Versuch. Die Regelbasis steuert den Roboter, wie in der Abbildung zu erkennen, auf einem Dreieckkurs um den Startpunkt. Das Ziel der Linie zu Folgen und den Parcours zu umrunden wurde somit nicht erreicht.

Gefundene Regelbasis mit einem Fitnesswert von 42:

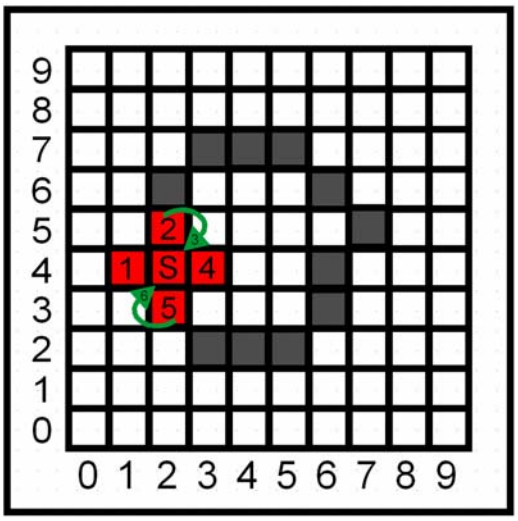


Abbildung 6.8: Versuch 2 - Bewegungsmuster des besten Genoms

Analyse der Versuchsergebnisse:

Grafische Analyse der durchschnittlichen Fitnesswerte pro Generation:

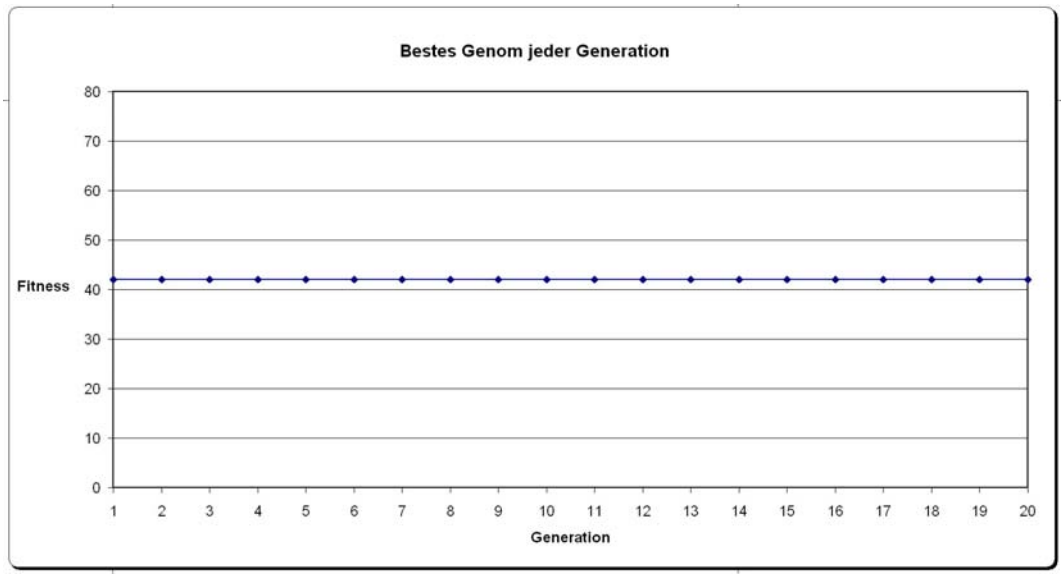


Abbildung 6.9: Versuch 2 - Bestes Genom jeder Generation

Die Grafische Analyse der besten Fitnesswerte pro Generation hat ergeben, dass die Fitness, wie auch schon im vorangegangen Versuch, nicht gesteigert werden konnte. Das beste Genom wurde aus der Urpopulation ermittelt und hat einen Fitnesswert von 42 erhalten. Dieser Wert ist geringfügig höher als der Fitnesswert des Genoms aus dem 1. Versuch. Die geringfügig höhere Bewertung wird in Abbildung 6.10 sichtbar. Die ermittelte Regelbasis verhält sich ein wenig mobiler und legt weitere Strecken, wenn auch unbefriedigend, in der Nähe der Linie zurück.

Durchschnittlichen Fitness aller Genome pro Generation:

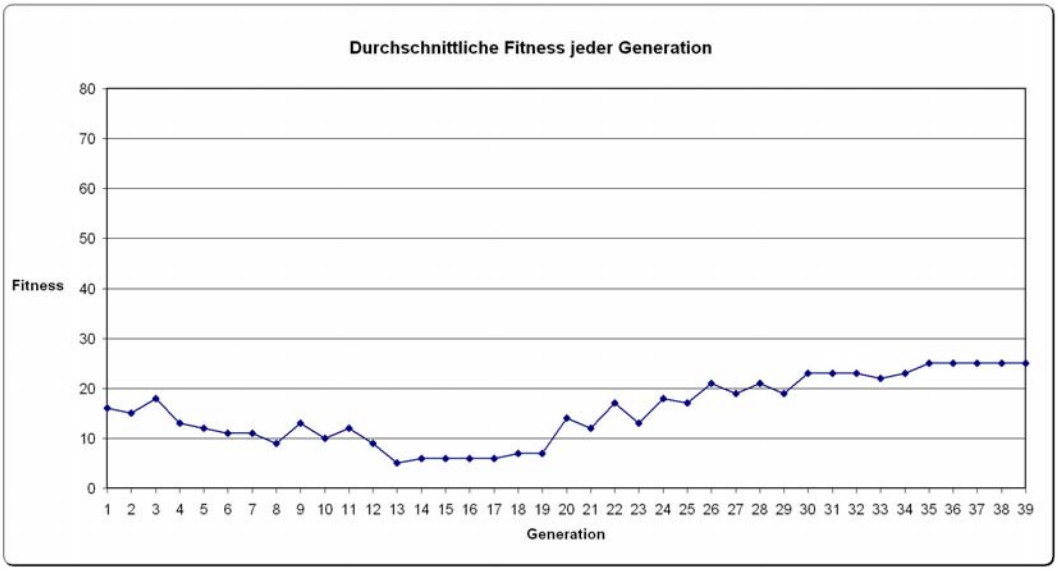


Abbildung 6.10: Versuch 2 - Durchschnittliche Fitness

Die durchschnittliche Fitness der Generationen fällt im Vergleich zum 1. Versuch insgesamt geringer aus. Die durchschnittliche Fitness fällt bis zur 13. Generation ab und steigert sich von dort an langsam wieder. Ab der 20. Generation ist ein deutliches hin und herspringen der Fitnesswerte zu erkennen. Ab der 35. Generation, im Vergleich zum 1. Versuch recht spät, ist keinerlei Varianz in den Werten mehr zu erkennen. Hier ist der genetische Prozess zum Erliegen gekommen.

Wie schon im 1. Versuch, zeigt die Analyse des gefundenen Genoms, dass die Abbildung der Sensorwerte auf die möglichen Aktionen nicht ausreichend gut vorgenommen wurde.

Das gefundene beste Genom:

Gen Nr:	Prämissensymbol:	Aktionssymbol:
0	0	3
1	1	3
2	2	6
3	3	3
4	4	1
5	5	2
6	6	4
7	7	2

Tabelle 6.7: Simulator Versuch 2 - Bestes Genom

Schlussfolgerung:

Wie schon im ersten Versuch, konnte eine Verbesserung der Individuen der Urpopulation nicht beobachtet werden. Aus den gewonnen Daten lässt sich schließen, dass eine hohe Mutationsrate in Verbindung mit einer geringen Kreuzungsrate den genetischen Prozess nicht effektiv genug durch den Suchraum führt. Die hohe Mutationsrate scheint in diesem Fall eher kontraproduktiv zu sein, da sie eine systematische Durchkreuzung des Suchraumes verhindert. Aus diesem Grund soll in dem nächsten Versuch mit einer niedrigeren Mutationsrate und der ursprünglich hohen Kreuzungsrate gearbeitet werden.

6.3.3 Der 3. Versuch

Die Versuchsparameter:

Parameter:	**Wert:**
Fitnessfunktion Streckenmessung	Aus
Populationsgröße	15
Anzahl der Generationen	100
Elitarismusfunktion	Ein
Lebenszyklen pro Genom	20
Kreuzungswahrscheinlichkeit	40%
Mutationswahrscheinlichkeit	5%
Direkte Selektion	55%

Ergebnisse des 3. Versuchs:

Auf den ersten Blick scheint sich das gefundene Bewegungsmuster, in punkto Qualität, nicht vom dem des ersten Versuchs zu unterscheiden. Der Roboter bewegt sich für 20 Zyklen auf zwei Kästchen, innerhalb der virtuellen Welt, hin und her.

Gefundene Regelbasis mit einem Fitnesswert von 75:

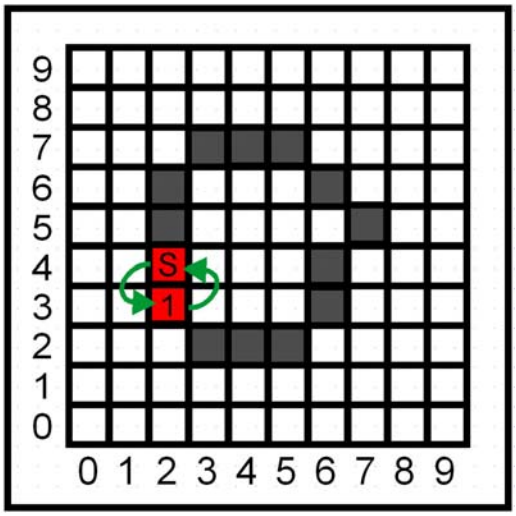


Abbildung 6.11: Versuch 3 - Bewegungsmuster des besten Genoms

Die erzeugte Regelbasis hat einen Fitnesswert von 75 erreicht. Dies ist im Vergleich zu den vorangegangen Versuchen eine Steigerung von 35 Punkten. Im Unterschied zum Bewegungsmuster des ersten Versuchs, bewegt sich der Roboter dieses mal auf der vorgegebenen Linie. Ein Teil des Ziels wurde somit erreicht.

Analyse der Versuchsergebnisse:

Die grafische Analyse der besten Fitnesswerte pro Generation zeigt, im Gegensatz zu den vorangegangen Versuchen, eine deutliche Verbesserung des Genommaterials über die Generationen.

Grafische Analyse der durchschnittlichen Fitnesswerte pro Generation:

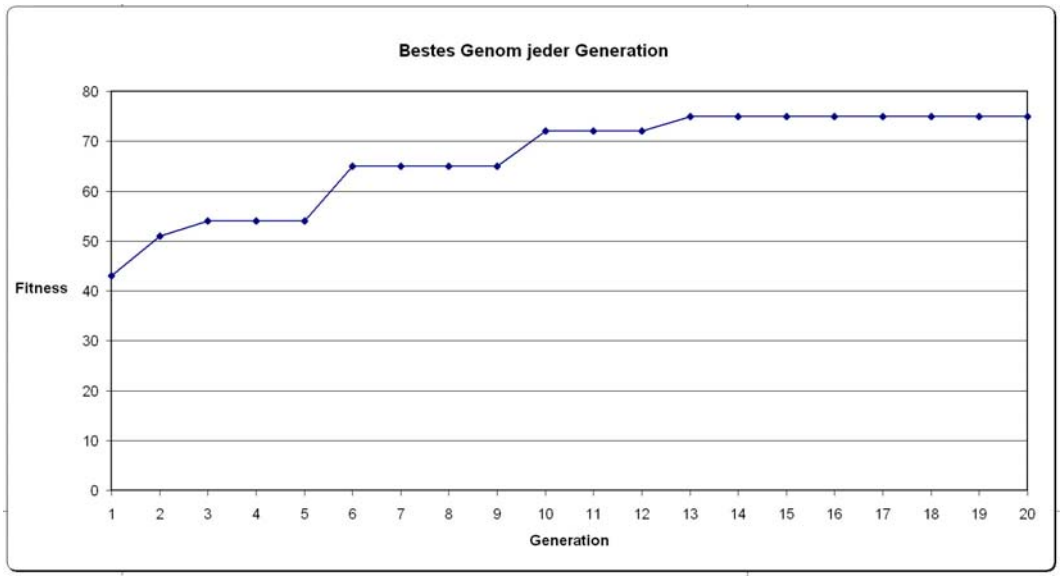


Abbildung 6.12: Versuch 3 - Bestes Genom jeder Generation

Mit einem in der Urpopulation gefundenen besten Fitnesswert von 40 Punkten, steigert sich die Qualität über 13 Generationen bis zu dem besten Genom mit einer Fitness von 75 Punkten.

Der Verlauf der durchschnittlichen Fitness pro Generation zeigt eine deutliche Varianz der Fitnesswerte. Erst ab der 73. Generation ist eine gleichbleibende durchschnittliche Fitness zu erkennen. Diese tritt im Vergleich zu den ersten beiden Versuchen (in der 16. und in der 36. Generation) sehr spät ein.

Durchschnittlichen Fitness aller Genome pro Generation:

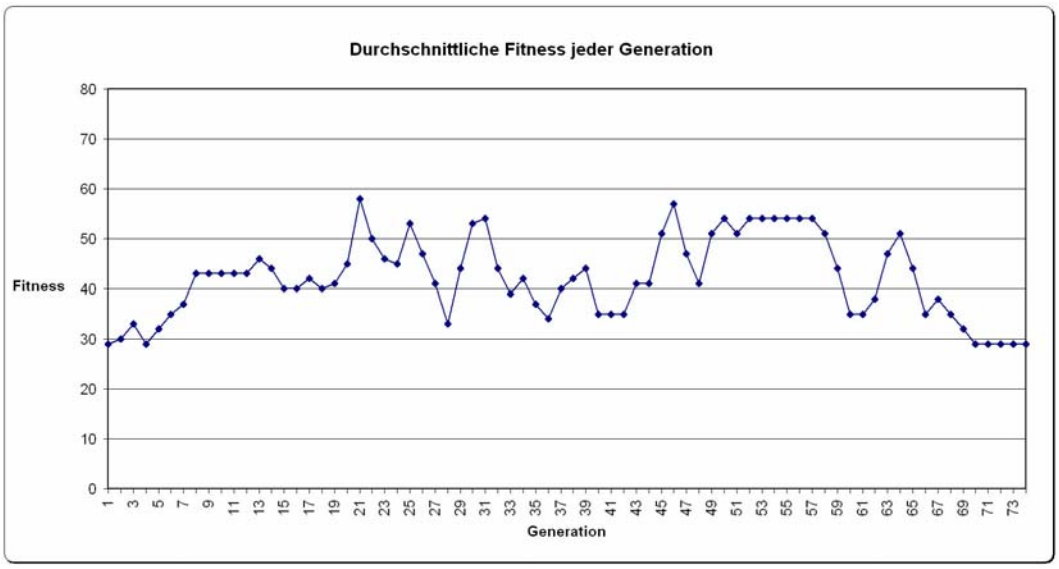


Abbildung 6.13: Versuch 3 - Durchschnittliche Fitness

Das beste gefundene Genom:

Gen Nr:	Prämissensymbol:	Aktionssymbol:
0	0	5
1	1	4
2	2	4
3	3	7
4	4	5
5	5	3
6	6	6
7	7	4

Tabelle 6.8: Simulator Versuch 3 - Bestes Genom

Schlussfolgerung:

Das in der Urpopulation gefundene beste Genom mit einer Fitness von 40 Punkten konnte im Verlauf des evolutionären Prozesses auf 75 Fitnesspunkte verbessert werden. Die graphische Analyse zeigt, dass der genetische Prozess mit den gewählten Einstellungen, eine Verbesserung des genetischen Materials ermöglicht. Daher kann eine Mutationsrate von 5% und eine Kreuzungsrate von 40% als geeignete Einstellung angesehen werden. Das erzeugte Bewegungsmuster hingegen ist, trotz einer Fitnessverbesserung, als unbefriedigend zu bezeichnen. Da trotz einer Verbesserung der Fitness das

Resultat nicht dem geforderten Ziel entspricht, ist die Fitnessfunktion noch einmal auf ihre Tauglichkeit zu überprüfen.

Verbesserung der Fitnessfunktion:

Trotz einer Erhöhung des Fitnesswertes im Vergleich zu den ersten Versuchen wird das Ziel, dass der Roboter der Linie folgt, nicht erreicht. Es stellt sich nun die Frage, warum ist das von der Regelbasis erzeugte Bewegungsmuster, trotz seiner relativ guten Fitness, so unbefriedigend ausgefallen? Zur Analyse muss noch einmal die benutzte Fitnessfunktion und das erzeugte Verhalten betrachtet werden.

Wie aus der für die Fitnessfunktion festgelegten Bewertungstabelle (Tabelle 6.3) in Kapitel 6.2.5 hervorgeht, wird die höchste Punktzahl vergeben, wenn der Roboter sich direkt auf der Linie bewegt. Die Sensoren S1 – S3 weisen dabei das Muster 010 auf. Dieses Muster wird in der momentanen Fitnessfunktion mit der Punktzahl 100 bewertet. Wie sich jetzt zeigt, wurde ein wichtiges Ziel dabei vernachlässigt. Die Fitnessfunktion bewertet ein Hin- und Herfahren auf der Linie genau so gut, wie das Zurücklegen einer Distanz auf der vorgegebenen Linie. Die Fitnessfunktion muss daher um eine Streckenmessung erweitert werden, die das Zurücklegen von größeren Strecken mit in die Bewertung einbezieht und stärker belohnt.

Die Bewertungstabelle wird folgendermaßen angepasst:

S1	S2	S3	Qualitative Bewertung	Quantitative Bewertung	Beschreibung
0	1	0	Sehr gut	150	Roboter befindet sich genau auf der Linie
1	0	0	Gut	50	Roboter befindet sich rechts neben der Linie
0	0	1	Gut	50	Roboter befindet sich links neben der Linie
1	1	1	Ausreichend	20	Roboter steht direkt vor der Linie
0	1	1	Schlecht	5	Roboter steht schräg vor der Linie
1	1	0	Schlecht	5	Roboter steht schräg vor der Linie
1	0	1	Schlecht	5	Roboter steht direkt vor einer Krümmung der Linie
0	0	0	Sehr schlecht	0	Der Roboter hat die Linie aus der Sicht verloren

Tabelle 6.9: Bewertungstabelle - Fitnessfunktion mit Streckenmessung

Die Bewertung für „Roboter befindet sich genau auf der Linie“ wurde noch einmal um 50 Punkte erhöht, um den „Anreiz“ für dieses Verhalten zu erhöhen. Gleichzeit wird zur Realisierung einer Streckenmessung folgende Regelung eingeführt, die ein Zurücklegen größerer Distanzen in einer Richtung zusätzlich belohnt.

Pseudocode Distanzbelohnung:

1. Wenn Distance Counter = 4 setze diesen zurück auf 0
2. Aktuelle Ausrichtung des Roboters mit der letzten Ausrichtung des Roboters vergleichen
3. Ist die Ausrichtung gleich mit der letzten Ausrichtung, addiere zur aktuellen Fitness einen Wert von 100 Punkten und erhöhe Distance Counter um 1.
4. Ist die Ausrichtung ungleich der letzten festgestellten Ausrichtung setzte Distance Counter zurück.

Diese Vorgehensweise stellt eine vereinfachte Distanzmessung dar, da sie lediglich 3 aufeinanderfolgende Fahrten in die selbe Richtung zusätzlich belohnt. Diese Vorgehensweise wurde gewählt, weil eine vollständige Distanzmessung über die Zyklenanzahl programmiertechnisch sehr aufwendig gewesen wäre. Trotzdem wird ein positiver Einfluss auf das gewünschte Ziel, sich in einer Richtung auf der Linie fortzubewegen, erhofft. Der 4. Versuch soll zeigen, ob die Verbesserung ausreichend ist.

Die verbesserte Fitnessfunktion:

$$\textit{Overall Fitness} = \frac{\textit{Summe der Fitnesswerte Bewertungstabelle + Distanzbonus}}{\textit{Anzahl der Zyklen}}$$

6.3.4 Der 4. Versuch

Die in dem 3. Versuch benutzten Parameter werden als geeignet angesehen und weiterverwendet. Um das im 3. Versuch aufgetretene Distanzproblem zu beseitigen, soll die verbesserte Fitnessfunktion zum Einsatz kommen.

Die Versuchsparameter:

Parameter:	**Wert:**
Fitnessfunktion Streckenmessung	Ein
Populationsgröße	15
Anzahl der Generationen	100
Elitarismusfunktion	Ein
Lebenszyklen pro Genom	20
Kreuzungswahrscheinlichkeit	40%
Mutationswahrscheinlichkeit	5%
Direkte Selektion	55%

Ergebnisse des 4. Versuchs:

Die in dem 4. Versuch gefundene Regelbasis steuert den Roboter, wie in der Abbildung zu erkennen, genau auf der vorgegebenen Linie. Die um die Distanzmessung verbesserte Fitnessfunktion führt somit zu dem gewünschten Verhalten. Lediglich eine kleine Abweichung von der Linie in Schritt 8 ist zu erkennen. Dennoch kann der 4. Versuch als erfolgreich bewertet werden.

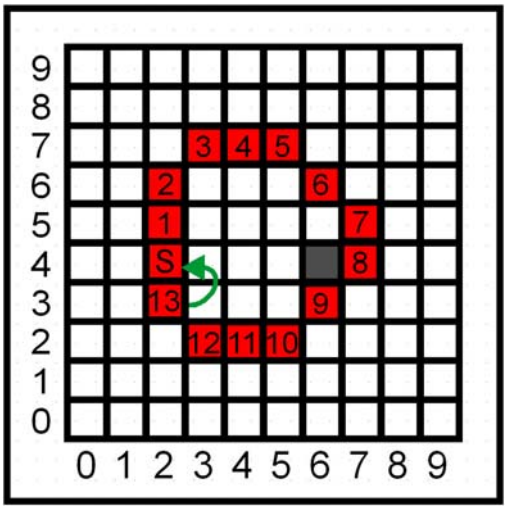


Abbildung 6.14: Versuch 4 - Bewegungsmuster des besten Genoms

Analyse der Versuchsergebnisse:

Die grafische Analyse der besten Fitnesswerte pro Generation zeigt, dass das Genom mit der höchsten Fitness in der 13. Generation generiert wurde. Das

beste gefundene Genom in der Urpopulation hat einen Fitnesswert von 46 Punkten erreicht. Die Fitness des anfänglich besten Genoms konnte so innerhalb von 13 Generationen um 81 Punkte auf 127 Fitnesspunkte verbessert werden. Es ist dabei zu beachten, dass diese Werte nicht direkt mit den Werten der vorangegangenen Versuche zu vergleichen sind, da die verbesserte Fitnessfunktion andere Punkteverteilungen vornimmt. Das bedeutet, dass die gleichen quantitativen Werte nicht zwingend auf eine vergleichbare Qualität der Genome hinweisen.

Grafische Analyse der besten Fitnesswerte pro Generation:

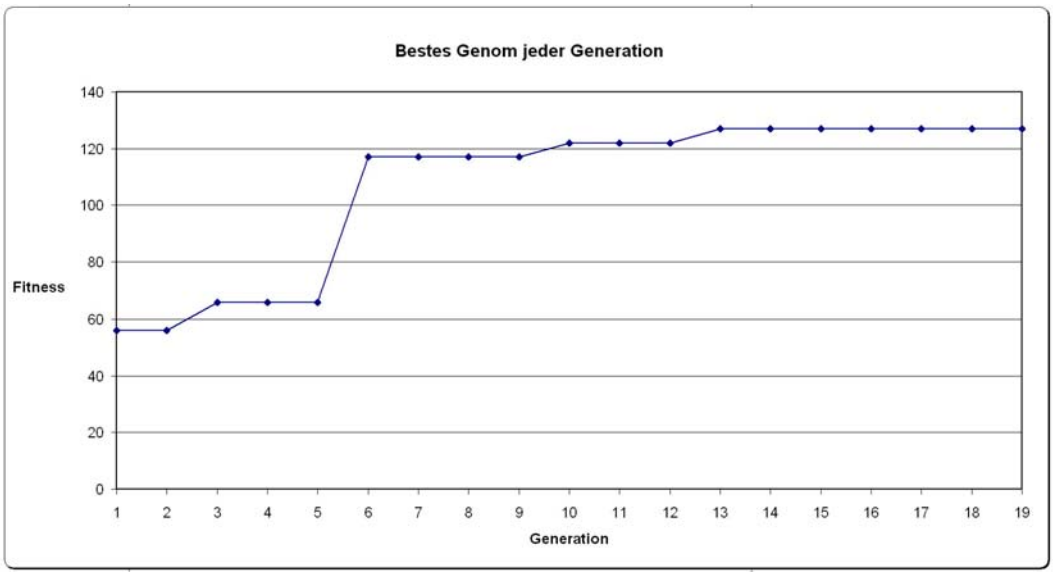


Abbildung 6.15: Versuch 4 - Bestes Genom jeder Generation

Betrachtet man die grafische Analyse der durchschnittlichen Fitness pro Generation, ist innerhalb der ersten 25 Generationen eine sehr starke Varianz der Werte zu erkennen. Insbesondere zwischen der 17. und 19. Generation heben sich die durchschnittlichen Fitnesswerte stark voneinander ab. Ab der 26. Generation bleiben die Werte innerhalb eines engeren Bereichs und unterscheiden sich von Generation zu Generation durchschnittlich um 5 Fitnesspunkte. Bei Erreichen der 50. Generation ist kaum noch eine Varianz der Werte zu erkennen. Der genetische Prozess kommt ab der 74. Generation dann fast vollständig zum Erliegen.

Durchschnittlichen Fitness aller Genome pro Generation:

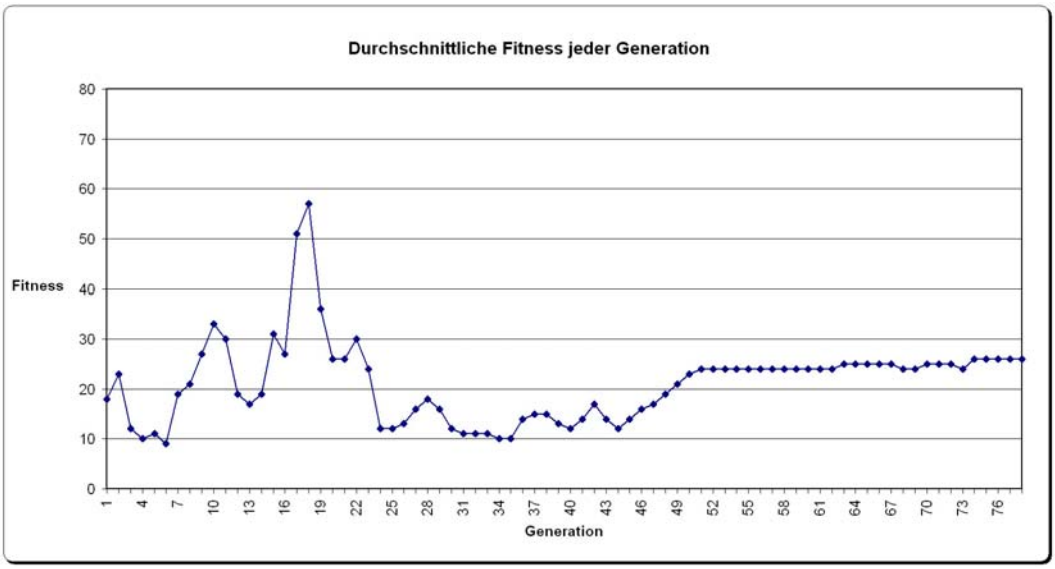


Abbildung 6.16: Versuch 4 - Durchschnittliche Fitness

Das gefundene beste Genom:

Gen Nr:	Prämissensymbol:	Aktionssymbol:
0	0	1
1	1	6
2	2	0
3	3	6
4	4	1
5	5	1
6	6	4
7	7	4

Tabelle 6.10: Simulator Versuch 3 - Bestes Genom

Schlussfolgerung:

Das erzeugte Verhalten des gefundenen Genoms, sowie die grafische Auswertung der Fitnesswerte, deuten darauf hin, dass sowohl die Parametrierung, als auch die verbesserte Fitnessfunktion für die Lösung des gegebenen Problems geeignet sind.
Die eingestellten Parameter können somit in dem Realversuch weiterverwendet werden.

6.3.5 Ergebnisse / Zusammenfassung

Der Verlauf der Versuchsreihe zeigt deutlich, welche Auswirkungen die unterschiedliche Gewichtung der evolutionären Methoden auf den genetischen Prozess hat. Dies wird insbesondere bei der Parametrierung der Mutationswahrscheinlichkeit deutlich. Ein zu groß gewählter Wert lässt den genetischen Prozess nicht greifen und behindert die systematische Durchkreuzung des Suchraumes. Hingegen wirkt sich ein relativ klein gewählter Wert von 5% positiv auf die Suche aus. Das Verrennen in lokalen Maxima kann so effektiv verhindert werden. In Kombination mit einer Kreuzungswahrscheinlichkeit von 40% und einer direkten Emigrationsrate von 55% konnten gute Ergebnisse erzielt werden.

Die Ergebnisse belegen ebenfalls, dass das gewählte Heiratschema, also die Methode zur Selektion der Individuen für die Erzeugung der Nachfolgepopulationen, geeignet ist. Die zum Einsatz kommende Schöpfkellenmethode soll daher für den realen Roboterversuch weiterverwendet werden. Das gewählte Ersetzungsschema (die gesamte Population wird ersetzt), in Verbindung mit der Elitismusfunktion hat sich ebenfalls als brauchbar herausgestellt. Eine im Zusammenhang mit dieser Methode stehende Stagnation der durchschnittlichen Fitnesswerte in späteren Generationen soll an dieser Stelle vernachlässigt werden, da der genetische Prozess zur gewünschten Lösung geführt hat.

Wie sich herausgestellt hat, ist die Fitnessfunktion ein weiterer wichtiger Faktor für die erfolgreiche Durchführung des genetischen Prozesses. Die Abbildung des Zieles in Form von numerischen Werten ist keine triviale Aufgabe. Kleine Änderungen im Bewertungssystem hatten eine große Auswirkung in Bezug auf das gewünschte Ziel zur Folge. Die nicht offensichtliche, aber doch unzureichende Kodierung des Problems, wurde im dritten Versuch deutlich. Trotz eines hohen erreichten Fitnesswertes, war das Ergebnis nicht befriedigend (Vor- und Zurückfahren auf der Linie). Erst die Erweiterung der Fitnessfunktion um eine Distanzmessung führte zum anvisierten Ergebnis. Dies zeigt noch einmal deutlich, dass die Kodierung mit großer Sorgfalt ausgeführt werden muss und ein Vorwissen über das Problem vorhanden sein sollte.

Zusammenfassend kann gesagt werden, dass die mit dem Simulator ausgeführten Versuche ein voller Erfolg waren und wichtige Eckdaten für den Realversuch geliefert haben. Die Logdateien zu den durchgeführten Versuchen, sowie die erstellte Simulationssoftware befinden sich auf der beigefügten CD-Rom.

6.4 Probleme

An dieser Stelle sollen noch einmal die schwerwiegendsten Probleme, die bei der Erstellung des Simulators aufgetreten sind, besprochen werden:

Problem: Verfügbare Bibliotheken

Da für die Zielplattform lediglich ein C-Compiler zur Verfügung stand, konnte leider nicht auf bereits frei verfügbare Bibliotheken für genetische Algorithmen zurückgegriffen werden. Die gefundenen Bibliotheken waren alle in einer objektorientierten Sprache, in den meisten Fällen C++, geschrieben. Daraus resultierte ein anfänglich hoher programmiertechnischer Aufwand. Trotzdem konnten aus der frei verfügbaren C++ Bibliothek GAlib von Matthew Wall [WALL MIT 1996] wichtige Anregungen gewonnen werden.

Problem: Schöpfkellenfunktion

Die anfänglich gewählte Schöpfkellengröße von 7 hat dazu geführt, dass bei sehr kleinen Populationen (15 Generationen) immer die gleichen Genome ausgewählt wurden. Dies führte schnell zur Stagnation des Algorithmus, da die neuen Generationen aus zu wenig unterschiedlichem Genommaterial aufgebaut waren. Das Problem konnte erst nach langwierigen Test- und Vergleichsreihen aufgedeckt werden.

Lösung: Es wurde eine kleinere Schöpfkellengröße von 3 gewählt. So konnte eine bessere Streuung der Genome in den Nachfolgegenerationen erreicht werden. Eine Stagnation trat so nur noch in seltenen Fällen auf.

7 DER ROBOTER

Wie in der Einleitung bereits beschrieben, soll mit Hilfe des vorgestellten und in dem Simulator bereits erprobten genetischen Verfahren eine Choreographie erlernt werden. Das Ziel der Choreographie soll die Positionierung des Roboters auf einer Linie mit dem Infrarotball und dem Zieltor sein. Als ideales Ergebnis wird dabei die Ausrichtung des Roboters mit seiner Stirnseite auf die Ziele gewertet. Die zentralen Fragen hierbei sind, ob dieses komplexe Verhalten in einer angemessenen Zeit erlernt werden kann und ob das Ziel sich ausreichend genau in der Fitnessfunktion kodieren lässt. In der folgenden Abbildung ist das zu erlernende Verhalten noch einmal skizziert. Es ist vorgesehen, dass der Roboter als definierte Startposition zwischen den Sequenzen in einer der oberen Ecken der Spielfläche positioniert wird. Von dort ausgehend, soll er sich dann so positionieren (Beispielhafter Verlauf blaue Linie), dass er sich mit dem IR-Ball und dem grünen Tor auf einer Linie befindet. Dabei sollte sich der Roboter möglichst mit der Front zum Ball ausrichten.

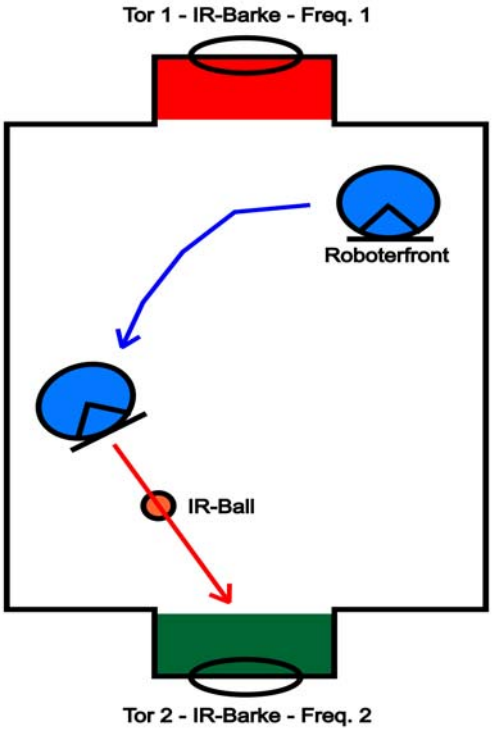


Abbildung 7.1: Mögliche Choreographie

7.1 Design und Implementierung

7.1.1 Die Umwelt des Roboters

Die Umwelt des Roboters besteht aus einer 3 * 2 Meter großen Spielfläche, die von allen Seiten durch 20 cm hohe Holzwangen begrenzt ist. An den Stirnseiten befinden sich rechteckige Ausbuchtungen, die als Tore fungieren. Die Tore sind zur Unterscheidung mit einer blauen und roten Fläche markiert, sowie mit zwei IR-Torbaken ausgestattet. Die Torbarken senden mit zwei unterschiedlichen Frequenzen, wodurch dem Roboter die Erkennung und Unterscheidung der Tore möglich ist. Der Boden der Spielfläche ist mit einem grauen Flies ausgelegt, der dem Roboterantrieb ausreichend Bodenhaftung bieten soll. In der folgenden Abbildung ist der fertige Versuchsaufbau zu sehen.

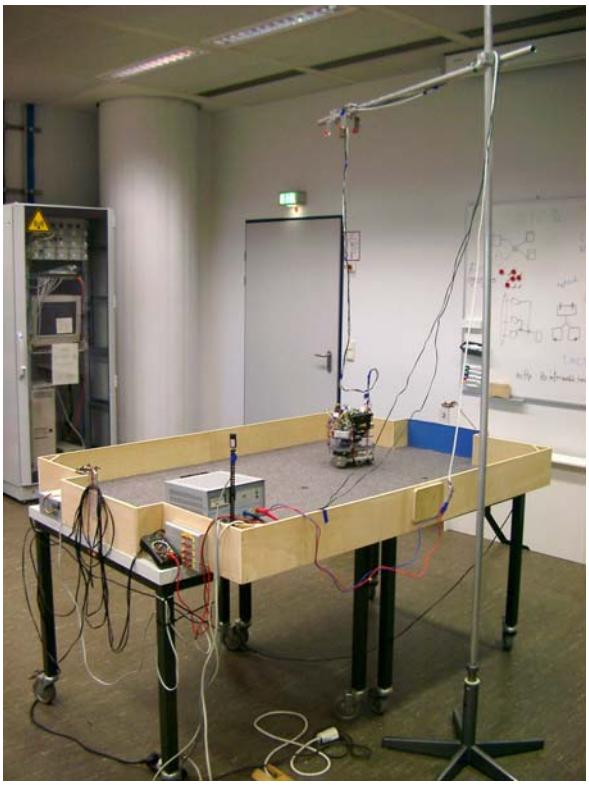


Abbildung 7.2: Versuchsaufbau

7.1.2 Die Aktoren

Als Antriebssystem steht dem Roboter ein Dreiachsantrieb zur Verfügung. Die Achsen sind dabei symmetrisch im Winkel von 120 Grad zueinander angebracht. Verbunden mit drei Motoren zum Antrieb der Achsen und den daran montierten omnidirektionalen Rädern, entsteht ein äußerst wendiges System.

Abbildung 7.3: Omnidirektionaler Antrieb
(Quelle M. Manger)

Jede Achse kann aktiv durch einen Motor angetrieben werden und gleichzeitig passiv seitlich rollen. Mit Hilfe unterschiedlicher Drehgeschwindigkeiten der Achsen ist es möglich, jede beliebige Position anzufahren, ohne vorher den Roboter explizit drehen zu müssen. Von Michael Manger [MANGER 2004], dem Entwickler der Roboterplattform, wurden insgesamt 27 praktikable Bewegungsabläufe[7] deklariert, die für den Versuch weiterverwendet werden sollen. In der folgenden Abbildung sind zur Verdeutlichung noch einmal die direkt anfahrbaren Himmelsrichtungen und die dafür nötigen Motoreinstellungen aufgeführt.

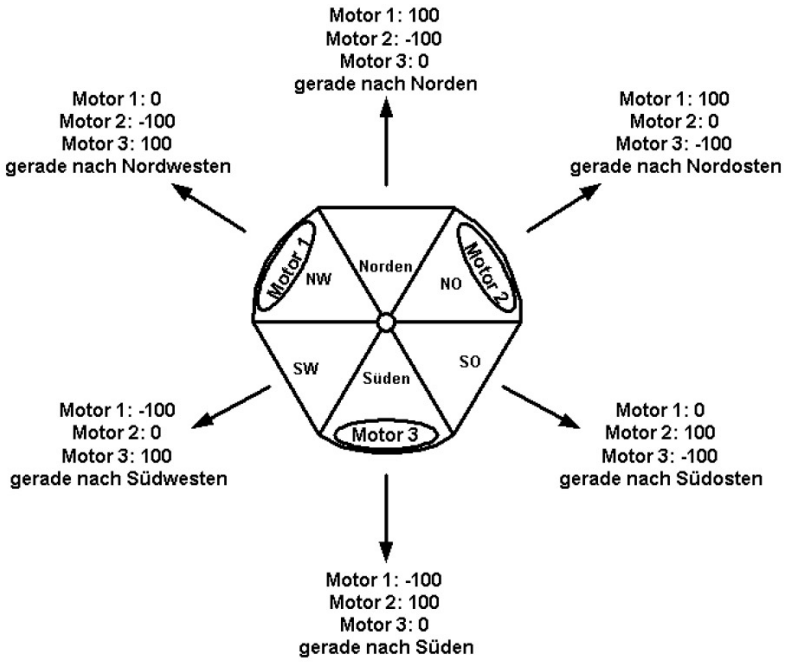


Abbildung 7.4: Hauptfahrtrichtungen
(Quelle M. Manger)

[7] Rein theoretisch wären mit unterschiedlichen Motoreinstellungen (10 Geschwindigkeitsstufen werden unterstützt) einige Tausend Fahrkombinationen möglich. In der Praxis hat sich aber gezeigt, dass der Antrieb lediglich in den Einstellungen 0% oder 100% Leistung reproduzierbare Fahrmanöver absolviert.

Wie im Kapitel 5.2.1 bereits beschrieben, müssen die möglichen Aktionen, die der Antrieb ausführen kann in Aktionssymbole kodiert werden. Die Kodierung der Symbole soll auch hier in Form von reellen Zahlen erfolgen, um den Kodierungsaufwand zu minimieren. Für den Regelinterpreter werden die möglichen Aktionen, wie in der folgenden Liste beschrieben, als numerische Symbole abgebildet.

Aktions-symbol	Bewegungsrichtung	Motor 1	Motor 2	Motor 3
1	Im Stand linksrum drehen	-100%	-100%	-100%
2	Radius um Rad 3 / linksrum	-100%	-100%	0 %
3	Radius ca. 48 cm um Rad 3 / linksrum	-100%	-100%	100 %
4	Radius um Rad 2 / linksrum	-100%	0 %	-100 %
5	Radius ca. 7,5 cm über Rad 2 und 3 / linksrum	-100%	0 %	0 %
6	Gerade aus Richtung Südwest	-100%	0 %	100 %
7	Radius ca. 48 cm um Rad 2 / linksrum	-100%	100 %	-100 %
8	Gerade aus Richtung Süden	-100%	100 %	0 %
9	Radius ca. 48 cm um Rad 1 / rechtsrum	-100%	100 %	100 %
10	Radius um Rad 1 / linksrum	0 %	-100 %	-100 %
11	Radius ca. 7,5 cm über Rad 1 und 3 / linksrum	0 %	-100%	0 %
12	Gerade aus Richtung Nordwest	0 %	-100%	100 %
13	Radius ca. 7,5 cm über Rad 1 und 2 / linksrum	0 %	0 %	-100 %
14	STOP	0 %	0 %	0 %
15	Radius ca. 7,5 cm über Rad 1 und 2 / rechtsrum	0 %	0 %	100 %
16	Gerade aus Richtung Südost	0 %	100 %	-100 %
17	Radius ca. 7,5 cm über Rad 1 und 3 / rechtsrum	0 %	100 %	0 %
18	Radius um Rad 1 / rechtsrum	0 %	100 %	100 %
19	Radius ca. 48 cm um Rad 1 / linksrum	100 %	-100 %	-100 %
20	Gerade aus Richtung Norden	100 %	-100 %	0 %
21	Radius ca. 48 cm um Rad 2 / rechtsrum	100 %	-100 %	100 %
22	Gerade aus Richtung Nordost	100 %	0 %	-100 %
23	Radius ca. 7,5 cm über Rad 2 und 3 / rechtsrum	100 %	0 %	0 %
24	Radius um Rad 2 / rechtsrum	100 %	0 %	100 %
25	Radius ca. 48 cm um Rad 3 / rechtsrum	100 %	100 %	-100 %
26	Radius um Rad 3 / rechtsrum	100 %	100 %	0 %
27	Im Stand rechtsrum drehen	100 %	100 %	100 %

Abbildung 7.5: Bewegungsmöglichkeiten des Antriebssystems
(Quelle: M. Manger)

7.1.3 Die Sensoren

Der Roboter verfügt über verschiedene Sensoren, um sich in seiner Umwelt orientieren zu können. Dies sind zum einen die 8 Bumpersensoren, die den Roboter bei Versagen der aktuell zu bewertenden Regelbasis vor größeren Schäden bewahren soll und zum anderen die vier Sharp Abstandssensoren, die zur Orientierung innerhalb der Spielfläche eingesetzt werden. Für die Steuerung über die Regelbasen sind die zwei Infrarot Ballsensoren, die auf einem beweglichen Arm installiert sind und der Torerkennungssensor, der sich auf der definierten Vorderseite des Roboters befindet, ausschlaggebend. Über diese Sensoren kann die Position des Roboters in Relation zu den Zielen erfasst werden.

Die Ballsensoren:

Der zu erkennende IR-Ball (Fa. Wiltronics), der auf der Spielfläche positioniert wird, verfügt über einfache IR-Sendedioden. Die Dioden sind in dem Ball radial angeordnet und strahlen ein unmoduliertes Infrarotlicht aus. Durch die besondere Anordnung der Sendedioden, gibt der Ball eine in alle Richtungen gleichmäßige Strahlung ab. Die von dem Ball ausgesendete Infrarotstrahlung kann von zwei Ballsensoren, die auf dem Roboter installiert sind, erkannt werden. Die Ballsensoren sind auf der untersten Ebene des Roboters auf einem beweglichen Arm angebracht. Über einen Servomotor[8] kann jeder Sensor in einem Bereich von 180 Grad bewegt werden. Die beiden Ballsensoren sind über zwei Analogports mit dem Aksen-Board verbunden. Die Ballsensoren liefern eine Spannung, die sich analog zur Ausrichtung des Infrarotballes verhält. Auf experimentellem Wege wurde ein Messwert herausgefunden, bei dem davon ausgegangen werden kann, dass sich der Ball in Blickrichtung der Sensoren befindet. Um einen möglichst genauen Winkel zu erhalten, wurden die Sensoröffnungen durch zwei kleine Röhrchen begrenzt. So konnte der Erfassungsbereich der Sensoren eingeengt werden, um ausreichend genaue Werte zu erhalten und so eine Aussage über den Winkel treffen zu können.
In der folgenden Abbildung ist die Roboterplattform mit den zwei Ballsensoren und deren Scanbereich zu erkennen.

[8] Der Sensorarm wurde ursprünglich über einen Getriebemotor angetrieben, der keine zuverlässigen Rückschlüsse auf die Winkelstellung des Armes zuließ. Daher wurde der ursprüngliche Antrieb von mir durch einen Servomotor ersetzt.

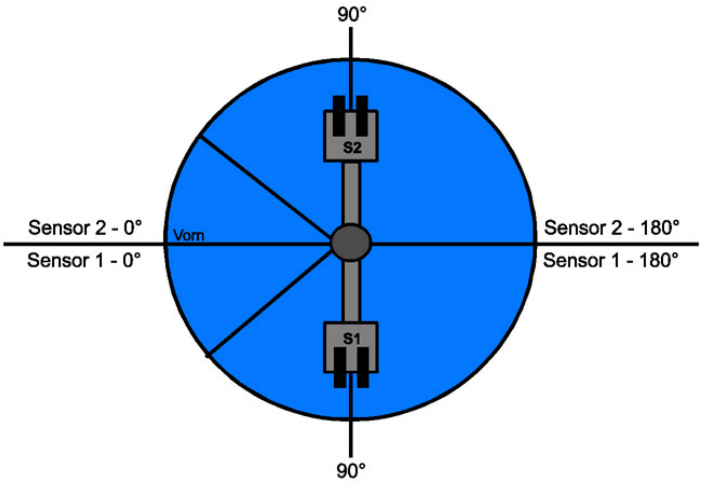


Abbildung 7.6: Roboter - Scanbereich der Ballsensoren

Per Definition wird die Nullstellung für beide Sensoren auf der Vorderseite des Roboters festgelegt. Befindet sich also einer der Sensoren auf der Vorderseite des Roboters, wird diese Stellung per Definition mit 0 Grad interpretiert. Erkennt einer der Sensoren bei 0 Grad den Ball, befindet sich dieser also direkt vor der Front des Roboters. Erkennt einer der Sensoren den Ball hingegen bei 180 Grad, befindet sich der Ball genau hinter dem Roboter. Über die Messwerte Servostellung und „Ball wurde erkannt“, kann somit eine Aussage darüber getroffen werden, in welchem Winkel sich der Ball relativ zur Front des Roboters befindet und ob er sich rechts oder links neben der Front befindet.

Der Torsensor:

Die Tore der Spielfläche sind mit zwei Infrarot Torbarken ausgestattet. Zur Unterscheidung der Tore senden die Barken standardmäßig mit einer 40 KHz Trägerfrequenz, auf die ein 100 Hz oder 125 Hz Signal aufmoduliert wird. Die Trägerfrequenz hilft dabei, dass Signal unempfindlich gegenüber Umgebungslicht zu machen.

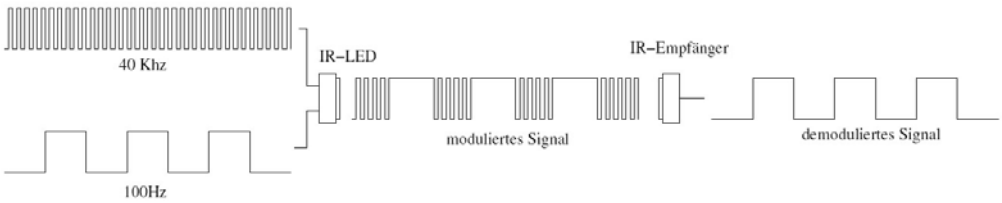


Abbildung 7.7: Trägerfrequenz mit 100Hz moduliert
(Quelle: Nutzerhandbuch Aksenboard)

Zur Erkennung der Torbarken befindet sich ein Torerkennungssensor (Fa. Wiltronics), mit drei Empfangsdioden auf dem Roboter. Die eingesetzten Empfangsdioden sind bereits mit einer kleinen Elektronik ausgestattet, die einen Ausgang auf Low schaltet, sobald ein 40KHz Signal detektiert wird. Das auf die Trägerfrequenz aufmodulierte Signal von 100Hz oder 125Hz kann so durch messen des Taktverhältnisses (Flankenmessung) unterschieden werden. Die Bibliothek des Aksen-Boards enthält eine Routine, über die eine vorgegebene Frequenz, die von den Sensoren geliefert wird, erkannt werden kann. Da bei jeder Übertragung durch Störquellen Fehler auftreten können, kann ein Wert für die maximale Fehlerzahl pro Periode angegeben werden. Über eine Methode „mod_irX_status()“ kann die Empfangsqualität abgerufen werden. Sie übergibt die Anzahl der letzten zusammenhängenden Perioden. Desto größer dieser Wert ist, desto direkter fällt das Signal der Torbarken in die Empfangsdiode.
Der Torsensor verfügt über drei Empfangsdioden, deren Sichtbereich, wie auch schon bei den Ballsensoren, durch Röhrchen eingeengt wurde. Durch die Einengung des Sichtbereiches, kann auch hier eine genauere Aussage über den Winkel relativ zum Tor getroffen werden. Über die von den Sensoren gemessenen Flanken und die daraus resultierende Empfangsqualität lässt sich somit bestimmen, ob der Roboter direkt vor einem Tor steht (alle drei Sensoren sprechen stark an), oder ob er sich eher rechts oder links vom Tor befindet (ein oder zwei Sensoren auf der entsprechenden Seite sprechen stark an).
Die Messwerte zur Bestimmung der Lage des Roboters relativ zum Zieltor wurde wie auch schon bei den Ballsensoren auf experimentellem Wege festgestellt.

Bumpersensoren:

Der Roboter verfügt über acht Bumpersensoren, die rund um den Roboter angebracht sind. Die installierte Bumperleiste ist mit acht Microschaltern verbunden, die eine Kollisionserkennung ermöglichen. Die Bumpersensoren stellen eine Art Fallbacksicherung dar. Sollte die Qualität einer zu testenden Regelbasis sehr schlecht ausfallen und der Roboter gegen eine Begrenzung fahren, so kann dies über die installierten Bumper erkannt werden. Für diesen Fall wird die aktuelle Regelbasis mit einem Fitnesswert von 0 abgestraft.

Sharpsensoren:

Auf der Ebene 2 sind vier Abstandssensoren installiert. Dabei sind drei Sensoren nach vorne und ein Sensor nach hinten gerichtet. Die eingesetzten Sensoren der Firma Sharp arbeiten auf Infrarot Basis und können in einem Bereich von 8 – 60 cm feste Objekte erkennen. Die Sharpsensoren werden benötigt, um den Roboter nach Testen einer Regelbasis in einen definierten Ausgangszustand zu bringen. Geplant ist dabei, unter zu Hilfenahme der Torsensoren, eine Ecke der Spielfläche anzufahren und den Roboter in Richtung Spielmitte auszurichten. Dieser Vorgang wird über eine Methode „GoHome()“ realisiert, die unabhängig von der Regelinterpretation abläuft.

7.1.4 Der Prämissenencoder

Der Prämissenencoder sorgt, wie schon in der Simulationssoftware, für die Kodierung der Sensorwerte in die von dem Regelinterpreter benötigten Prämissensymbole. Die Kodierung des bei dem simulierten Roboter eingesetzten Sensors in die entsprechenden Prämissensymbole, ist signalbedingt kein Problem gewesen. Da der Sensor lediglich aus drei Helligkeitssensoren besteht, welche 1 oder 0 für Linie erkannt oder nicht erkannt zurück gegeben haben. Daraus hat sich eine überschaubar Anzahl von Prämissen ergeben. (2^3 = 8) In der realen Welt liefern Sensoren leider nicht so klar abgrenzbare und zuverlässige Werte. Da die Sensoren des Roboters analoge Werte zurückliefern und sich daraus in Kombination eine sehr große Anzahl von Prämissen ergeben würde, habe ich mich entschlossen die Sensorwerte in Bereiche einzuteilen.

Durch die Klassifizierung der Sensorwerte in einzelne Bereiche kann die Anzahl der Prämissen erheblich verringert werden, ohne jedoch befürchten zu müssen, dass die Werte für die Steuerung des Roboters nicht mehr ausreichend genau sind. Der zu durchquerende Suchraum verkleinert sich durch die Vorgehensweise außerdem erheblich, was wiederum die Chancen in einer begrenzten Zeit eine Lösung zu finden erheblich steigert.

Klassifizierung der Ballsensoren:

In der folgenden Abbildung ist die Einteilung der einzelnen Scanbreiche eingezeichnet. Die Front des Roboters ist auf der linken Seite zu sehen. Die Ballsensoren S1 und S2 decken jeweils einen Bereich von 180° auf der linken und rechten Hälfte des Roboters ab. Die Sensoren werden dabei über einen Servomotor bewegt. Die Aufteilung soll in insgesamt sieben Bereiche erfolgen.

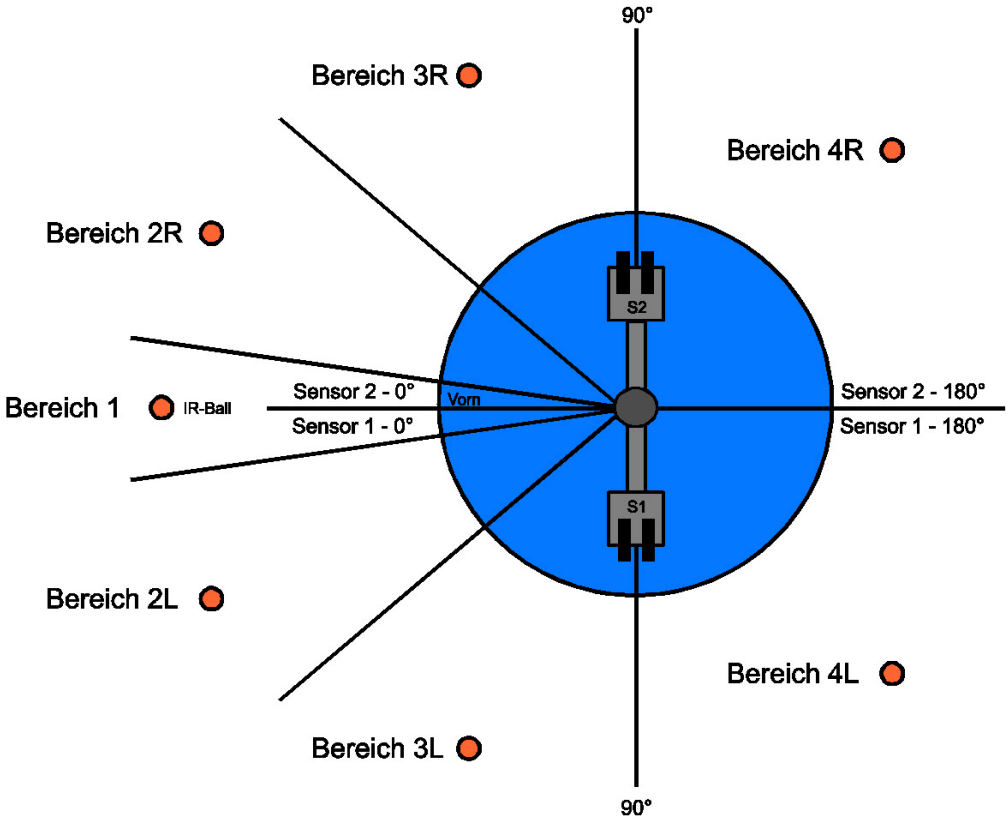


Abbildung 7.8: Ballsensor - Klassifizierung der Scanbereiche

Die sieben Bereiche des Ballsensors:

- Bereich 1: Der Ball befindet sich direkt vor dem Roboter. Dieser Bereich ist bewusst sehr schmal ausgewählt, um exakt feststellen zu können, ob der Ball sich in der Zielposition befindet.

- Bereich 2L: Der Ball befindet sich sehr nahe im linken Frontbereich. Dieser Bereiche erstreckt sich ca. über 40° auf der linken Seite.

- Bereich 2R: Der Ball befindet sich sehr nahe im rechten Frontbereich. Dieser Bereich erstreckt sich ca. über 40° auf der rechten Seite.

- Bereich 3L / 3R: Der Ball befindet sich nahe dem linken oder rechten Frontbereich. Diese Bereiche erstrecken sich zu beiden Seiten über ca. 45°.

- Bereich 4L / 4R: Der Ball befindet sich auf der linken oder rechten Seite hinter dem Roboter. Dies ist der größte Bereich der Aufteilung und erstreckt sich auf die beiden hinteren Hälften jeweils über 90°.

Eine Sensorfunktion „FindeIRBall()“ welche die Ballsensoren bedient, übernimmt die Auswertung der erfassten Daten und prüft auf die oben angegebenen

Bereiche. Die Sensorfunktion gibt dann die entsprechende Information, in welchem Bereich sich der Ball befindet, in Form einer Konstanten zurück. Für den Fall, dass der Ball nicht gefunden werden kann, wird eine zusätzliche Konstante eingeführt.

Folgende mögliche Konstanten liefert die Sensorfunktion zurück:

DEFINE	KONSTANTE	BESCHREIBUNG
SENS_BALL_IN_FRONT	31	// Ball genau vor der Roboterstirnseite
SENS_BALL_VERY_CLOSE_R	32	// Ball sehr nah rechts vom Roboter
SENS_BALL_VERY_CLOSE_L	33	// Ball sehr nah links vom Roboter
SENS_BALL_CLOSE_R	34	// Ball nah rechts vom Roboter
SENS_BALL_CLOSE_L	35	// Ball nah links vom Roboter
SENS_BALL_FAR_OFF_R	36	// Ball entfernt rechts vom Roboter
SENS_BALL_FAR_OFF_L	37	// Ball entfernt rechts vom Roboter
SENS_BALL_NOT	38	// Ball ist nicht zu sehen

Klassifizierung des Torsensors:

Wie schon beim Ballsensor, soll zur Reduzierung der möglichen Prämissen eine Einteilung in verschiedene Bereiche vorgenommen werden. Der Torsensor hat, wie auf der Abbildung zu erkennen, eine keilförmige Form mit drei Öffnungen für die Empfangsdioden. Der Empfangsbereich der mittleren Diode D2 ist am stärksten eingegrenzt. Dies ist nötig, um zuverlässig registrieren zu können, dass sich ein Tor in direkter Linie zur Roboterfront befindet. Die beiden Empfangsdioden D1 und D3 sind mit einer relativ großen Öffnung ausgestattet, um auch noch IR-Strahlung aus den Randbereichen einfangen zu können. Insgesamt ist der Empfangsbereich des Torsensors in fünf Bereiche aufgeteilt.

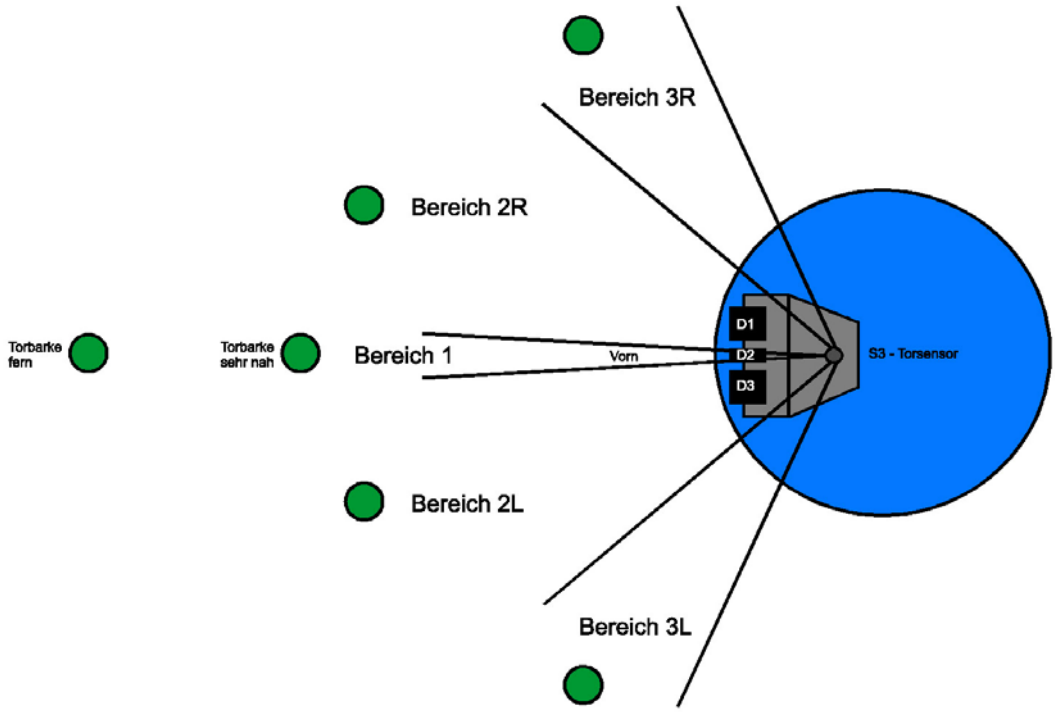


Abbildung 7.9: Torsensor - Klassifizierung der Scanbereiche

Für jeden der definierten Bereiche wurde auf experimentellem Wege ein entsprechender Messwert für die Empfangsdioden D1-D3 ermittelt. Dabei hat sich eine Eigenart der Sensoren herausgestellt: Während bei der Messung direkt gegenüber einer Torbarke (Torbarke sehr nah) Sensor D1 - D3 ansprechen, wird auf größerer Distanz (Torbarke fern) lediglich der mittlere Sensor D2 angesprochen. Obwohl in beiden Fällen die Torbarke direkt in Front des Roboters ist, werden sehr unterschiedliche Messwerte ermittelt. Auch eine Modifikation der Einlassöffnungen des Torsensors hat dieses Phänomen nicht beseitigen können. Da zwei unterschiedliche Messwerte den Bereich 1 repräsentieren und dabei etwas über die Entfernung zum Ziel ausgesagt wird, habe ich mich entschlossen für den Bereich 1 zwei Prämissen einzuführen.

Die fünf Bereiche des Torsensors:

- Bereich 1: Das Tor befindet sich direkt gegenüber dem Roboter. Dieser Bereich ist bewusst sehr schmal ausgewählt, um exakt feststellen zu können, ob das Tor sich auf einer Linie mit der Front befindet. Dieser Bereich wird, wie oben beschrieben, durch zwei verschiedene Sensorkombinationen abgedeckt, die zusätzlich eine Aussage über die Entfernung des Tores zulässt.

- Bereich 2L / 2R: Das Tor befindet sich sehr nahe im linken oder rechten Frontbereich des Roboters. Diese Bereiche erstrecken sich ca. über 40° auf der linken und rechten Seite.

- Bereich 3L / 3R: Das Tor befindet sich nahe dem linken oder rechten Frontbereich. Diese Bereiche erstrecken sich zu beiden Seiten über ca. 30°.

Der Sichtbereich des Torsensors deckt so ca. 160° ab. Die Einordnung und Verarbeitung der Sensorwerte übernimmt eine Funktion „Gate(GateNr)“. Das zu erkennende Tor wird per Parameter an die Funktion übergeben.

```
Gate ( MyGate )     // MyGate     = 100Hz
Gate ( NotMyGate )  // NotMyGate  = 125Hz
```

Zurückgeliefert wird der klassifizierte Sensorbereich, in dem sich das Tor momentan in Relation zur Roboterfront befindet. Für den Fall, dass das Zieltor sich nicht im Sichtbereich des Sensors befindet, wird eine zusätzliche Konstante eingeführt. Daraus ergeben sich 7 Einteilungen. In der folgenden Tabelle sind die möglichen Rückgabewerte der Funktion aufgeführt.

Folgende mögliche Konstanten liefert die Funktion „Gate“ zurück:

DEFINE	KONSTANTE	BESCHREIBUNG
SENS_GATE_R	21	// Tor ist rechts nah
SENS_GATE_L	22	// Tor ist links nah
SENS_GATE_M	23	// Tor ist mittig fern(D2)
SENS_GATE_LM	24	// Tor ist links sehr nah
SENS_GATE_RM	25	// Tor ist rechts sehr nah
SENS_GATE_FULL	26	// Tor ist mittig nah (D1-D3)
SENS_GATE_NOT	27	// Tor ist nicht zu sehen

Umsetzung der Sensorkonstanten in Prämissen

Aus der eben beschriebenen Klassifizierung der Sensorwerte in die einzelnen Bereiche ergibt sich nun eine überschaubare Anzahl von Konstanten, die von den Sensoren zurückgeliefert werden. Von dem Ballsensor werden acht und von dem Torsensor sieben Konstanten geliefert. Diese können nun in entsprechende Prämissen umgewandelt werden. Da die Sensorwerte stets in Kombination auftreten, ergibt sich aus den 8 Ballwerten und den 7 Torwerten eine Prämissenmenge von 8 * 7 = 56.

Da die Prämissenanzahl sehr hoch ist, werden diese automatisch innerhalb des Decoders aus den möglichen Sensorkonstantenkombinationen zusammengestellt. Die Prämissen werden, wie auch schon im Simulator, auf eine Folge von reellen Zahlen abgebildet. Für alle Sensorkombinationen wird somit Prämisse 1-56 erstellt und im Genom kodiert.

7.1.5 Das Genom (Regelbasis)

Durch die Menge der erzeugten Prämissen ergibt sich eine definierte Länge der einzelnen Regelbasen. In jedem Genom muss die Prämisse 1-56 mit einer Zufallsaktion belegt werden. Dabei wird die Zufallsaktion aus der Menge der 27 möglichen Aktionssymbole ausgewählt. Die Datenstruktur der Simulations-software, die aus einem mehrdimensionalen Array besteht, wurde auf die entsprechende Größe angepasst.

7.1.6 Die Fitnessfunktion

Das Ziel des Roboters soll es sein, sich mit seiner Frontseite auf eine Linie mit dem Ball und dem Tor zu bewegen. Auf Basis der von Torsensor und Ballsensor gelieferten Daten, muss dieses Ziel nun in Form einer Fitnessfunktion abgebildet werden. Die Sensordaten werden, wie auch schon im Simulator, zur Bewertung der ausgeführten Aktionen herangezogen. Dabei ist es wichtig, zwischen gewünschten und unerwünschten Sensordaten zu unterscheiden, um dies in die Fitnessfunktion integrieren zu können. Die Startposition des Roboters soll sich stets in einer der oberen Ecken befinden. Der Infrarotball soll während des Versuchs statisch auf einem Punkt der Spielfläche liegen.

In der folgenden Abbildung ist das Ziel noch einmal verdeutlicht:

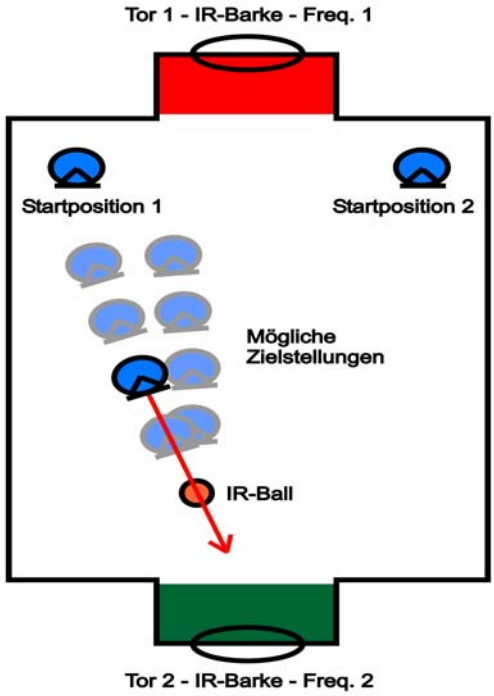


Abbildung 7.10: Mögliche Zielstellungen

Da sich je nach Entfernung zum Zieltor der Empfangskegel des mittleren Torsensors verkleinert oder vergrößert, sind die oben angegeben Zielstellungen denkbar.

Die „Idealen“ Sensordaten:

Die Fitnessfunktion setzt sich aus zwei Werten zusammen, anhand derer die Qualität der aktuellen Regelbasis bewertet werden kann. Dies sind zum einen die Daten der Ballsensoren und zum anderen die gemessenen Werte des Torsensors. Der Ball befindet sich in einer idealen Stellung, wenn er direkt gegenüber der Front des Roboters liegt. Das bedeutet, wenn sich einer der Ballsensoren im Bereich 1 (bei 0°) befindet, so ist ein Teil des Ziels erreicht. Registriert der Torsensor gleichzeitig die Zieltorbarke direkt gegenüber dem Roboter, ist das Gesamtziel erreicht. Es kann somit festgestellt werden, das desto kleiner der Winkel zwischen Ball und Front des Roboters, sowie zwischen Tor und Front des Roboter ist, desto besser wurde das anvisierte Ziel erreicht. Umgekehrt, je größer die Winkel sind, desto weiter ist der Roboter von seiner Zielposition entfernt.

Daraus ergibt sich folgende vereinfachte Fitnessfunktion:

Fitness(x) = (max. Winkel Ball - Winkel Ballsensor) + (max. Winkel Tor - Winkel Torsensor)

Max. Winkel Ball: Dies ist der maximal mögliche Winkel, den der Ball von der Roboterfront entfernt liegen kann.

Max. Winkel Tor: Dies ist der maximal mögliche Winkel, die der Torsensor als Abweichung von der Roboterfront erfassen kann.

Da die Ballsensorfunktion bei Erkennen des Balles den aktuellen Winkel des Sensors zurückgibt (Winkel des Servomotors), kann dieser Wert direkt von dem maximal möglichen Winkel (180°) abgezogen und benutzt werden. Es soll gelten, je kleiner der Winkel, desto höher fällt die Fitness aus. Der Torsensor liefert keinen direkten Winkel zurück, da aber ein direkter Zusammenhang zwischen der Höhe der gemessenen Sensorwerte und dem Winkel zum Tor liegt, können die Werte mit einer einfache Modifikation ebenfalls direkt benutzt werden.

Es gilt, je direkter einer der drei Infrarot Sensoren innerhalb des Torsensors das Tor sieht, desto höher fällt deren Messwert aus. Daraus folgt, dass die Sensorwerte des Torsensors direkt in die Fitnessfunktion einfließen können.

Fitness(x) = (max. Winkel Ball – Winkel des Ballsensor) + TorIRD1 + TorIRD2 + TorIRD3

TorIRD1 – TorIRd3: Analoger Messwert der in dem Torsensor verbauten IR-Empfangsdioden.

In der folgenden Abbildung sind die typischen Messwerte für die möglichen Stellungen des Tores noch einmal grob aufgeführt. Die Prozentwerte beziehen sich auf den maximal zu erzeugenden Messwert. Die Messwerte der IR-Dioden zwischen den einzelnen Torwinkeln verlaufen dabei fließend.

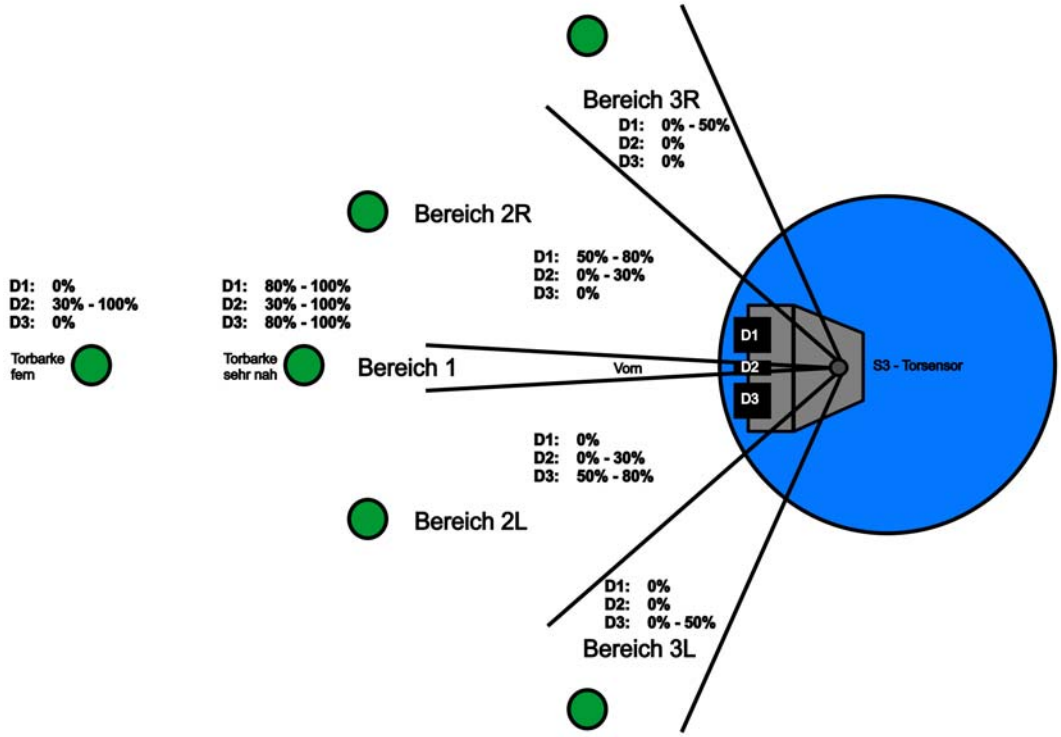


Abbildung 7.11: Torsensor - Messwerte

Die Eigenheiten des Torsensors in Bezug auf die Situationen „Torbarke fern“ und Torbake sehr nah“ vor dem Roboter, fließt bei näherer Betrachtung ausreichend in die Fitnessfunktion ein. Für den Fall, dass sich das Tor weit entfernt von der Roboterfront befindet, fällt die Wertung in der Fitnessfunktion geringer aus. Lediglich die Sensor Diode 2 liefert einen Messwert zwischen 30% und 100%. Sollte sich das Tor sehr nahe direkt vor dem Roboter befinden, wird der maximale Messwerte erreicht. Alle drei Sensordioden sprechen dann an. Daraus folgt, das bei direkter Übernahme der Sensorwerte die Situation „Torbarke sehr nah“ besser bewertet wird, als „Torbarke fern“. Da dies ein gewünschtes Verhalten ist, muss die Fitnessfunktion an dieser Stelle nicht weiter verfeinert werden.

Es stellt sich nun die Frage, warum nicht, wie auch schon bei der Simulation, einfach die Prämissen zur Fitnessbemessung herangezogen wurden? Die Sensorwerte in der physikalischen Welt sind weitaus differenzierter, als jene in der Simulation. Da die Prämissen eine Klassifizierung der Bereiche darstellen,

sind die Werte in Bezug auf das Ziel daraus folgend relativ ungenau. Da es bei der Fitnessmessung aber auf möglichst differenzierte Werte ankommt, habe ich mich entschlossen die analogen Sensorwerte und die genaue Winkelangabe des Ballsensorservos zur Bewertung heranzuziehen.

7.1.7 Die Benutzerschnittstelle

Die Implementierung der Benutzerschnittstelle findet auf dem Lart-Board statt, da dieses eine einfache Anbindung über die integrierte RS232 Schnittstelle an den PC ermöglicht. Die Ein- und Ausgaben des Programms können so, mittels eines Terminalprogramms, per serieller Verbindung übertragen werden. Die bereits vorhandene Schnittstelle aus der Simulationssoftware soll dafür übernommen und erweitert werden.

Das Interface der Benutzerschnittstelle wird um den Parameter „TimeToLife“ erweitert.

Option:	Mögliche Parameter:	Beschreibung:
-TimeToLife	10 ms – 5000 ms	Gibt die Ausführungsdauer der gefundenen Aktion im Regelinterpreter an.

Der Parameter „TimeToLife“ gibt die Ausführungszeit pro Lebenszyklus vor. Der Wert wird in ms angegeben und bestimmt die Laufzeit der Antriebsmotoren für eine, in der Regelbasis gefundene, Aktion pro ausgeführtem Zyklus. Der Wert gibt somit die Reichweite an, die sich der Roboter während eines Zyklus bewegen kann. In der Simulation wird dieser Wert von der simulierten Umwelt selbst definiert (ein Karo pro Zyklus) und musste somit nicht explizit vorgegeben werden. Für eine bessere Kontrolle der verteilten Anwendung werden außerdem einige Statusmeldungen direkt über das auf dem Aksen-Board installierte LCD-Display ausgegeben.

Folgende Statusmeldungen werden über das LCD – Display ausgegeben:

- **Warte auf neues Genom:**
  Zeigt an, dass die Virtuelle Maschine auf die Übertragung eines neuen Genoms von dem genetischen Optimierer wartet.

- **Übertragung abgeschlossen:**
  Die Übertragung des Genoms und der Parameter ist abgeschlossen.

- **Teste Genom - Zyklus Nr. X von Y:**
  Zeigt an in welchem Zyklus X von insgesamt Y Zyklen sich der Regelinterpreter befindet.

- **Torsensorwerte / Ballsensorwerte / Abstandssensoren:**
  Bei der Fitnessmessung und GoHome Funktion werden die Sensorwerte zur Kontrolle angezeigt.

- **GoHome und Home gefunden:**
  Zeigt an, dass der Regelinterpreter einen kompletten Zyklus beendet hat und sich für die Bewertung der nächsten Regelbasis in die Ausgangsstellung begibt.

- **Fitness:**
  Nach jeder Fitnessmessung wird das aktuelle Zwischenergebnis zur Kontrolle ausgegeben.

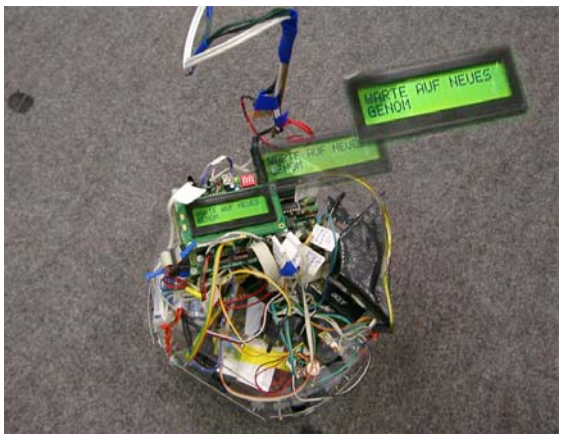


Abbildung 7.12: Aksenboard - Statusmeldung

7.1.8 Die Kommunikationsschnittstelle

Die Kommunikationsschnittstelle wird für die Übertragung der zu prüfenden Genome, Fitnesswerte und Parameter zwischen Evaluator und Virtueller Maschine benötigt. Der Datenaustausch wird bei der realen Roboteranwendung über den vorhandenen CAN-Bus realisiert. Die Übertragung soll über zwei Methoden Send(Parameter) und Receive(Parameter) implementiert werden. Für den Datenaustausch wird ein einfaches handshakebasiertes Protokoll eingesetzt. Die Methoden zur Datenübertragung sind in die Virtuelle Maschine und den Evaluator eingebettet. Folgende Funktionalitäten werden von den Methoden zur Datenübertragung unterstützt.

- **VM Bereitschaft Signalisieren:**
  Die Virtuelle Maschine kann dem Evaluator nach Durchführung einer Aufgabe signalisieren, dass sie bereit ist neue Aufgaben entgegen zu nehmen.

- **Versende Rahmenparameter für die Regelinterpretation:**
  Der Evaluator weist die Virtuelle Maschine an die definierten Rahmenparameter (Zykluslänge und Ausführungsdauer eines Zyklus) entgegen zu nehmen. Die abgeschlossene Übertragung wird von der Virtuellen Maschine mit OK quittiert.

- **Versende Genom zur Bewertung:**
  Der Evaluator weist die Virtuelle Maschine an, das aktuell zu bewertende Genom entgegen zu nehmen. Die Übertragung wird mit einer Empfangsbestätigung der Virtuellen Maschine abgeschlossen.

- **Warte auf Fitnesswert:**
  Der Evaluator signalisiert der Virtuellen Maschine, dass sie auf den aktuellen Fitnesswert wartet. Der Empfang des Fitnesswertes wird von dem Evaluator nach der Übertragung bestätigt.

Die Methoden basieren auf den mitgelieferten Bibliotheken zur Ansteuerung des CAN-Busses. Für die Kommunikation zwischen dem Evaluator und der Virtuellen Maschine werden folgende Messages definiert, die eine Synchronisation der Datenübertragung ermöglichen sollen:

KONSTANTE	WERT	BESCHREIBUNG
CAN_TRANSFER_OK	99	Übertragung erfolgreich
CAN_SEND_PARAMETRATION	11	Vorbereitung auf Parameterübertragung
CAN_SEND_RUNTIME	44	Laufzeit pro Aktion übertragen
CAN_SEND_CYCLES	55	Anzahl der Durchläufe pro Genom
CAN_SEND_GENOM	22	Neues Genom senden
CAN_SEND_FITNESS	33	Neuen Fitnesswert senden
CAN_WAIT_FOR_MESSAGE	66	Warte auf Nachricht
CAN_AWAIT_NEW_ORDER	1	Warte auf Order vom Lart-Board

Die Funktionen sind blockierend ausgelegt, um eine Synchronisation zwischen Evaluator und Virtueller Maschine zu ermöglichen. Für die Übertragung der Genome an die Virtuelle Maschine ist ein einfaches HandShake-Protokoll implementiert. Ein Ablaufchart mit der implementierten Datenübertragung ist im Anhang zu finden.

7.1.9 Erweiterung des Regelinterpreters

Wie bereits im Grundkonzept festgestellt, sind für den Fall, dass der Roboter fälschlicherweise die Arealbegrenzung berührt weitere Vorkehrungen nötig. Aus einer Kollision mit der Arealbegrenzung lässt sich schließen, dass die aktuelle Regelbasis unbrauchbar ist. Daher muss es innerhalb des Regelinterpreters einen übergeordneten Mechanismus geben, der Schaden am Roboter verhindert und die Ausführung der aktuellen Regelbasis beendet und bestraft. Für die Realtimeanwendungen soll daher eine Subsumptionsarchitektur eingesetzt

werden, welche die Ausführung der Regelbasis überwacht und bei Auslösen eines Bumpers in den aktuellen Vorgang eingreift. Die Abstrafung der Regelbasis soll, wie schon beim Simulator, mit einer Fitnessbewertung von 0 vollzogen werden.

Wie funktioniert eine Subsumptionsarchitektur und welche Vorteile bringt sie? Die Zeit ist in jeder Realtimeanwendung ein kritischer Faktor. Ein Problem bei Echtzeitanwendungen ist, dass zur Verarbeitung der Sensordaten eine gewisse Zeit benötigt wird. Je mehr Zeit ein Programm zur Auflösung widersprüchlicher Sensordaten, zur Modellierung eines Weltbildes und zur Findung einer optimierten Aktion braucht, desto größer ist die Gefahr, dass sich die Realität in der zwischen Zeit so radikal geändert hat, dass die Aktion nicht mehr zu dem gewünschten Ergebnis führt. Die Wahrscheinlichkeit, dass dieser Fall zutrifft, vergrößert sich mit der Zeitspanne zwischen Wahrnehmung und Handlung des Roboters. Mit Hilfe der Subsumptionsarchitektur, entwickelt von Rodney Brooks [BROOKS 2005], soll die Verzögerungszeit minimiert werden. Dies wird durch eine Architektur erreicht, die sich durch hierarchisch angeordnete Ebenen von Steuersystemen auszeichnet. Die einzelnen Ebenen werden durch angeschlossene Sensoren angesprochen und können parallel ablaufen. Widersprüchliche Sensordaten werden parallel in Aktionen umgewandelt. Durch eine Fusionierung der Aktionen, die mit Hilfe eines nach Prioritätsstufen unterteilten Auswahlschemas vollzogen wird, kann für jede Situation ein dominantes Verhalten bestimmt werden. Diese Vorgehensweise kann mit dem menschlichen Verhalten verglichen werden. Stellen sie sich vor sie stehen mit Ihrem Wagen auf den Bahngleisen, ihr Handy klingelt und ein Zug fährt heran. Beide Informationen verlangen eine Handlung von Ihnen. Die meisten Menschen würden in dieser Situation als erstes einmal die Bahngleise verlassen und danach das Handy abnehmen. Dies ist eine sinnvolle Priorisierung der Handlungen, da sie das Überleben sichert. Dieses Verhalten kann mit Hilfe der Subsumptionsarchitektur nachgebildet werden. Übertragen auf das Roboter-programm bedeutet dies, dass die Auslösung eines Bumpers in der Priorität höher steht, als die Ausführung der aktuellen Regelinterpretation. Das Verhaltensnetz für diese Aufgabe sieht folgendermaßen aus:

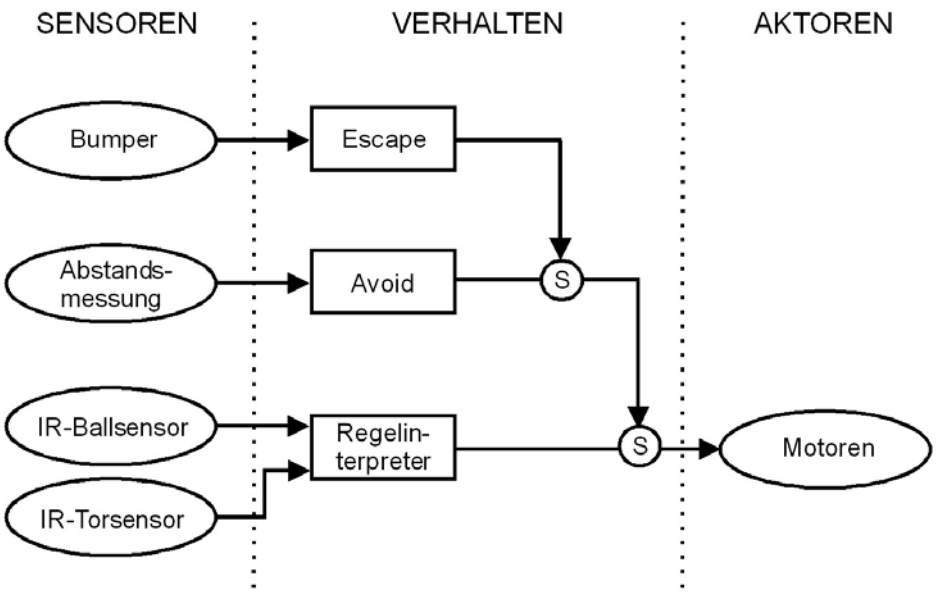


Abbildung 7.13: Subsumptionsarchitektur

Wie in der Abbildung zu erkennen ist, werden die implementierten Verhaltensweisen direkt von den Sensoren ausgelöst. Mit aufsteigendem Rang steigt die Priorität des entsprechenden Verhaltens. So kann ein höherwertiges Verhalten die niederwertigen Verhalten unterdrücken. Die Verhalten Escape, Avoid, und Regelinterpretation erzeugen eine Reihe von Signalen, die sich in den Unterbrecherknoten (S) treffen. Signale die von einer dominanten Verbindung her eintreffen (Pfeilspitzen), unterdrücken die Signale der unterlegenen Verbindungen. Sollten keine dominanten Signale eintreffen, werden die unterlegenen ausgegeben. Dies wird durch subsumption der einzelnen Verhaltensweisen erreicht. Da alle Verhaltensweisen parallel ablaufen, kann so recht schnell auf neue Sensorwerte reagiert werden. Die Nebenläufigkeit der Verhalten wird jeweils durch Zuordnung eines eigenen Prozesses realisiert.

Der Vorteil dieser Methode ist, dass kein Weltbild benötigt wird. Der Roboter ist so schnell in der Lage auf Veränderungen in der Umgebung zu reagieren. Die Ballsensoren und der Torsensor sind, wie in der Abbildung zu erkennen, mit dem Regelinterpreter verbunden. Im normalen Betrieb kann so die Fitnessmessung durchgeführt werden. Sollte sich der Roboter durch eine ungeeignete Regelbasis der Spielflächenbegrenzung zu stark nähern, greift das übergeordnete Verhalten „Avoid“ ein und fährt den Roboter ein Stück von der Wand weg, danach übernimmt die Regelbasis wieder das Verhalten. Gleichzeitig wird der Regelinterpreter angewiesen die Fitness der aktuellen Regelbasis zu verringern. Sollten die Sharpsensoren aufgrund ungünstiger Umstände versagen, greift das übergeordnete Verhalten „Escape“ und beendet die Regelinterpretation. Der Regelinterpreter wird dann angewiesen die Regelbasis mit der Fitness Null zu bewerten.

7.1.10 Realisierung des „Finde nach Hause“ Ablaufs

Der Regelinterpreter bewertet die zu prüfenden Regelbasen für die angegebene Anzahl von Zyklen und ermittelt mit Hilfe der Fitnessfunktion deren Qualität. Nach Ablauf eines jeden Zyklus, muss der Roboter in die definierte Ausgangsstellung zurückgebracht werden. Als Ausgangstellung wurde eine der oberen Ecken des gegnerischen Tores definiert. Um dieses Ziel zu erreichen, ruft der Regelinterpreter nach einem abgeschlossenen Zyklus die Funktion „GoHome()“ auf. Die „GoHome()“ Funktion übernimmt die Steuerung und fährt den Roboter mit Hilfe der Abstands- und Torsensoren in die gewünschte Ausgangsposition zurück.

Der Ablauf ist im folgenden Pseudocode verdeutlicht:

1. Drehe den Roboter so lange auf der Stelle, bis das gegnerische Tor im rechten oder linken Bereich des Torsensors erkannt wurde.

2. Fahre so lange gerade aus, bis die Abstandssensoren die Bande erkannt haben.

3. Drehe den Roboter um 180°.

4. Fahre 1,5 Sekunden gerade aus und stoppe. Zielposition erreicht.

6. Gib die Kontrolle zurück an die Virtuelle Maschine.

Da vorgesehen ist, dass der Roboter im Versuchsumfeld über eine Kabelpeitsche mit Strom versorgt wird, dreht der Roboter sich abwechselnd links und rechts herum. So soll ein Verknoten der Betriebsleitungen verhindert werden.

7.2 Probleme

Im folgenden Abschnitt sind noch einmal die Probleme, die bei der Erstellung der Software und mit der Technik aufgetreten sind, zusammengefasst. Die meisten Probleme sind im Zusammenhang mit der Roboterelektronik und -mechanik aufgetreten. Insbesondere die hohe mechanische Anfälligkeit der Roboterkonstruktion ist hierbei problematisch gewesen.

7.2.1 Elektronik

Problem: Stromversorgung über Akkupacks

Von dem Entwickler des Roboters war vorgesehen, die Elektronik mit Hilfe verschiedener Akkupacks mit Strom zu versorgen. Da die Versuchsreihen aber voraussichtlich mehrere Stunden Betriebszeit in Anspruch nehmen würden, waren Akkus als Stromquelle unbrauchbar. Daher musste eine Stromquelle gefunden werden, die genügend Leistung bringt, um die stromhungrigen Antriebsmotoren und das RCUBE-System betreiben zu können (ca. 2 A bei voller Last) und die darüber hinaus unterschiedliche Spannungen zur Verfügung stellen kann.

Lösung: Zur Lösung diese Problems wurde ein ausgemustertes PC-Netzteil, das 12V und 5V mit ausreichender Leistung zur Verfügung stellt, sowie ein einstellbares Labornetzteil eingesetzt. So konnten alle Komponenten, inkl. des nachgebauten stationären Infrarotballs, über längere Zeit betrieben werden. Der Strom wurde den Geräten über eine freihängende Kabelpeitsche zugeführt.

Problem: Stromversorgung – Spannungsabfall bei Betrieb der Antriebsmotoren dadurch Reset des RCUBE-Systems.

Bei gleichzeitigem Einsatz aller drei Antriebsmotoren ist es immer wieder zu Störungen im Betrieb des RCUBE-Systems gekommen. Es wurde festgestellt, dass die Spannung bei zu hohen Strömen zusammenbricht und so einen Reset des Lart-Boards auslöste.

Lösung: Erhöhung des Leitungsquerschnitts der Versorgungsleitung des RCUBE-Systems. Um den Leitungswiderstand gering zu halten, wurde die Zuleitung durch ein handelsübliches Lautsprecherkabel mit einem Querschnitt von 2,5 mm² ersetzt. Dadurch konnten Spannungseinbrüche effektiv verhindert werden.

7.2.2 Mechanik

Die schwerwiegendsten Probleme sind im Zusammenhang mit der Mechanik des Roboters aufgetreten. Die Bauweise des zur Verfügung stehenden Roboters ist nicht auf längere Nutzung ausgelegt. Insbesondere die Verwendung von Plastikteilen hat zu einem schnellen Verschleiß des Geräts geführt.

Problem: Durchdrehende Antriebsräder

Während der verschiedenen Fahrmanöver kommt es immer wieder zum Durchdrehen einzelner Räder. Insbesondere wenn nur eines der Antriebsräder angesteuert wird, neigt dieses dazu auf der Stelle zu drehen.

Lösung: Als Lösung wurde das Gewicht des Roboters durch Ballast in Form von Akkus erhöht und gleichzeitig darauf geachtet, dass das Gewicht gleichmäßig auf der Plattform verteilt ist. Der Reibungswiderstand konnte dadurch etwas verbessert werden. Bei der Ermittlung des richtigen Gewichts musste ein Kompromiss zwischen gewonnenem Grip und Gewicht gefunden werden. Da ein zu hohes Gewicht die Motoren überlastet hätte, tritt ein Durchdrehen der Räder in seltenen Fällen immer noch auf. Das Problem konnte somit nicht gänzlich beseitigt werden.

Problem: Zahnräder Antriebseinheit

Bei längeren Fahrten neigen die zum Einsatz kommenden Plastikzahnräder dazu zu brechen. Bedingt durch die Bauart, ist der Verschleiß der Plastikbauteile sehr hoch. Hier kann nur eine Neukonzeption des Antriebs auf Basis festerer Materialien Abhilfe schaffen.

Problem: Bumperleiste löst sich ab

Das Befestigungsprinzip der Bumperleiste des Roboters ist sehr störanfällig. So fällt diese nach einigen Zusammenstößen immer wieder ab. Auch die Verwendung eines speziellen Klebers konnte hier keine Abhilfe schaffen.

Problem: Kabelbrüche im Versorgungskanal

Durch die drehbaren Teile des Ballsensors, kommt es immer wieder zu Kabelbrüchen an den Lötstellen der Ballsensoren und im Inneren des Kabelkanals selbst. Die Kabelstränge werden durch Bewegung der Sensorarme mechanisch belastet und neigen dadurch zu Kabelbrüchen.

Lösung: Die Lötstellen an den Sensoren wurden durch Heißkleber mechanisch entlastet. Ein Brechen der Kabel innerhalb des Kabelkanals kann bei längerer Belastung nicht ausgeschlossen werden.

Problem: Mechanik der beweglichen Ballsensorarme

Der bewegliche Arm der Ballsensoren ist auf der untersten Plattform gelagert und wird durch eine mittlere Schraube gehalten. Bei längerem Einsatz der Ballsensoren entsteht ein Abrieb auf den belasteten Plastikteilen, dadurch neigt der Arm zum Verkanten auf der Plattform. Die Arretierung des Arms muss so ständig nachgestellt werden. Dieses Problem kann nur durch eine Neukonzeption des Sensors beseitigt werden.

7.2.3 Software

Problem: Multitasking in Verbindung mit der Torerkennung

Für die Subsumptionsarchitektur ist die Verwendung von verschiedenen Tasks für die einzelnen Verhalten unumgänglich. Leider hat sich herausgestellt, dass die Torerkennung in Verbindung mit der Anwendung von verschiedenen Tasks nicht mehr zuverlässig funktioniert. Dieses Problem ist in dem verwendeten Betriebsystem des Lart-Boards begründet.

Lösung: Da die Verwendung von verschiedenen Tasks nicht möglich war, ohne die Torerkennung zu stören, wurde versucht das Programm an den entsprechenden Stellen zu serialisieren. Die Bumpererkennung und die damit verbundene Reaktionszeit auf das Ereignis hat sich dadurch auf eine Sekunde erhöht. Diese Verzögerung konnte in Kauf genommen werden. Die frühzeitige Erkennung der Bande durch die Sharpsensoren und die geplante Vermeidungstechnik musste entfernt werden. Mittlerweile gibt es ein neues Betriebsystem für das Lart-Board, dass eine zuverlässige Torerkennungsroutine in Verbindung mit mehreren Tasks zulässt.

8 DIE ROBOTERVERSUCHE

8.1 Versuchsvorbereitung

Da die geplanten Versuchsreihen über einige Stunden laufen werden, sind bezüglich der Stromversorgung und Archivierung der ermittelten Daten Vorbereitungen zu treffen.

Die Stromversorgung:

Der Roboter wird standardmäßig mit zwei Akkupacks betrieben, welche die RVISION-Plattform und eine Schaltung zur Verstärkung der Abstandssensoren versorgen. Die Akkupacks haben eine Betriebsdauer von max. 1 Stunde, reichen also für eine kontinuierliche Stromversorgung während des Experiments nicht aus. Daher ist geplant den Roboter über eine Kabelpeitsche zu versorgen. Da verschiedene Spannungen benötigt werden, soll zur Stromversorgung ein Labornetzgerät, sowie ein handelsübliches PC Netzteil eingesetzt werden. Der Infrarotball der Firma Wiltronics wird über einen integrierten 9 Volt Akku betrieben und ist somit ebenfalls nicht für einen Dauerbetrieb ausgelegt. Da sich der Ball nicht, ohne diesen zu beschädigen, über eine externe Stromquelle versorgen lässt, wurde ein einfacher stationärer Infrarotsender nachgebaut. Der Infrarotsender besteht aus fünf in Reihe geschalteten Infrarotdioden. Der stationäre Infrarotsender wird über die 5 Volt Versorgungsspannung des PC Netzteils mitbetrieben.

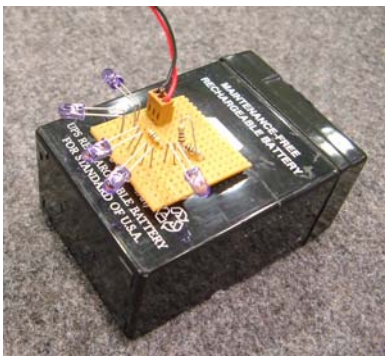


Abbildung 8.1: Stationärer Infrarotsender - IR-Ball

Datenaufzeichnung:

Die RVISION Plattform verfügt standardmäßig über eine flüchtige Ramdisk, welche für die Protokollierung der anfallenden Daten nicht geeignet ist. Ein im Fachbereich entwickelter persistenter Speicher auf CompactFlash Basis, passend zur RCUBE Architektur, war zum Zeitpunkt dieser Diplomarbeit noch nicht verfügbar. Mein Kommilitone Lars Brandt beschäftigte sich zur Zeit im Rahmen seiner Diplomarbeit [BRANDT 2005] mit der Bluetooth Technologie. So kam die Idee auf, die Übertragung der Protokolldaten über eine Bluetooth Verbindung vorzunehmen. Für diesen Zweck wurde ein Bluetoothmodul so modifiziert, dass es sich direkt an die RS232 Schnittstelle des Lart-Boards anschließen lässt. Auf der PC Seite wurde über ein handelsübliches USB Dongle die Verbindung zum Linux Terminal des Lart-Boards hergestellt. So war es möglich, die Ausgaben der Software über ein Terminalprogramm zu protokollieren.

Abbildung 8.2: Spannungsversorgung - Bluetoothsendemast

Systemvorbereitung:

Um das RVISION System in den Ausgangszustand zu bringen, ist es nötig die verteilte Anwendung auf das System aufzuspielen. Das Aksen-Board wird ebenso wie das Lart-Board über die integrierte RS232 Schnittstelle mit den erzeugten Programmen versorgt. Nach dem Reset des Aksen-Boards wartet die Vorgangssteuerung der Virtuellen Maschine auf Anweisung vom Lartboard (CanBus). Das Programm des Lart-Boards wird über die Eingabezeile mit den ausgewählten Parametern gestartet. Nach dem Starten der Hauptanwendung übernimmt diese die Kontrolle über die Virtuelle Maschine und der genetische Prozess beginnt.

8.2 Versuchsdurchführung

Die Versuchsdurchführung soll zeigen, ob auch komplexeres Verhalten mit einem genetischen Algorithmus erlernbar ist. Dabei sollen die im Simulatorversuch erfolgreich ermittelten Rahmenparameter verwendet werden. Für die Parameter „Cycles“ und „TimePerCycle“ wurden mit Hilfe einer Versuchsreihe geeignete Werte ermittelt. Beide Parameter beschreiben die Reichweite, die der Roboter während eines kompletten Zyklus (Fitnessmessung) auf der Spielfläche zurücklegen kann. Hierbei ist darauf zu achten, dass der Roboter einerseits ausreichend Strecke zurücklegen kann, um das gewünschte Ziel erreichen zu können, andererseits die Reichweite nicht zu groß zu wählen, um unnötige Kollisionen mit der Spielflächenbande zu verhindern. Für die Antriebsdauer der Motoren pro Cycle haben sich 500ms als geeignet erwiesen. In dieser Zeit kann der Roboter eine Strecke von ca. fünf Zentimetern zurücklegen – eine ausreichende Strecke, um sichtbare Veränderungen der Roboterposition hervorzurufen. Auf der anderen Seite ist die Strecke klein genug, um die, sich durch die Bewegung geänderten, Umweltdaten ausreichend genau erfassen zu können. Eine Reaktion auf die sich verändernde Umwelt, kann so in einem sinnvollen Zeitrahmen vollzogen werden.

Im Gegensatz zum simulierten Roboter, ist die Zeit ein nicht zu vernachlässigender Faktor bei der Evaluierung einer geeigneten Lösung durch den Algorithmus. Der Zeitbedarf für die Fitnessermittlung eines Genoms lässt sich mit folgender Formel grob berechnen:

$$t_{fitness} = (cycles * timepercycle * t_{sensor}) + t_{process}$$

Der Wert $t_{process}$ und t_{sensor} sind großzügig geschätzte Zeiträume, die durch Beobachtung (Status LCD Display) ermittelt wurden. Folgende Vorgänge innerhalb des Systems werden dabei abgedeckt:

Die Zeit $t_{process}$ setzt sich folgendermaßen zusammen:

- Datenübertragung zwischen Lart-Board und Aksen-Board (ca. 1 Sek.)
- Genetischer Prozess - Erzeugung der Nachfolgegeneration (ca. 2 Sek)

Daraus ergibt sich für $t_{process}$ eine Zeitspanne von ca. 3 Sekunden.

Der Wert t_{sensor} ist der Zeitraum, der zur Erfassung der Sensorwerte nach einem ausgeführten Cycle benötigt wird. Die Erfassung der Sensorwerte dauert, aufgrund der teilweise durch Servos zu bewegenden Sensoren, recht lange und beträgt ca. 10 Sekunden pro Messung.

Aus den erfassten Werten ergibt sich dann folgender Zeitverbrauch für die Evaluierung der Qualität eines Genoms:

$$t_{fitness} = 105\ Sek = (7 * 0{,}5\ Sek * 10\ Sek) + 3\ Sek$$

Aus der Größe der Population und der Anzahl der zu erzeugenden Generationen, lässt sich mit Hilfe des gewonnenen Wertes $t_{fitness}$ eine grobe Abschätzung vornehmen, wie lange der genetische Prozess laufen wird. Eine Populationszahl von 15 hat sich in dem Simulatorversuch als brauchbar herausgestellt und soll hier weiterverwendet werden. In Anbetracht des Zeitbedarfs, sollen die Versuche mit einer Generationenanzahl von max. 20 gestartet werden. Da in den Simulatorversuchen stets vor der 15. Generation ein brauchbares Genom gefunden wurde, sollte die Erzeugung von 20 Generationen ausreichen. Aus den festgelegten Werten ergibt sich dann eine ungefähre maximale Laufzeit von 8,75 Stunden:

$$t_{evolve} = PopSize * Generations * t_{fitness}$$

$$t_{evolve} = 8{,}75\ Std = 15 * 20 * 105\ Sek$$

Wie die Simulatorversuche außerdem gezeigt haben, sind die Ausgangslagen durch die zufällige Erzeugung der Urpopulation sehr unterschiedlich. Dies kann zur Folge haben, dass der Algorithmus, je nach Ausgangsqualität der Genome, unterschiedlich schnell gegen ein brauchbares Ergebnis konvergiert. Aus diesem Grund sind insgesamt drei Roboterversuche mit den ermittelten Parametern geplant.

Folgende Versuchsparameter sollen für die drei Versuche eingesetzt werden:

Die Versuchsparameter:

Parameter:	**Wert:**
Populationsgröße	15
Anzahl der Generationen	20
Elitarismusfunktion	Ein
Lebenszyklen pro Genom	7
Ausführungsdauer pro Regel	500 ms
Kreuzungswahrscheinlichkeit	40%
Mutationswahrscheinlichkeit	5%
Direkte Selektion	55%

8.2.1 Der 1. Versuch

Ergebnisse des 1. Versuchs:

Der erste Versuch konnte nicht wie geplant durchgeführt werden. Nach einer Laufzeit von ca. einer Stunde trat ein mechanischer Defekt an der Antriebseinheit des Roboters auf. Eines der Antriebszahnräder war zerbrochen. Der Versuch musste daher vorzeitig beendet werden. Wie sich herausstellte, war die Bluetoothdatenübertragung während des Versuchs sehr störanfällig und ist mehrfach abgebrochen, so dass die Aufzeichnungen nicht richtig durchgeführt werden konnten. Der erste Versuch hat daher zu keinen brauchbaren Ergebnissen geführt.

Schlussfolgerungen:

Die Antriebseinheit wurde nach dem ersten Versuch überprüft und das defekte Zahnrad ersetzt. Um die Verbindungsqualität der Bluetoothübertragung zu verbessern, wurde ein Sendemast mit dem Bluetoothdongle direkt neben der Spielfläche platziert. Weitere Verbindungsabbrüche sollten so vermieden werden.
Da weitere mechanische Defekte während der Versuche möglich waren, wurde die Software um eine zyklische Ausgabe für das beste Genom erweitert, die nach jeder evaluierten Generation das Elitegenom ausgegeben würde. So sollte auch bei einem frühzeitigen Abbruch des Versuches ein Ergebnis gesichert werden können.

8.2.2 Der 2. Versuch

Der zweite Versuch musste wegen eines mechanischen Defekts nach ca. 2 Stunden Laufzeit abgebrochen werden. Die Arretierung der Sensorplattform hatte sich gelöst und zum Versagen der drehbaren Ballsensoren geführt. Trotz des vorzeitigen Abbruchs konnte der genetische Algorithmus die Urpopulation bis in die fünfte Generation weiterentwickeln. Die aufgezeichneten Daten haben sich dank der verbesserten Übertragungsqualität als brauchbar erwiesen. Durch die Softwareverbesserung nach dem ersten Versuch, konnte auch das bis dahin beste gefundene Genom gesichert und das Bewegungsmuster per Videoaufzeichnung dokumentiert werden. Das beste gefundene Genom hat einen Fitnesswert von 220 Punkten erreicht.

Ergebnisse des 2. Versuchs:

In der folgenden Abbildung ist das erzeugte Bewegungsmuster des, bis zum Abbruch des Versuchs gefundenen, besten Genoms dokumentiert. Die Darstellung des Verhaltens in der Grafik wurde aus einer Videoaufzeichnung gewonnen. Die Fahrt des Roboters konnte daher nicht Zentimeter genau vermessen werden. Da die Grafik aufgrund von Beobachtungen entstanden ist, stellt sie keinen Anspruch an quantitative Genauigkeit, sie soll lediglich zur subjektiven Bewertung des gefundenen Genoms beitragen.

Qualitatives Bewegungsmuster des gefundenen Genoms:

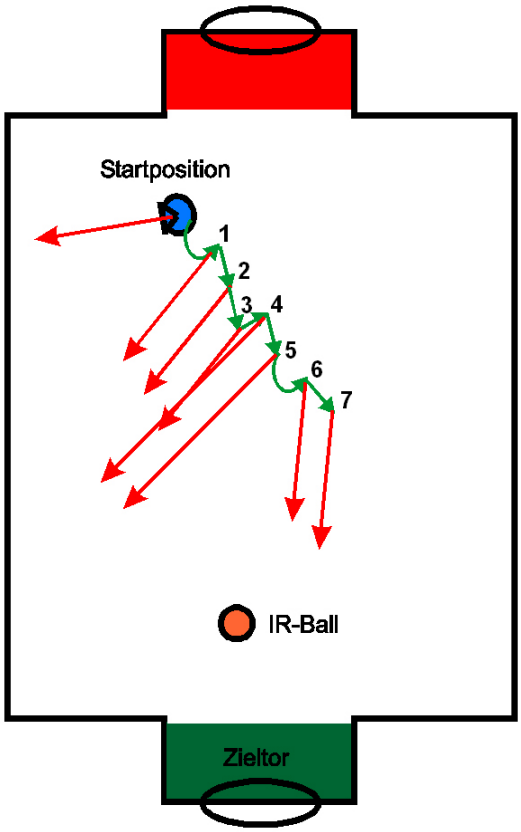


Abbildung 8.3: Roboterversuch 2 - Bewegungsmuster

In der Abbildung wird der Bewegungsablauf des Roboters durch die grünen Pfeile und die darüber angeordneten Zahlen dargestellt. Die von den Zielpunkten der grünen Pfeile ausgehenden roten Pfeile, zeigen die Ausrichtung des Roboters nach Beendigung eines Zwischenschrittes an. Dies ist für die Bewertung wichtig, da sich der Roboter ja mit seiner Front in eine Linie zum Infrarotball und dem Zieltor bewegen soll.

In der Startposition ist der Roboter mit seiner Front fast genau nach Westen ausgerichtet. Das Tor und der Ball befinden sich also relativ weit entfernt von der Front des Roboters. Im ersten Schritt macht der Roboter einen ca. 45 Grad weiten Bogen in Richtung Südosten und nähert sich mit seiner Front dem Tor und dem Ball an. In den Schritten zwei und drei fährt der Roboter auf einer Geraden, ohne seine Ausrichtung zu ändern, in Richtung Tor. Die Bewegung im vierten Schritt stellt eine Besonderheit dar, da diese nicht wie vorgesehen ausgeführt werden konnte. Bei dem Versuch sich zu bewegen, drehte eines der Antriebsräder auf dem Teppich durch. Das Ergebnis ist eine sehr kleine Bewegung in Richtung Nordosten. Die Ausrichtung hat sich dabei, wie im Schaubild zu erkennen, nur minimal zu ungunsten des Ziels geändert. In Schritt 5 wird die Ausrichtung durch eine weitere Drehung von 45 Grad auf die Ziele

verändert. Das primäre Ziel, die Roboterfront in einer Line auf Ball und Tor auszurichten, wurde somit fast erreicht.

Analyse der Versuchsergebnisse:

In dem nachfolgenden Chart ist eine grafische Analyse des Fitnessverlaufes über die erzeugten Generationen zu sehen:

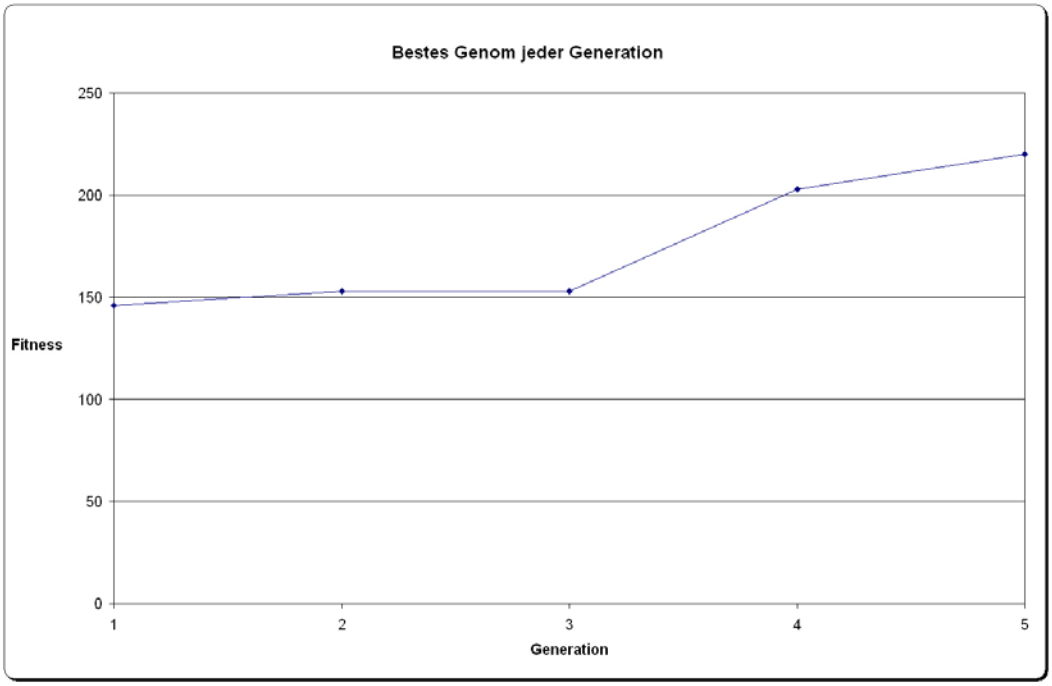


Abbildung 8.4: Roboterversuch 2 - Bestes Genom jeder Generation

Aus dem Chart lässt sich ablesen, dass die Fitness des besten Urgenoms kontinuierlich über die fünf evaluierten Generationen verbessert werden konnte. Der ursprüngliche Fitnesswert von 146 Punkten konnte um 76 Punkte auf 222 Punkte verbessert werden. Das beste Genom wurde in der fünften und letzten erzeugten Generation gefunden. Aus der Verbesserung des Genommaterials lässt sich schließen, dass der evolutionäre Prozess gegriffen hat.

In dem nächsten Chart ist die durchschnittliche Fitness der Generationen aufgeführt:

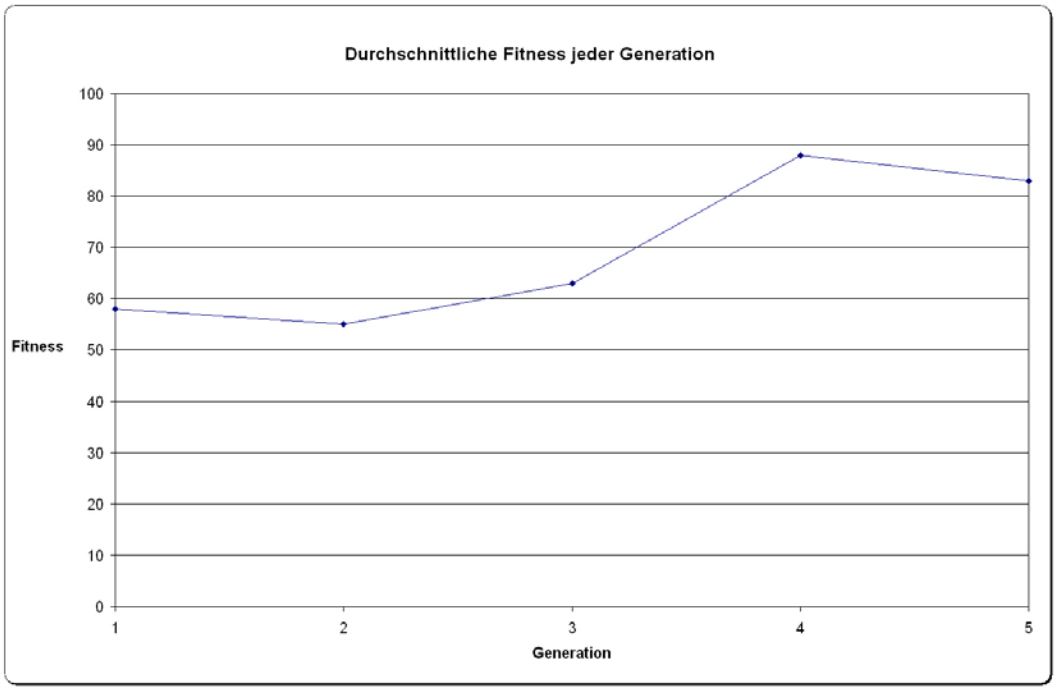


Abbildung 8.5: Roboterversuch 2 - durchschnittliche Fitness

Über die fünf erzeugten Generationen ist eine deutliche Schwankung der durchschnittlichen Fitnesswerte zu erkennen. Diese erreichen in der vierten Generation mit knapp 90 Punkten ihren Höhepunkt.

Schlussfolgerungen:

Aus der subjektiven Bewegungsanalyse lässt sich ablesen, dass das gefundene Genom in der Lage ist den Roboter in Richtung Tor und Ball auszurichten. Hierbei muss allerdings beachtet werden, dass dies nicht zwingend auch für andere Ausgangsstellungen des Roboters zutreffen muss. Die grafische Analyse zeigt deutlich, dass das Genommaterial über die fünf erzeugten Generationen verbessert werden konnte. Diese Ergebnisse lassen darauf schließen, dass das vorgegebene komplexe Verhalten über den vorgegebenen genetischen Algorithmus erlernbar ist. Das in die Fitnessfunktion kodierte Problem scheint ausreichend genau abgebildet.

8.2.3 Der 3. Versuch

Der dritte Versuch musst nach 4 Stunden wegen diverser Hardwaredefekte abgebrochen werden. Der Roboterantrieb, sowie der bewegliche Ballsensor wurden immer unzuverlässiger und fielen schließlich ganz aus. Der dritte

Roboterversuch lief trotzdem deutlich länger als der zweite Versuch. Die Ursprungsgenome konnten immerhin bis zur neunten Generation weiterentwickelt werden. Das beste gefundene Genom hat einen Fitnesswert von 246 Punkten erreicht.

Ergebnisse des 3. Versuchs:

Bewegungsmuster des gefundenen Genoms:

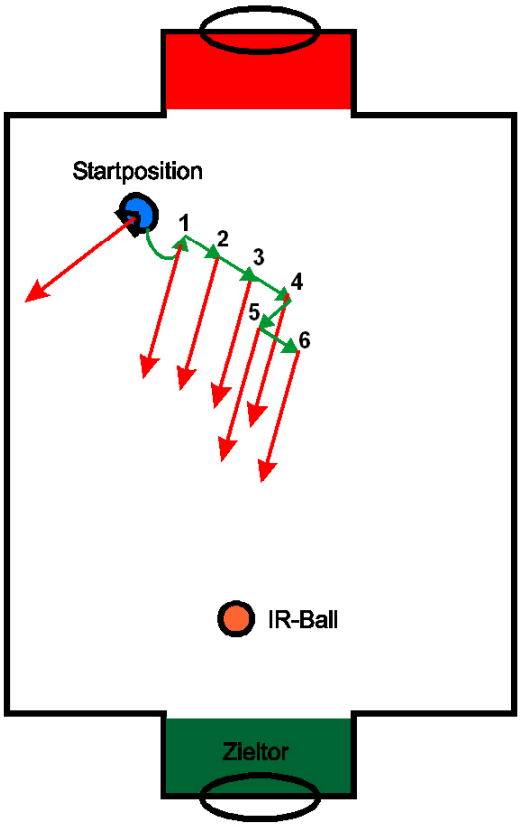


Abbildung 8.6: Roboterversuch 3 - Bewegungsmuster

In der Startposition ist der Roboter bei der Ausführung des gefundenen Genoms in Richtung Südwest ausgerichtet. Wie schon im zweiten Versuch dreht sich der Roboter im ersten Schritt mit einer Drehbewegung in Richtung Tor und Ball. Für die nächsten drei Schritte wird eine gerade Fahrt durchgeführt, ohne dass sich dabei der Blickwinkel des Roboters verändert. Durch die gerade Fahrt nähert sich die Blickrichtung allerdings immer mehr der gewünschten Linie (Tor und Ball). In den letzten beiden Zyklen vollzieht der Roboter einen Zick Zack Kurs, wiederum ohne sich dabei zu drehen. In der Position sechs wurde dann das Ziel erreicht, die Roboterfront in einer Linie auf Tor und Ball auszurichten.

Analyse der Versuchsergebnisse:

In dem nachfolgenden Chart ist eine grafische Analyse des Fitnessverlaufes über die erzeugten Generationen zu sehen:

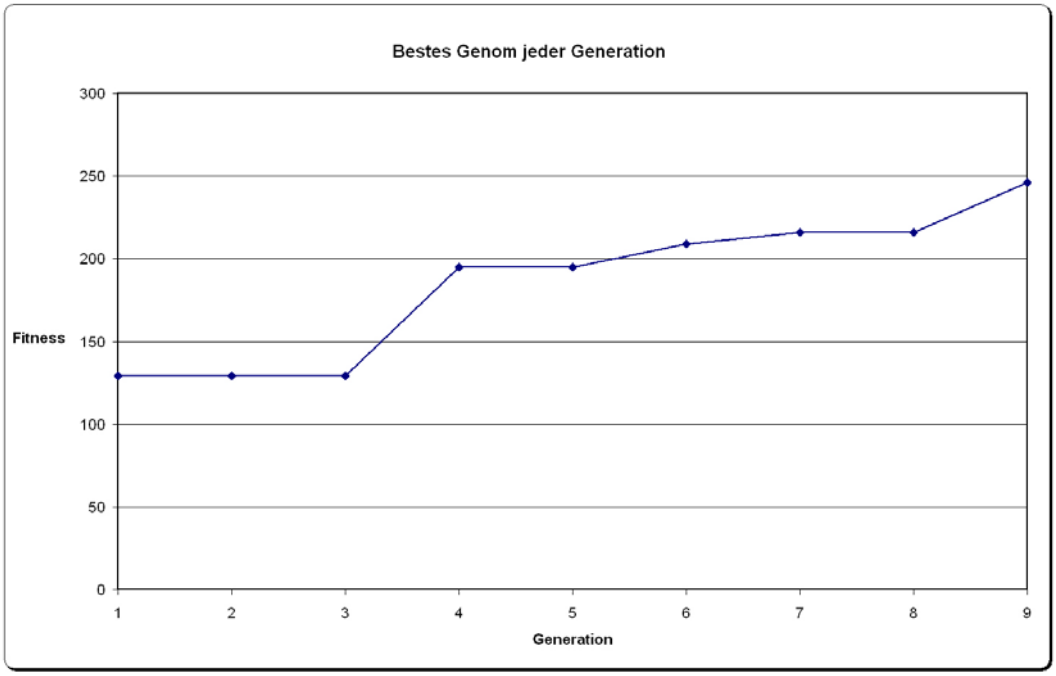


Abbildung 8.7: Roboterversuch 3 - bestes Genom jeder Generation

Die grafische Analyse der besten Genome pro Generation zeigt deutlich, dass eine evolutionäre Verbesserung der Genomqualität über die neun erzeugten Generationen erreicht wurde. Die Fitness des besten Urgenoms konnte von 129 Punkte um 117 Punkte auf 246 Punkte verbessert werden. Vergleicht man das beste Ursprungsgenom (129 Punkte) mit dem besten Urgenom des vorherigen Versuchs (146), so fällt die geringere Qualität der Urpopulation auf. Vergleicht man jetzt außerdem die Werte in Generation 5 mit denen des vorherigen Versuchs, zeigt sich ein Unterschied (V2=222 Punkte – V3=195 Punkte) von 27 Punkten. Die schlechtere Qualität der Urpopulation scheint sich somit auf den weiteren Verlauf des evolutionären Prozesses auszuwirken. Der zweite Versuch hatte mit seinem besseren Ausgangsmaterial in Generation 5 im Vergleich bereits einen Vorsprung von 27 Punkten. Dies ist allerdings nur eine These und müsste in umfangreichen Versuchsreihen bestätigt werden.

In dem nächsten Chart ist die durchschnittliche Fitness der Generationen aufgeführt:

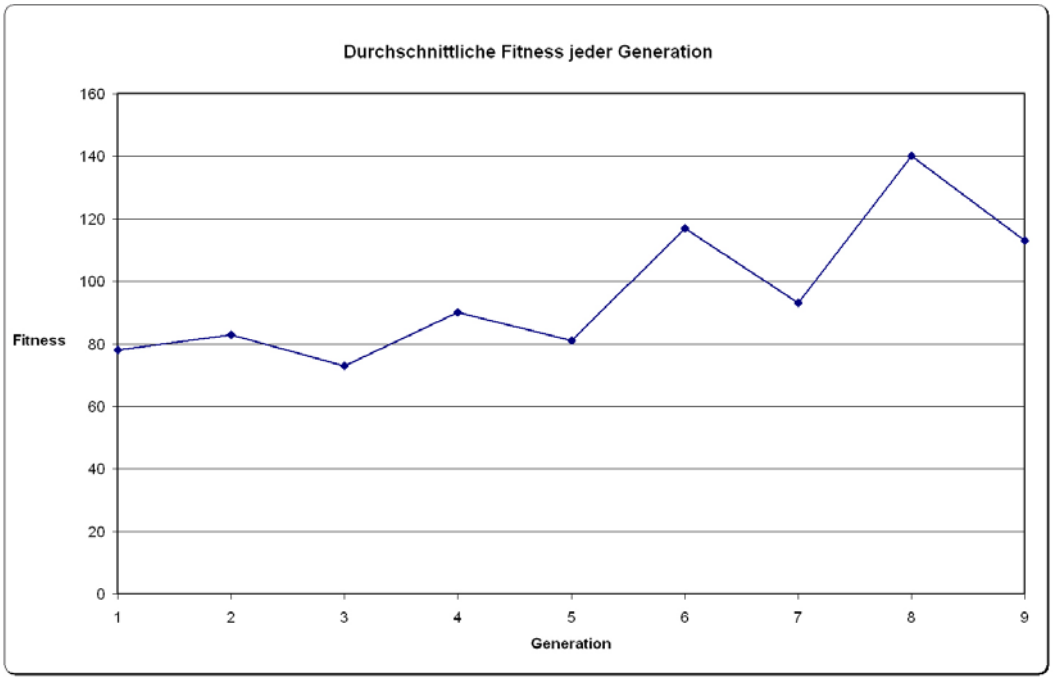


Abbildung 8.8: Roboterversuch 3 - durchschnittliche Fitness

Wie schon in den vorherigen Versuchen, bei denen der evolutionäre Prozess gegriffen hat, ist eine deutliche Schwankung der durchschnittlichen Fitness pro Generation zu erkennen. Auffällig ist hierbei, dass sich die Amplitude mit steigender Generationenzahl scheinbar vergrößert. Im Vergleich mit dem vorherigen Versuch ist die durchschnittliche Fitness der Generationen um ca. 20 bis 30 Punkte besser. Diese Ergebnisse lassen vermuten, dass nicht die Höhe der durchschnittlichen Fitness pro Generation ausschlaggebend für eine gute evolutionäre Basis ist, sondern die überdurchschnittlich hohe Fitness einzelner zufällig erzeugter Individuen der Urpopulation.

Schlussfolgerungen:

Sowohl die Auswertung des subjektiven Bewegungsmuster, als auch die grafische Analyse zeigen, dass das in der Fitnessfunktion kodierte Ziel erreicht werden konnte.

8.3 Ergebnisse / Zusammenfassung

Die Versuchsreihen zeigen, dass das gewünschte komplexe Verhalten mit Hilfe des gewählten genetischen Algorithmus erlernbar ist. Weiterhin lassen die

Ergebnisse vermuten, dass eine Korrelation zwischen einzelnen zufällig erzeugten hochwertigen Genomen in der Urpopulation zu einem positiven Verlauf der Evolution (schnellere Erhöhung der Fitness) führt. Außerdem ist auffällig, das die Versuchsergebnisse bei einem direkten Vergleich der Werte, darauf hindeuten, dass nicht die Höhe der durchschnittlichen Fitness pro Generation ausschlaggebend für eine gute evolutionäre Basis ist, sondern die überdurchschnittlich hohe Fitness einzelner Individuen – also dass trotz starker Schwankungen der durchschnittlichen Fitness, die Qualität der Fitness der besten Genome kontinuierlich steigt. Dies sind allerdings nur Thesen und müssten mit Hilfe weiterer Versuchsreihen bestätigt werden.

Kritisches zur Fitnessfunktion:

Das in die Fitnessfunktion kodierte Problem scheint auf den ersten Blick ausreichend genau abgebildet zu sein. Die in den Versuchen gefundenen Regelbasen richten den Roboter relativ gut auf einer Linie mit Tor und Ball aus. Dabei ist allerdings zu beachten, dass die gefundenen Regelbasen lediglich für die im Versuch aufgezeichneten Ausgangsstellungen wie gewünscht funktionieren. Anders als im Simulatorversuch, wo die Startstellung des Roboters immer gleich war, ist eine exakte Ausrichtung des Roboters auf die Startposition nur sehr ungenau möglich gewesen. Daraus folgt, dass das gewonnene Elitegenom einer Generation nicht zwingend eine gleichbleibende Qualität in der Nachfolgegeneration erzeugen muss. Dies hat auch die Auswertung der Versuchslogs ergeben (Vergleich der Fitness jedes ersten Genoms einer Generation). Ein Test, ob sich die gefundenen Regelbasen auch bei unterschiedlichen Startpositionen bewährt hätten, konnte leider nicht mehr durchgeführt werden. Nach den Versuchsreihen waren zwei der Antriebsmotoren defekt. Eine weitere kritische Frage bleibt offen: Hätte sich auch ein Elitegenom bilden können, dass sich beim Ausrichten von den Zielen weiter entfernt, anstatt auf sie zu zufahren? In der Fitnessfunktion ist lediglich ein kleiner Anreiz[9] kodiert, dem Ziel Tor näher zu kommen. Dieser ist im Verhältnis zur Gesamtbewertung aber eher klein und kommt auch nur bei direkter Ausrichtung auf das Tor zum tragen. In den Versuchsreihen waren immer wieder Regelbasen zu erkennen, die sich scheinbar auf die Ziele ausrichteten, sich dabei aber von den Zielen wegbewegten. Um zu prüfen, ob sich auch ein Genom mit diesem Verhalten hätte durchsetzen können, wären weitere Versuche nötig gewesen. Die Simulatorversuche haben analog hierzu deutlich gezeigt, welche Auswirkungen das Fehlen eines Aspekts in der Fitnessfunktion bewirken kann (Vor- und Zurückfahren auf der Stelle ohne Streckenmessung). Eine genauere Entfernungsmessung, um festzustellen wie weit weg sich der Roboter von den Zielen befindet, wäre mit den benutzen Sensoren nicht möglich gewesen. Um dieses Problem zu lösen, müsste über eine andere oder erweiterte Sensorik nachgedacht werden. Hier wäre evtl. eine externe Kamera, die eine automatische Distanzmessung zwischen den Objekten ausführen kann und so eine weitere Bewertungsgrundlage bietet, sinnvoll gewesen.

[9] Torsensorbesonderheit – Bei direkter Gegenüberstellung zum Tor gibt es zwei unterschiedliche Messwerte für eine nahe und ferne Stellung des Roboters.

Zusammenfassend kann gesagt werden, dass die Versuche gezeigt haben, dass das gewünschte Ziel mit den gewählten Verfahren erlernbar ist.

Alle Versuchsergebnisse, Videodokumente und die erstellte Software befinden sich auf der beigefügten CD-Rom.

8.4 Probleme während des Versuchs

Die meisten Probleme mit der Mechanik, die schon während der Realisierungsphase bekannt wurden und nicht gelöst werden konnten, sind bei der Ausführung der Versuchsreihen erneut aufgetreten.

Folgende Problem sind erneut aufgetreten:

- Durchdrehende Antriebsräder
- Gebrochen Zahnräder der Antriebseinheit
- Ablösen der Bumperleiste
- Versagen der Mechanik der beweglichen Ballsensorarme

Ein neues Problem brachten die für den Antrieb umgebauten Servomotoren. Die aus dem Modellbau stammenden Motoren waren nicht für einen dauerhaften Gebrauch ausgelegt. Während des 3. Versuchs wurden die Motoren aufgrund von Abnutzungserscheinungen zerstört. Hier kann, ebenso wie bei der Zahnradproblematik, nur eine Neukonzeption des gesamten Antriebs Abhilfe schaffen.

Folgende zusätzlich Probleme sind bei der Elektronik aufgetreten:

Problem: Sporadische Abbrüche der Bluetoothübertragung

Bei der Datenübertragung zwischen dem RCUBE-System und dem PC, zur Aufzeichnung der gewonnenen Versuchdaten, ist es immer wieder zu Abbrüchen gekommen. Die Sendeleistung des eingesetzten Bluetoothdongle und der auf Roboterseite eingesetzten Bluetoothplatine war für die ursprünglich zu überbrückende Strecke nicht ausreichend.

Lösung: Das Bluetoothdongle wurde mit Hilfe eines USB Verlängerungskabels und eines Sendemastes direkt an der Spielfläche platziert. Verbindungsabbrüche konnten so vermieden werden.

Problem: Torerkennung im Zusammenhang mit der GoHome Funktion

Während die GoHome Funktion ausgeführt wird, dreht sich der Roboter um das Heimattor zu finden. Hierbei kommt es sporadisch zu Fehlerkennungen des

Tores. Fälschlicherweise wird das gegnerische Tor als Heimattor erkannt. Dies ist höchstwahrscheinlich dadurch zu erklären, dass bei einer schrägen Blickrichtung auf das gegnerische Tor die Impulse der Torbarke lückenhaft gezählt werden und so die falsche Frequenz erkannt wird.

Lösung: Da der Fehler systembedingt auftritt, konnte hierfür keine technische Lösung gefunden werden. Als Workaround wurde der Roboter während der Versuchsreihen bei Anfahrt auf das falsche Tor per Hand auf die richtige Startposition gesetzt.

9 RESÜMEE

Dieses Kapitel fasst noch einmal die gewonnenen Ergebnisse und Erfahrungen zusammen.

9.1 Ergebnisse der Arbeit

Die vorliegende Arbeit dokumentiert die Architektur einer lernfähigen Software, die zur Steuerung eines mobilen Roboters eingesetzt wird. Die Adaptionsfähigkeit in Hinsicht auf das zu erlernende Ziel wurde dabei mit Hilfe eines genetischen Verfahrens realisiert.

Zur Ermittlung der genetischen Rahmenparameter (Mutations-, Kreuzungsraten, Generationen-, Populationsanzahl usw.) wurde im Vorfeld eine Simulatorsoftware erstellt, mit deren Hilfe anhand eines vereinfachten Problems, die benötigten Werte ermittelt werden konnten. Diese Vorgehensweise hat sich als sinnvoll herausgestellt, da die gesuchten Parameter schneller als mit der hardwarebasierten Variante ermittelt werden konnten. Mit Hilfe der gewonnenen Parameter und der einfachen Portabilität der Simulatorsoftware auf das Zielsystem, konnten die Versuchsreihen mit der Roboterplattform zeitlich optimiert werden. Der Simulator hat auch hinsichtlich der Formalisierung der Lernaufgabe (Fitnessfunktion) wichtige Ergebnisse hervorgebracht. Hier zeigte sich in den Versuchreihen, dass die Kodierung des gewünschten Ziels in eine Fitnessfunktion, kein trivialer Vorgang ist und mit besonderer Sorgfalt ausgeführt werden muss. Probleme, die mit dem zur Erstellung der Urpopulation benötigten Zufallsgenerators aufgetreten sind, konnten durch die schnelle Überprüfbarkeit des Algorithmus leichter überwunden werden, als es mit einer Testreihe auf dem Roboter möglich und sinnvoll gewesen wäre.

Die Portierung der programmierten Module auf das Robotersystem konnte wie geplant durchgeführt werden. Mit Hilfe verschiedener Versuchsreihen konnten die Sensorwerte klassifiziert und für die Verarbeitung im genetischen System bereit gestellt werden. Die Formalisierung des Ziels in Form der Fitnessfunktion bedurfte eines gewissen systematischen Vorwissens über die gestellte Aufgabe und somit auch ein Teilwissen über die Lösung. Hierbei ist die Aufgabe, dass zu erreichende Ziel in Form „eines“ Wertes abzubilden (dem Fitnesswert) die schwierigste Aufgabe gewesen. Das Arbeiten mit der Roboterplattform und den angeschlossenen Sensoren und Aktoren hat deutlich gemacht, dass die von den Sensoren gemessenen Werte sehr unzuverlässig sein können, sowie, dass die Ausführung von Aktionen nicht immer reproduzierbare Ergebnisse hervorbringen muss.

Mit Meßungenauigkeiten, wie auch Antriebsfehler in Form von durchdrehenden Rädern muss in der „realen Welt“ bei der Programmierung mobiler Roboter gerechnet werden. Es war zu beobachten, dass einige der Fehler scheinbar durch den genetischen Lernvorgang kompensiert wurden.
Die Roboterversuche zeigen, trotz der Hardwaremängel und den daraus resultierenden Abbrüchen während der Versuchsreihen, dass das vorgegebene komplexe Verhalten mit Hilfe des gewählten genetischen Algorithmus erlernbar ist.

9.2 Kritische Anmerkungen und Ausblick

Wie die Simulator- und Roboterversuche belegt haben, ist das genetische Verfahren für das Erlernen einer Choreographie zur Steuerung eines mobilen Roboters geeignet. Der Einsatz eines genetischen Verfahrens kann insbesondere bei komplexen Problemen nützlich sein, bei denen es z.B. eine große Anzahl von Stellgrößen gibt, kein kinematisches Modell des Antriebes vorhanden ist, oder andere zur Lösung benötigten Daten fehlen. Eine Programmierung von Hand ist in solchen Fällen sehr schwierig, wenn nicht sogar unmöglich. Der Suchraum, in dem sich eine geeignet Lösung des Problems befindet, kann in solchen Fällen so groß sein, dass mit konventionellen Suchmethoden keine Lösung in einer angemessenen Zeit gefunden werden kann. Vergleichbare Algorithmen, wie der Hillclimbing Algorithmus, können innerhalb des Lösungsraums in lokalen Maxima stecken bleiben und so evtl. bessere Lösungen nicht finden. Dieser Effekt kann bei den genetischen Algorithmen durch die Mutationsmethode abgemildert werden. Dennoch gibt es einige kritische Punkte zu beachten. Die Erstellung einer geeigneten Fitnessfunktion, also die Formalisierung des Problems, ist keine triviale Aufgabe. Hier ist es insbesondere nötig, dass eine Möglichkeit für das System besteht eine Selbstinspektion durchzuführen oder dass eine externe Prüfung stattfinden kann. Mit anderen Worten, kann über die an das System angeschlossenen oder extern vorhandenen Sensoren (Kameras etc.) ermittelt werden, ob die ausgeführten Aktionen als gut oder schlecht zu bewerten sind? Hier kann z.B. die Wegstreckenmessung, oder wie im Fall des verwendeten Roboters die Entfernungsmessung zum Tor oder zum Infrarotball problematisch sein. Diese war mit den verfügbaren Sensoren nicht möglich. Zu Bedenken ist auch, dass die Modellbildung, also das Design des Genoms und die Erstellung des Interpreters, eine Menge programmiertechnische Vorarbeit verlangt. Ebenso wie für die Fitnessfunktion, muss ein Gefühl für die benötigten Rahmenparameter entwickelt werden. Dies ist in theoretischer Vorarbeit nur begrenzt möglich und bedarf, wie sich herausgestellt hat, praktischer Arbeit mit den Algorithmen. Hier trifft das Zitat von Wolfgang Banzhaf zu:

> *“The art of choosing an appropriate representation and an appropriate set of operators is often a matter of experience and intuition, and can only be mastered by working with the algorithms.”*
>
> *(Quelle: [BANZHAF 1998])*

Für einige Probleme der mobilen Robotik werden die genetische Verfahren nur sehr schwer einsetzbar sein. Dies ist insbesondere bei Anwendungen der Fall, wo ein Versagen der Steuerung den eingesetzten Roboter zerstören würde und ein automatisierter Mechanismus zum Selbstschutz[10] der Geräte nur schwer realisierbar ist (z.B. bei Flugzeugen). Der in den Versuchen eingesetzte Roboter wurde durch ein einfach zu implementierendes Verhalten, das übergeordnet zum genetischen Prozess agierte, geschützt. Hier konnte eine Bumperabfrage bei Versagen einer Regelbasis den Roboter vor Schaden bewahren.

Die von Michael Manger [MANGER 2004] entwickelte Prototypplattform hat, wie die Versuchsreihen gezeigt haben, mit einigen Mängeln zu kämpfen. Bei der verwendeten Konstruktion waren der Antrieb, sowie der drehbare Ballsensor die Hauptprobleme. Diese Probleme waren die Motivation eine neue verbesserte Plattform zu erstellen. Dieser Herausforderung hat sich mein Kommilitone Michael Ziener [ZIENER 2005] in seiner Diplomarbeit gestellt. Entstanden ist eine flexible Experimentierplattform mit folgenden verbesserten Eigenschaften:

- Robuste Aluminiumkonstruktion
- Stabile Antriebseinheit mit verbesserter Leistung
- Rundum stationäre Abstandssensoren
- Stabile Bumperleisten
- Beweglich Farbkamera zur Objekterkennung [BRANDT 2005]

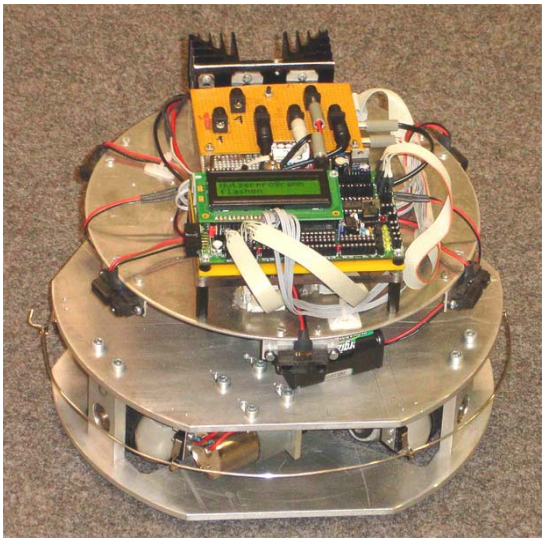


Abbildung 9.1: Verbesserte Experimentelle Roboterplattform
(Quelle: Michael Ziener [ZIENER 2005])

[10] Um z.B. Modellhubschraubern das Fliegen beizubringen, müsste ein übergeordneter Mechanismus geschaffen werden, der das Gerät bei Ausführung schlechter Genome, vor der Zerstörung schützt. Wenn die Steuerung des Gerätes dabei sehr komplex ist, wie im Falle eines Hubschraubers, wäre ein programmierter automatisierter Schutz nur sehr schwierig zu realisieren und müsste über einen menschlichen Bediener, der die Steuerung im Bedarfsfall übernimmt, gesichert werden.

Die verbesserte Robustheit würde es erlauben den genetischen Algorithmus länger als es mit der alten Plattform möglich war laufen zu lassen. Hier wäre insbesondere interessant, ob eine verlängerte Laufzeit die Qualität des erzeugten Verhaltens noch verbessern könnte. Die neue Plattform bietet durch die installierte Kamera weitere neue Möglichkeiten. Hier wäre die Verarbeitung der Kamerabilder mit Hilfe eines genetischen Algorithmus ein weiterer interessanter Aspekt. Folgende Lernaufgaben könnten evtl. vorgenommen werden:

- Abstandsmessung von Objekten in Relation zum Roboter
- Objektmanipulation (z.B. Ball in ein Tor schießen)
- usw.

Zusammenfassend kann gesagt werden, dass die vorgestellten genetischen Verfahren in der Lage sind erstaunliche Adaptionsleistungen hervorzubringen. Die zur Genetischen Programmierung von John Koza [KOZA 1992] weiterentwickelten evolutionären Algorithmen bieten sogar noch umfangreichere Möglichkeiten in Bezug auf die Komplexität der zu lösenden Aufgaben.

Die vorgestellten Methoden scheinen einer möglichen Antwort, auf die von Arthur Samuel gestellten Frage, näher zu kommen.

“How can computers learn to solve problems without being explicitly programmed? In other words: How can computers be made to do what is needed to be done, without being told exactly how to do it?"

(Arthur Samuel, 1950s)

10 LITERATURVERZEICHNIS

1. **AKSEN 2005**: Boersch, Ingo: Nutzerhandbuch AKSEN-Board, FH Brandenburg, Fachbereich Informatik und Medien, 2005

2. **BANZHAF 1998**: Banzhaf, Wolfgang: Genetic Programming, Springer, 1998, ISBN – 3-540-6436-05

3. **BOERSCH**: Boersch, Ingo: FH Brandenburg – Labor für künstliche Intelligenz, Online URL: http://ots.fh-brandenburg.de/index.php/, Zugriffsdatum Januar 2005

4. **BRANDT 2005**: Brandt, Lars: Entwicklung eines Objekttrackers für Embedded Systems zur Steuerung mobiler Roboter , Diplomarbeit, HAW Hamburg, Fachbereich Informatik, 2005

5. **BROOKS 2005**: Brooks, Rodney: MIT Artificial Intelligence Lab., Online URL: http://www.ai.mit.edu/, Zugriffsdatum Januar 2005

6. **CALLAN 2003**: Callan, Robert: Neuronale Netze im Klartext, Pearson Studium, 2003, ISBN – 3-8273-7071-X

7. **CAWSEY 2003**: Cawsey, Alison: Künstliche Intelligenz im Klartext, Pearson Studium, 2003, ISBN – 3-8273-7068-X

8. **EIBEN 2003**: Eiben, A.E.; Smith, J.E.: Introduction to Evolutionary Computing Series: Natural Computing Series, Springer, 2003, ISBN – 3-540-40184-9

9. **FRAUNHOFER AIS 2005**: Fraunhofer Gesellschaft: Fraunhofer Institut für Autonome Intelligente Systeme, Online URL: http://www.ais.fraunhofer.de/, Zugriffsdatum Januar 2005

10. **GMD 1998**: GMD; Fraunhofer Gesellschaft: Artikel über Frank Kirchner, Roboter von der Tafelrunde, Online URL: http://www.gmd.de/pointer/2-98/arthur.html, Zugriffsdatum Januar 2005

11. **GÖRZ 2003**: Görz, Günther; Rollinger, Claus-Rainer; Schneeberger, Josef: Handbuch der künstlichen Intelligenz, Oldenbourg, 2003, ISBN – 3-486-27212-8

12. **HONDA D 2005**: Honda Robots Deutschland: Honda Asimo, Online URL: http://www.honda-robots.com/index_ori.html, Zugriffsdatum Januar 2005

13. **HONDA WORLD 2005**: Honda Robots World: Honda Asimo, Online URL: http://world.honda.com/ASIMO/, Zugriffsdatum Januar 2005

14. **JEDITOR**: Pestov, Slava; Gellene, John: Jedit programmers texteditor, Online URL: http://www.jedit.org/, Zugriffsdatum Dezember 2004

15. **JONES 1996**: Jones, Joseph L.; Flynn, Anita M.: Mobile Roboter – Von der Idee zur Implementierung, Addison-Wesley, 1996, ISBN – 3-89319-855-5

5 **KOZA 1992**: Koza, John R.: Genetic Programming – On the programming of computers by means of natural selection, The MIT Press, 1992, ISBN – 0-262-11170-5

16. **KOZA ONLINE 2004**: Koza, John R.: Genetic Programming Inc. Research Group, Online URL: http://www.genetic-programming.com/, Zugriffsdatum Dezember 2004

17. **MANGER 2004**: Manger, Michael: Design und Realisierung einer experimentellen Plattform für Roboterfußball, Diplomarbeit, HAW Hamburg, Fachbereich Informatik, 2004

18. **MENZEL 2000**: Menzel, Peter; D'Aluisio, Faith: Robo sapiens, The MIT Press, 2000, ISBN – 0-262-13382-2

19. **PLYJOJUMP**: Plyojump, Online URL: http://www.plyojump.com/qrio.html/, Zugriffsdatum August 2005

20. **PORSCHKE 2002**: Porschke, Jan Philip: Genetische Programmierung von verhaltensbasierten Agenten, Diplomarbeit, HAW Hamburg, Fachbereich Informatik, 2002

21. **PRINZ 2002**: Prinz, Peter; Kirch-Prinz, Ulla: C kurz und gut, O´Reilly, 2002, ISBN – 3-89721-238-2

22. **RECHENBERG 2002**: Rechenberg, Peter; Pomberger, Gustav: Informatik Handbuch, Hanser Verlag München, 2002, ISBN – 3-446-21842-4

23. **RCUBE 2005**: Boersch, I: Getting started with RVISION, FH Brandenburg, Fachbereich Informatik und Medien, 2005

24. **SCHÖNEBURG 1993**: Schöneburg, Eberhard; Heinzmann, Frank; Feddersen, Sven: Genetische Algorithmen und Evolutionsstrategien, Addison-Wesley, 1993, ISBN – 3-89319-493-2

25. **SDCC**: Vigor, Kevin; Knol, Johan; Dattalo, Scott: SDCC - Small Device C Compiler, Online URL: http:/sdcc.sourceforge.net/, Zugriffsdatum Dezember 2004

26. **SONY AIBO**: Sony AIBO, Online URL: http://www.eu.aibo.com/, Zugriffsdatum August 2005

27. **SONY QRIO**: Sony QRIO, Online URL: http://www.sony.net/SonyInfo/QRIO/, Zugriffsdatum August 2005

28. **UNI DORTMUND**: Uni Dortmund: Universität Dortmund – Lehrstuhl für künstliche Intelligenz, Online URL: http://www-ai.cs.uni-dortmund.de/index.html/, Zugriffsdatum April 2005

29. **WALL MIT 1996**: Wall, Matthew: GAlib: A C++ Library of Genetic Algorithm Components, Online URL: http://lancet.mit.edu/ga/, Zugriffsdatum November 2004

30. **WIKI**: Wikipedia, Online URL: http://de.wikipedia.org/wiki/Hauptseite/, Zugriffsdatum April 2005

31. **WILTRONICS 2005**: Wiltronics Research Pty Ltd: Wiltronics Robots, Online URL: http://www.wiltronics.com.au/, Zugriffsdatum Januar 2005

32. **ZIENER 2005**: Ziener, Michael: Konstruktion eines programmierbaren, omnidirektionalen Roboters , Diplomarbeit, HAW Hamburg, Fachbereich Informatik, 2005

11 ABBILDUNGSVERZEICHNIS

12 ANHÄNGE

Anhang A

Im Folgenden sind exemplarische, von der Simulatorsoftware generierte Ausgaben, zu sehen:

Parameterübersicht Genetischer Prozess:

```
Diplomarbeit 2005 - Timo Storjohann

Genetische Lernverfahren - LineFollow Simulator


Starte Genetischen Prozess mit folgenden Parametern:

Populationsgroesse:          15
Generationen:                100
Lebenszyklen pro Genom:      15
Mutationswahrscheinlichkeit: 10%
Kreuzungswahrscheinlichkeit: 80%
Direkte Emigration:          10%
Elitarismus Option:          1
Advanced Fitness:            1
```

Urpopulation:

```
##########################################################
######    Populationsgenerator wird gestartet    ######
##########################################################


Individuum 1 von 15 wird erzeugt

Sensorkombination Nr: 0 -- 0 = Aktion 5
Sensorkombination Nr: 1 -- 1 = Aktion 7
Sensorkombination Nr: 2 -- 2 = Aktion 4
Sensorkombination Nr: 3 -- 3 = Aktion 3
Sensorkombination Nr: 4 -- 4 = Aktion 7
Sensorkombination Nr: 5 -- 5 = Aktion 3
Sensorkombination Nr: 6 -- 6 = Aktion 2
Sensorkombination Nr: 7 -- 7 = Aktion 3


Individuum 2 von 15 wird erzeugt

Sensorkombination Nr: 0 -- 0 = Aktion 3
Sensorkombination Nr: 1 -- 1 = Aktion 7
Sensorkombination Nr: 2 -- 2 = Aktion 1
Sensorkombination Nr: 3 -- 3 = Aktion 1
```

```
...

Individuum 15 von 15 wird erzeugt

Sensorkombination Nr: 0 -- 0 = Aktion 7
Sensorkombination Nr: 1 -- 1 = Aktion 7
Sensorkombination Nr: 2 -- 2 = Aktion 3
Sensorkombination Nr: 3 -- 3 = Aktion 0
Sensorkombination Nr: 4 -- 4 = Aktion 5
Sensorkombination Nr: 5 -- 5 = Aktion 7
Sensorkombination Nr: 6 -- 6 = Aktion 5
Sensorkombination Nr: 7 -- 7 = Aktion 6


##### neue Population mit 15 Genomen generiert #####
```

„Best of Fitness“ für jede Generation:

```
Verlauf der besten Fitness jeder Generation:

Generation Nr: 1 - Fitness: 5
Generation Nr: 2 - Fitness: 16
Generation Nr: 3 - Fitness: 17
Generation Nr: 4 - Fitness: 20
Generation Nr: 5 - Fitness: 34
Generation Nr: 6 - Fitness: 123
Generation Nr: 7 - Fitness: 130
Generation Nr: 8 - Fitness: 131
Generation Nr: 9 - Fitness: 131
Generation Nr: 10 - Fitness: 135
Generation Nr: 11 - Fitness: 145
Generation Nr: 12 - Fitness: 148
Generation Nr: 13 - Fitness: 149
Generation Nr: 14 - Fitness: 152
Generation Nr: 15 - Fitness: 163
Generation Nr: 16 - Fitness: 163
Generation Nr: 17 - Fitness: 163

...
```

Fitnessverlauf der gesamten Evolution:

```
...

Fitness Nr:20 = 33
Fitness Nr:19 = 46
Fitness Nr:18 = 42
Fitness Nr:17 = 163
Fitness Nr:16 = 0
Fitness Nr:15 = 14
Fitness Nr:14 = 156
```

```
Fitness Nr:13 = 26
Fitness Nr:12 = 20
Fitness Nr:11 = 0
Fitness Nr:10 = 0
Fitness Nr:9 = 31
Fitness Nr:8 = 29
Fitness Nr:7 = 33
Fitness Nr:6 = 52
Fitness Nr:5 = 14
Fitness Nr:4 = 22
Fitness Nr:3 = 0
Fitness Nr:2 = 0
Fitness Nr:1 = 42

Beste Fitness:170

...
```

Verhalten der Regelbasen:

```
...

REGELBASIS INTERPRETIEREN:

FAHRE SCHRAEG RECHTS VORWAERTS

FAHRE RUECKWERTS

FAHRE RECHTS

FAHRE SCHRAEG LINKS RUECKWERTS

FAHRE SCHRAEG RECHTS RUECKWERTS

-- Bumper ausgelöst -- Regelbasis unbrauchbar -- Abbruch

...
```

Ergebnis:

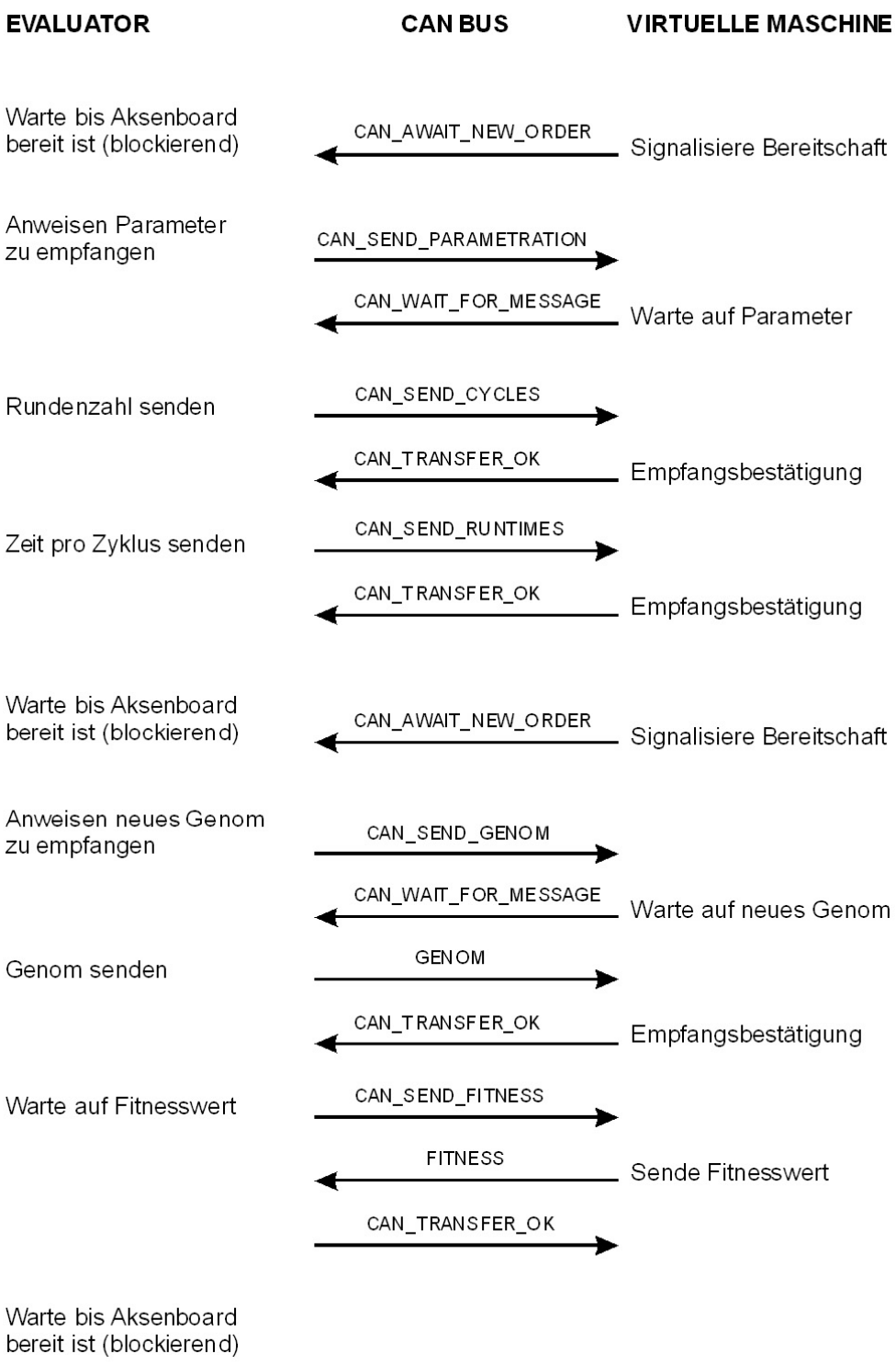
```
...

Beste Fitness:106


Gen Nr:0   Sensor:0  Aktion:2
Gen Nr:1   Sensor:1  Aktion:3
Gen Nr:2   Sensor:2  Aktion:0
Gen Nr:3   Sensor:3  Aktion:7
Gen Nr:4   Sensor:4  Aktion:4
Gen Nr:5   Sensor:5  Aktion:2
Gen Nr:6   Sensor:6  Aktion:6
Gen Nr:7   Sensor:7  Aktion:3
```

```
Gefundenes Genom testen

0000000000
0000000000
0001110000
0010001000
0010000100
00=0001000
0010001000
0001110000
0000000000
0000000000


Roboter ausgerichtet nach: Nord


REGELBASIS INTERPRETIEREN:
CNr:0 SensorSymbol: 2 - Aktion:0
FAHRE GERADEAUS


0000000000
0000000000
0001110000
0010001000
00=0000100
0010001000
0010001000
0001110000
0000000000
0000000000


Roboter ausgerichtet nach: Nord

CNr:1 SensorSymbol: 2 - Aktion:0
FAHRE GERADEAUS


0000000000
0000000000
0001110000
00=0001000
0010000100
0010001000
0010001000
0001110000
0000000000
0000000000


Roboter ausgerichtet nach: Nord

CNr:2 SensorSymbol: 4 - Aktion:4
```

```
FAHRE RUECKWERTS


0000000000
0000000000
0001110000
0010001000
00=0000100
0010001000
0010001000
0001110000
0000000000
0000000000


Roboter ausgerichtet nach: Sued

.......

.......

Programm Ende.
```

Anhang B

Im folgenden Chart ist das implementierte Handshakeprotokoll zur Übertragung des Genommaterials und der Fitnesswerte über den integrierten CAN-Bus zu sehen.



Versicherung über Selbstständigkeit

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.

Hamburg, den 23.08.2005

Timo Storjohann