Evolution von Laufrobotersteuerungen mit Genetischer Programmierung

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik
von

Jens Ziegler

Dortmund,

Februar 2003

Tag der mündlichen Prüfung:
18. Juni 2003

Dekan:
Prof. Dr. Bernhard Steffen

Gutachter:
Prof. Dr. rer.-nat. Wolfgang Banzhaf
Prof. Dr. Heinrich Müller

für Petra und Clemens

Diese Arbeit entstand mit freundlicher Unterstützung der Deutschen Forschungsgemeinschaft (DFG) im Rahmen meiner Tätigkeit im Schwerpunktprogramm „Autonomes Laufen“. Für das entgegengebrachte Vertrauen und die fachliche Unterstützung gilt mein Dank Herrn Prof. Dr. Wolfgang Banzhaf. Für die angeregten Diskussionen und fachlichen Kommentare gilt mein Dank meinen Kollegen Jens Niehaus und Wolfgang Kantschik.

Für die gute Zusammenarbeit, die zu den Ergebnissen in Abschnitt 8.3.2 und 8.3.3 geführt hat, möchte ich mich ganz besonders bei Jens Busch bedanken.

Ausserdem danke ich meinen studentischen Hilfskräften Christian Düntgen, Peter Fricke, Walter Nowak, Philipp Limbourg, Pattrick Matters und Jan Barnholt sowie der Projektgruppe 368 für ihre technische Unterstützung.

Mein Dank gilt ebenfalls Peter Nordin und Krister Wolff von der Chalmers University of Technology in Göteborg, Schweden, denn ohne die Zusammenarbeit mit ihnen hätte ZORC nicht das Licht der Welt erblickt.

Inhaltsverzeichnis

Einleitung

Die Oberfläche der Erde besteht zu ca. 30% aus Landmassen, von der nur etwa die Hälfte für konventionelle Fahrzeuge mit Rad- oder Kettenantrieb erreichbar ist. Ein weit größerer Anteil jedoch ist für Lebewesen mit Beinen zugänglich. Ein Grund dafür, dass Beine eine höhere Beweglichkeit als Räder oder Ketten in unwegsamen Regionen erlauben, besteht darin, dass die Laufbewegung nur wenige, vereinzelte Stützpunkte für die Beine benötigt, bei Rädern und Ketten hingegen ununterbrochen Kontakt zum Boden gewährleistet werden muss.

Trotz der erkennbaren hervorragenden Beweglichkeit jedes Menschen und der fortschreitenden wissenschaftlichen Untersuchung des Laufens bei anderen Lebewesen sind die dem Laufen zu Grunde liegenden Prinzipien aber erst teilweise offengelegt. Eine Möglichkeit diese Prinzipien näher zu untersuchen ist daher, Laufmaschinen zu konstruieren und ihre Eigenschaften zu analysieren, wie Raibert in [139] vorschlägt.

> „One way to learn more about plausible mechanisms for animal locomotion is to build machines that locomote using legs“

Die ersten Laufroboter in der Mitte des 20. Jahrhunderts waren tonnenschwere Konstruktionen, die teuer, unhandlich und langsam waren. Begünstigt durch technologische Fortschritte in der Mikroelektronik, Energietechnik, der Antriebstechnik und der Miniaturisierung und getrieben vom Wunsch nach dem Verständnis einer Fortbewegungsart, die vorherrschend bei der bodengebundenen Bewegung im Tierreich ist, wurde dennoch eine Vielzahl von Laufrobotern in den letzten Jahrzehnten konstruiert. Vorbilder in der Natur haben zum Teil Pate gestanden für den Entwurf von Laufrobotern, um von der im Verlauf der Evolution optimierten Konstruktion von Beinen, dem Zusammenspiel von Sensorik und Aktorik und der Steuerung von Gehbewegungen profitieren zu können. Diese biomimetische Robotik hat auch zur Entwicklung von humanoiden Robotern geführt, Robotern, deren Beweglichkeit und Abmessungen den menschlichen Proportionen nachempfunden sind.

Durch den stark interdisziplinären Charakter des Forschungsgebiets - Ingenieurwissenschaften z.B. für Konstruktion, Antrieb, Sensorik und Kommunikation, die Informatik z.B. für Datenstrukturen, Datenerfassung, Informationsverarbeitung und Planen, Lebenswissenschaften beispielsweise für Modelle der biologisch-chemischen Informationsverarbeitung, Kognition und Intelligenz - sind auch die untersuchten Fragestellungen ein Spiegel dieser Vielfalt. Die Suche nach leichten und gleichzeitig stabilen Materialien für die tragenden Konstruktionselemente, die Konstruktion von immer leistungsstärkeren und kompakteren Motoren und Getrieben, Untersuchungen an pneumatischen Antrieben, deren Funktionsweise den Muskeln nachempfunden ist, die Umsetzung von biomechanischen Erkenntnissen über die Geometrie und Anordnung von Beinen oder der Transfer von biologischen Kontrollmechanismen auf Laufroboter sind nur ein kleiner Ausschnitt der Themen, die zur Zeit aktuell sind.

Die Forschung an Laufrobotern dient gleichzeitig auch kommerziellen Interessen, wie beispielsweise der Erfolg des vierbeinigen Aibo-Roboters der japanischen Firma Sony zeigt. Ankündigungen von namhaften - und meist japanischen - Unternehmen, Laufroboter für unterschiedliche Zwecke in Kürze in Serie fertigen und vermarkten zu wollen, zeigen das wirtschaftliche Potenzial, das in dieser Art von Robotern vermutet wird.

Aus welchen Gebieten auch die Problemstellungen entstammen, die zu Entwurf und Konstruktion eines Laufroboters führen, eines muss dieser Roboter immer beherrschen: die stabile Fortbewegung mit Hilfe seiner Beine. Gerade hierin liegt eine der Hauptschwierigkeiten dieses Gebietes, denn die Implementierung eines Steuerungsprogramms für Laufroboter ist aufgrund der engen Verzahnung von Struktur und Funktion der einzelnen Beine und Glieder eines Laufroboters eine sehr komplexe Aufgabe. Hinzu kommt, dass bedingt durch die unterschiedlichen Roboterarchitekturen, die sich in der Anzahl der Beine, der Anzahl der Gelenke pro Bein, in den geometrischen Abmessungen und Massen und vielen weiteren Details unterscheiden, keine kanonische Form für den Entwurf einer Steuerung für Laufroboter existiert. Eine Möglichkeit zur Umgehung dieses Problems besteht darin, die Kontrollprogramme nicht manuell zu implementieren, sondern Verfahren zu entwickeln, die automatisch für beliebige Roboterarchtitekturen optimierte Laufprogramme generieren.

Evolutionäre Algorithmen sind ein Verfahren zur Optimierung, das sich an den Prinzipien der natürlichen Evolution orientiert. Eine Besonderheit dieser Klasse von Optimierverfahren ist es, dass für den Prozess der Optimierung keine Information über die Struktur des konkreten Problems benötigt wird. Der Raum der Problemvariablen wird durchsucht mit Hilfe einer Population von Suchpunkten, die jeweils spezielle Lösungen der zu optimierenden Funktion repräsentieren. Die Darstellung der Problemparameter erfolgt dabei meist über eine spezielle Kodierung. Wiederholte Veränderungen an der Zusammensetzung der Population mit speziellen Variationsoperatoren, wobei nur die besseren Lösungen gemäß einer definierten Gütefunktion beibehalten werden, erzielen im Verlauf der Evolution immer bessere Resultate. Die Gütefunktion ist hierbei nicht nur auf mathematische Kriterien beschränkt, sie kann auch subjektive Eindrücke eines menschlichen Betrachters einschließen.

Für den Fall der Evolution von Laufrobotersteuerungen wird die Güte der Lösungen bestimmt, indem die in den Individuen kodierte Laufrobotersteuerung ausgeführt und die Qualität des Programms ermittelt wird. Evolutionäre Algorithmen sind bekannt für ihre Kreativität bei komplexen Problemstellungen und haben sich vielfach bewährt, indem sie Lösungen generiert haben, deren Leistungsfähigkeit oft den manuell erstellten Lösungsansätzen gleichkam oder sie sogar übetraf. In Abwandlung des oben aufgeführten Zitats könnte die Lösung des Problems der Implementierung von Kontrollprogrammen für beliebige Laufroboter also lauten:

> „One way to learn more about plausible mechanisms for legged locomotion is to *evolve* control programs that locomote legged robots“

Und genau dies ist das Thema dieser Arbeit: Die Evolution von Laufrobotersteuerungen mit Genetischer Programmierung wird anhand unterschiedlicher Roboterarchitekturen sowohl in der Simulation als auch mit realen Robotern untersucht. Bedingt durch die lang andauernden Auswertungen von Laufprogrammen in Simulation und Realität werden die Methoden der Evolution - Evaluation, Selektion und Variation - zusätzlich einer gründlichen Analyse unterzogen, um den evolutionären Prozess zur automatischen Generierung von optimierten Laufrobotersteuerungen möglichst zu beschleunigen.

Im ersten Teil der Arbeit werden Grundlagen für die Evolution von Laufroboterprogrammen erläutert. Soweit notwendig, werden Basisbegriffe eingeführt, die im weiteren Verlaufe der Arbeit verwendet werden. An gegebenen Stellen wird auf weiterführende Literatur verwiesen. In Kapitel 1 wird die Thematik der autonomen mobilen Roboter eingehend betrachtet, wobei das Teilgebiet der Laufroboter und die speziell hierfür relevanten Problemstellungen dargestellt werden. Das zweite Kapitel beschäftigt sich mit der Genetischen Programmierung, einem Spezialfall der Evolutionären Algorithmen. Die für die Arbeit benötigten Begriffe des evolutionären Rechnens werden vereinbart, sowie die grundlegenden Mechanismen evolutionärer Algorithmen aufgezeigt. Hierbei wird detailliert auf die besonderen Eigenschaften der Genetischen Programmierung eingegangen. Im dritten Kapitel werden die mathematischen Grundlagen der Dynamiksimulation von Laufrobotern - Kinematik, Dynamik, Kollisionserkennung - dargestellt.

Der zweite Teil der Arbeit führt in die Thematik der Evolution von Laufrobotersteuerungen ein. In Kapitel 4 werden detailliert die generelle Zielsetzung und die verwendeten Methoden bei der Evolution von Roboter-

steuerungen beschrieben. Es wird hierbei besonders die Repräsentation von Kontrollstrukturen autonomer Roboter in Evolutionären Algorithmen und die Umsetzung auf reale oder simulierte Roboter diskutiert. Im folgenden Kapitel werden die Vielfalt der Fortbewegungsarten und die Konsequenzen des Formenreichtums heutiger autonomer mobiler Laufroboter präsentiert. Im anschließenden Kapitel, Kapitel 6, wird die Methodik für die Evolution von Laufroboterprogrammen mit Genetischer Programmierung eingeführt. Die Notwendigkeit der Evolution mit simulierten und realen Robotern, sowie die Möglichkeit, beide Alternativen gleichzeitig zu verwenden, wird diskutiert.

In Teil III der Arbeit werden die Experimente zur Evolution von Laufprogrammen präsentiert. In Kapitel 7 wird das Laufzeitverhalten der Evolution von Laufroboterprogrammen untersucht. Gegenstand der Analyse sind dort die Möglichkeiten zur Parallelisierung des Algorithmus, zur Einsparung von Rechenzeit bei der Auswertung von Laufprogrammen, zur Optimierung von Parametern des evolutionären Prozesses und zur Beschleunigung der Evolution mit Hilfe von Modellen der Fitnessfunktion bei der Genetischen Programmierung. Kapitel 8 zeigt die erfolgreiche Evolution von Laufprogrammen für eine Vielfalt von Roboterformen in der Simulation auf. Die Angabe eines Gütekriteriums ist essenziell für die Evolution von Laufmustern und wird eingehend diskutiert. Dargestellt werden die Auswirkungen verschiedener Varianten der Fitnessfunktion auf die resultierenden Bewegungsmuster der Roboter. Im Kapitel 9 werden für einen sowohl real als auch als Computermodell existierenden humanoiden Roboter Laufprogramme evolviert. Hierbei wird das in Kapitel 6 eingeführte Konzept einer zweiphasigen Evolution in Simulation und Realität erfolgreich angewendet. Die Ergebnisse von Evolutionsläufen mit vierbeinigen Laufrobotern werden in Kapitel 10 vorgestellt. Die Resultate der Experimente unter Einbeziehung der interaktiven Evolution und der Modellierung der Fitnessfunktion zeigen eine deutliche Verbesserung gegenüber dem Standardalgorithmus. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Entwicklungen auf dem Gebiet der Evolution von Laufrobotersteuerungen.

Teil I

Grundlagen

Kapitel 1

Autonome Roboter

Ursprünglich wurden Roboter[1] nur im Bereich der Automatisierungstechnik eingesetzt als multifunktionelle, vielfältig programmierbare flexible Produktionseinheiten (eine detaillierte Einführung in die Robotik findet sich z.B. in [126]). Die Grundform eines Industrieroboters ist ein Manipulatorroboter, dessen mit mehreren Gelenken und einem Endeffektor ausgerüsteter Arm für das Bewegen, Fügen, Bearbeiten, Prüfen von Werkstücken u.Ä. verwendet wird. Es existieren mittlerweile auch andere Formen von Robotern. Serviceroboter beispielsweise übernehmen Aufgaben, die nicht mit der Produktion in Zusammenhang stehen, sondern Kontroll- bzw. Informationsaufgaben darstellen. Ist ein Roboter nicht nur ortsfest, sondern im Raum frei beweglich, spricht man von mobilen Robotern. Die Möglichkeit zur kontrollierten Bewegung ist eine Eigenschaft, die bei nahezu allen höher entwickelten Lebewesen beobachtet werden kann und die für die vielseitige Verwendbarkeit von Robotern eine herausragende Rolle spielt.

Mobile Roboter existieren für eine Vielfalt von unterschiedliche Domänen: so gibt es die mittlerweile üblichen bodengebundenen Roboter, aber auch Unterwasserroboter, Flugroboter, Roboter für Planetenmissionen, Kanalroboter oder Kletterroboter. Die Vielfalt der Anwendungen geht mit einer Vielfalt an Bauformen einher. Die bei weitem wichtigste Anwendung ist allerdings die Bewegung in Umgebungen, die für die speziellen Bedürfnisse des Menschen zugeschnitten sind. Mobile Roboter sind ein wichtiges Teilgebiet der Robotik. Aspekte anderer Teilgebiete, beispielsweise aus dem Gebiet der Industrieroboter oder der Antriebstechnik, sind auch für mobile Roboter von Bedeutung, andere Problemstellungen hingegen, die meist aus der Fähigkeit zur Bewegung und Manipulation der Umwelt resultieren, sind spezifisch für mobile Roboter.

Autonome mobile Roboter sind seit Jahrzehnten Gegenstand der wissenschaftlichen Arbeit. Die ersten autonomen Vehikel stammen sogar aus dem 19. Jahrhundert [55]. Die Faszination, die von einer Maschine ausgeht, die Eigenschaften aufweist, wie sie normalerweise nur intelligenten Lebensformen zugeschrieben werden, insbesonders die Fähigkeit zur kontrollierten Bewegung und zu zweckmäßigen Reaktionen auf veränderliche Umweltbedingungen, ist ungebrochen und spiegelt sich unter anderem in der Vielfalt der existierenden Roboter wider.

1.1 Autonomie

Autonome Roboter sind Systeme, die ihre eigenen Regeln und Strategien entwickeln, gemäß denen ihr Verhalten bestimmt wird. Sie stehen damit im Gegensatz zu automatischen Systemen, die zwar selbstregulierend, aber extern bestimmt sind. Autonome Roboter können deshalb in ungewissen Umgebungen

[1] Der Begriff „Roboter“ stammt aus dem Tschechischen (*robota* = arbeiten) und wurde 1921 von Karel Capek im Bühnenstück „Rossums Universalroboter“ geprägt.

operieren, die Fähigkeit zur Selbstkontrolle resultiert dabei aus der Möglichkeit, die eigenen Verhaltensweisen dynamisch zu verändern und anzupassen [163]. Der Entwicklung von Robotern mit einer solchen Kontrollautonomie wird eine große Bedeutung zugemessen, denn durch ihre Fähigkeit, flexibel und anpassungsfähig auf neue Situationen und Umweltbedingungen zu reagieren, können sie in Bereichen zum Einsatz kommen, in denen automatische Systeme versagen und der Mensch wegen seiner geringen körperlichen Belastbarkeit nicht einsatzfähig ist. Ein Beispiel sind hier die bereits eingesetzten Roboter für die Beseitigung von Landminen [55].

Das Gebiet der autonomen mobilen Roboter ist sehr interdisziplinär aufgrund der auftretenden verschiedenen Problemstellungen. Fragestellungen nach der Energieversorgung (Energieautonomie) der Roboter, der Kommunikation, der Bahnplanung, der Sensorik und Aktorik werden durch Forschergruppen aus den Ingenieurwissenschaften bearbeitet. Die Informatik spielt eine besondere Rolle bei der Steuerung der Roboter. Methoden der Agentensysteme, der Künstlichen Intelligenz (KI) und Computational Intelligence (CI) werden für die Implementierung von Kontrollprogrammen umgesetzt. Zur kontrollierten Bewegung der Roboter werden Methoden der symbolischen Wissensrepräsentation, der Modellierung der Umwelt sowie zur Handlungsplanung verwendet. Auch Methoden des maschinellen Lernens werden eingesetzt, um die Kontrollsoftware autonomer mobiler Roboter möglichst flexibel auf Veränderungen der Umwelt reagieren zu lassen. Verhaltensbasierte Ansätze, die ihren Ursprung in Erkenntnissen über neuro-biologische Reiz-Reaktionsmechanismen und biochemische Informationsverarbeitungsprozesse in Lebewesen haben, zeigen eine Verbindung zur Biologie, Biochemie und Neurologie auf.

Wenn statt einem mehrere Roboter gleichzeitig verwendet werden, entstehen Bezüge zu weiteren Forschungsgebieten: verteilte oder kollektive künstliche Intelligenz, Multi-Agenten-Systeme sind hier involvierte Gebiete. Kommunikation und Kooperation in einer Gruppe von mehreren Robotern können auch unter dem Blickwinkel der sozialen Interaktion von Agenten betrachtet werden, so dass Forschungsgebiete wie die Soziologie oder Linguistik Beiträge zum Gebiet der autonomen Roboter leisten, die auf den ersten Blick weit davon entfernt scheinen.

Die im Moment bedeutendste Anwendung autonomer mobiler Roboter ist allerdings der Bereich der Unterhaltungselektronik. Der weltumspannende Absatzmarkt, Umsätze in Millionenhöhe bereits mit Kleinserien von Prototypen und die stetig steigende Nachfrage nach High-Tech-Unterhaltungselektronik mit spielerischen Elementen haben den in Forschungseinrichtungen anvisierten „klassischen“ Anwendungsbereich autonomer mobiler Roboter in den Hintergrund treten lassen. Die dortigen Anforderungen an autonome mobile Roboter müssen für Unterhaltungsroboter erweitert bzw. modifiziert werden, um die Wirtschaftlichkeit der Herstellung zu gewährleisten. Die Forschung auf dem Gebiet in firmeninternen Forschungs- und Entwicklungsabteilungen (wie beispielsweise bei Honda oder Sony) besteht aus diesem Grund nicht nur darin, Roboter möglichst autonom und möglichst mobil zu gestalten, sondern auch und vor allem darin, dieses Ziel möglichst kostengünstig zu erreichen.

1.2 Laufroboter

Ein spezielles Gebiet der autonomen mobilen Roboter stellen die Laufroboter dar. Als Laufroboter werden mobile bodengebundene Roboter bezeichnet, die zur Fortbewegung keine Räder, Kettentriebe oder ähnliche rotierende Mechanismen verwenden.

Für die intensive Forschung auf dem Gebiet der Laufroboter in den letzten Jahren gibt es zwei bedeutende Gründe. Auf der einen Seite sind Laufroboter konventionellen bodengebundenen mobilen Robotern bei besonders unebenem Terrain theoretisch überlegen (sowohl in natürlichen als auch in künstlichen, für Menschen entwickelten Umgebungen wie beispielsweise Treppenhäusern). Auf der anderen Seite ist Laufen eine seit Jahrmillionen in der Natur vielfach verwendete Fortbewegungsart, deren Untersuchung dazu beitragen soll, anspruchsvolle Bewegungsmuster auf Maschinen übertragen zu können. Zielsetzung ist hier einerseits,

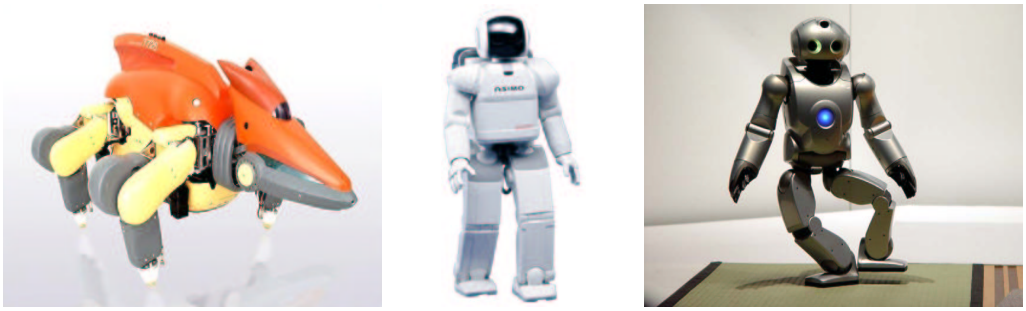


Abbildung 1.1: Im Jahr 2002 vorgestellte Laufroboter namhafter japanischer Firmen. **Links:** Autonomer Roboterwachund TS7 der Firma Sanyo Inc. **Mitte:** ca. 1,5 m großer humanoider Roboter Asimo der Honda Corp. **Rechts:** ca. 70 cm großer humanoider Roboter SDR-4X der Sony Corp.

bio-mechanische Einblicke in die Physik des Gehens zu bekommen, um daraus abgeleitete Gesetzmäßigkeiten für die Konstruktion von effizienten Laufrobotern bzw. deren Kontrollprogrammen zu verwenden, andererseits sollen die Erkenntnisse in der Medizin verwendet werden, beispielsweise auf dem Gebiet der Prothetik.

Im Unterschied zu mobilen rad- bzw. kettengetriebenen Robotern, bei denen der Schwerpunkt auf der Navigation in bekannten oder unbekannten Domänen d.h. im Bereich der Bahn- und Handlungsplanung, der Kartenerstellung etc. liegt, wird bei den Laufrobotern das Hauptaugenmerk auf die Fortbewegung an sich gelegt. Die Vielfalt der natürlichen Vorbilder und die durch die Problemstellung gegebenen Herausforderungen an mechanische und elektronische Bauteile, Antriebe, Sensorik, Regelungssysteme, etc. stellen eine faszinierende Aufgabe dar. Die Bandbreite der untersuchten Roboter reicht dabei von schlangenähnlichen Robotern ohne Extremitäten über vielbeinige Roboter bis hin zu humanoiden Robotern (siehe Abbildung 5.6). Laufroboter, speziell humanoide Roboter, sind besonders in Japan, dem weltweit führenden Land in diesem Bereich, in den letzten Jahren intensiv untersucht worden. In Abbildung 1.1 sind einige der vor Kurzem vorgestellten Laufroboter aufgeführt.

Kleine Laufroboter wie z.B. der vierbeinige Sony Aibo, der 1999 in nur limitierter Stückzahl als Kleinserie produziert wurde, haben den Markt für anspruchsvolle „Spielzeugroboter“ geöffnet. Trotz der - für den privaten Bereich - relativ hohen Kosten von 1.600 $ sind innerhalb der ersten zwei Jahre nach Markteinführung über 50.000 Exemplare weltweit verkauft worden. Der Aibo-Roboter ist umfassend ausgestattet, beispielsweise mit Farbkamera, Prozessor, Sensorik und robuster Aktorik und wird aus diesem Grund auch verbreitet für die Forschung auf dem Gebiet der autonomen Laufroboter eingesetzt, vor allem im Roboterfußball [173].

Der Entertainment-Markt ist zur Zeit die wichtigste Perspektive für Anwendungen autonomer Laufroboter, seitdem bei Forschergruppen weltweit Ernüchterung über die Anwendbarkeit von autonomen Laufrobotern eingetreten ist. Die Komplexität des Problemfeldes, beispielsweise die enge Verzahnung von Form und Funktion der Beine, die Stabilität des Laufens, die hohe Materialbeanspruchung, und die gleichzeitig mangelnde Akzeptanz von Seiten der Industrie bei sicherheitsrelevanten Einsatzgebieten haben einen breiten Einsatz von Laufrobotern in die Ferne rücken lassen [99, 166].

1.3 Einflüsse der Biologie auf die Robotik

Einflüsse der Biologie sind sichtbar in vielen Bereichen der Roboterforschung. Die Biomechanik des Gehens wird beispielsweise anhand von Laufbewegungen von Tieren (Insekten, Nagetieren, Ziegen, Pferden, oder auch Elefanten) untersucht, um Rückschlüsse ziehen zu können aus der gegenseitigen Bedin-

gung von Form und Funktion von Extremitäten bei Tieren. Die Erkenntnisse der „funktionalen Morphologie“ von Säugetieren können dann beispielsweise als Vorbild bei der Konstruktion von Laufrobotern dienen [86, 175, 176].

Auch bei der Kontrolle von autonomen mobilen Robotern werden von den Forschergruppen Anleihen bei der Biologie gemacht. Es existiert seit Anfang der neunziger Jahre eine Arbeitsrichtung, die für die Steuerung autonomer mobiler Roboter Kombinationen voneinander unabhängiger Verhaltensprimitive verwendet. Diese verhaltensbasierte Robotik beruht auf Erkenntnissen über die Motoriksteuerung primitiver Lebewesen. Über Reflexe, automatisch ausgelösten Reaktionen auf spezifische Sensorinformationen, reagieren Lebewesen präzise auf Situationen, die eine schnelle Handlung verlangen. Reflexe sind dem normalen Handeln untergeordnet, bestimmen aber das Verhalten, wenn der Reiz ein gewisses Maß überschreitet und für eine rein kognitive Bestimmung der günstigsten Reaktion keine Zeit bleibt.

Durch Kombination verschiedener einfacher Verhaltensprimitive können komplexe Verhaltensweisen emergieren. Auf diese Weise ist es möglich, autonome Roboter in einer intelligenten Art und Weise zu kontrolliert. Natürliche Vorbilder für die Kontrolle autonomer mobiler Roboter waren und sind insbesondere neuronale Steuerungen, aber auch chemische bzw. hormonelle Kontrollarchitekturen sind Gegenstand der Forschung. Die bakterielle Chemotaxis zum Beispiel, die Bewegung eines einzelligen Lebewesens entlang eines Nährstoffgradienten in seiner Umwelt, beruht auf parallel ablaufenden, rein chemischen Informationsverarbeitungsprozessen und kann als Vorbild für eine rein reaktive Steuerung mobiler Roboter dienen [3]. Zum Thema verhaltensbasierte Robotik existiert eine Reihe von Konferenzen [88] und auch eine speziell dieser Thematik gewidmete Zeitschrift [89].

Hinsichtlich der Erforschung des Gehens sind speziell die Bewegungen von Insekten intensiv untersucht worden [52, 103]. Dies hat mehrere Gründe: Einerseits sind Insekten mehrbeinig und trotzdem in ihrer neuronalen Struktur einfach genug, um umfassend erforscht werden zu können. Andererseits sind Insekten äußerst erfolgreich im Verlaufe der Evolution gewesen und deshalb als Vorbild für die Konstruktion mehrbeiniger Laufmaschinen sehr interessant. Untersuchungen z.B. der Stabheuschrecke [47] haben direkten Einfluss auf den Entwurf von Kontrollmechanismen für sechsbeinige Laufmaschinen [60, 64, 44] gehabt.

Zu den für die Roboterforschung besonders interessanten Gebieten gehört auch der Bereich des kollektiven Verhaltens bzw. der kollektiven Intelligenz. Hierbei stehen Eigenschaften von Gruppen im Fokus des Interesses, die hinsichtlich ihrer Übertragbarkeit auf Teams von Robotern untersucht werden. Viele Tierarten, die in Herden leben, zeigen komplexe Verhaltensweisen, die aus der Interaktion miteinander und aus der Reaktion auf dynamische Veränderungen in der Umwelt resultieren. Untersuchungen von Schwarmverhalten (engl. *flocking*) haben zu erstaunlich simplen Grundmechanismen für das Verhalten der Individuen innerhalb der Gruppe geführt [75, 141, 169], die für die Formationsbildung beispielsweise im Roboterfußball oder auch für das Militär interessante Vorbilder darstellen [29].

Die gegenseitige Befruchtung mit Erkenntnissen in Biologie und Robotik ist auch ein Grund für die Einrichtung des Schwerpunktprogramms „Autonomes Laufen“[2] der Deutschen Forschungsgemeinschaft (DFG) gewesen. Arbeitsgruppen aus über 15 Universitäten aus Biologie, Informatik und Ingenieurwissenschaften arbeiten dort seit 1997 interdisziplinär und sehr erfolgreich an der Erforschung der Grundlagen, der Kontrolle und den Anwendungen von Laufrobotern.

[2] Unter der URL `http://www.fzi.de/ids/dfg_schwerpunkt_laufen/start_page.html` ist die Homepage des SPP „Autonomes Laufen“ im Internet zu finden.

Kapitel 2

Physikalische Simulation

Die Simulation der Dynamik eines realen Roboters setzt eine hinreichend genaue physikalische Modellbildung voraus. Hierbei wird eine konkrete Roboterarchitektur auf ein Mehrkörpersystem (MKS) abgebildet. Ein MKS ist eine endliche Menge starrer Körper, die in einen dreidimensionalen Raum eingebettet und untereinander physikalisch und/oder geometrisch gekoppelt sind. Physikalische Kopplung wird durch Beeinflussung mittels Kräften oder Momenten realisiert. Geometrische Kopplung bedeutet, dass geometrische Objekte (Punkte, Geraden, Ebenen) zweier oder mehrerer Objekte zusammenfallen. Ein Beispiel hierfür ist ein ideales Rotationsgelenk, bei dem die Rotationsachse Teil beider durch das Gelenk verbundener Körper ist. In Abbildung 2.1 ist ein MKS mit geometrischen und physikalischen Kopplungen dargestellt.

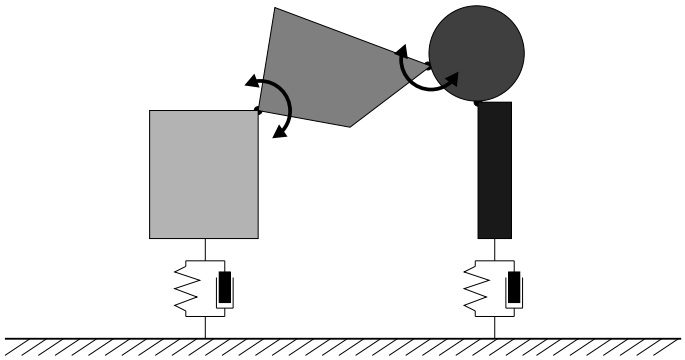


Abbildung 2.1: **Links:** Beispiel eines (2D-)Mehrkörpersystems mit geometrischen und physikalischen Kopplungen.

2.1 Kinematik

Das MKS ist zusätzlich in einem nicht zum System gehörenden Koordinatensystem I dargestellt (Welt- oder auch Inertialkoordinatensystem). Für jeden Körper des MKS existiert ein eigenes Bezugskoordinatensystem B_i (auch Basis genannt), gemäß dem z.B. auch die Angabe des Schwerpunktes oder der Gelenke erfolgt (Abbildung 2.5). Das MKS wird gebildet, indem eine minimale Anzahl von starren Körpern mit ihren jeweiligen Bewegungsmöglichkeiten sowie die geometrischen und physikalischen Kopplungen angegeben werden, die das reale System als idealisiertes Modell darstellen.

Die Kinematik des Systems legt die Bewegungsmöglichkeiten der Körper untereinander unter Berücksichtigung geometrischer Kopplungen fest. Diese geometrischen Kopplungen werden durch die Angabe von Bewegungsmöglichkeiten eines Körpers relativ zu einem Bezugskörper beschrieben (Relativkinematik). Der Körper, dessen Bewegungsmöglichkeiten beschrieben werden soll, ist mit dem Bezugskörper durch ein Gelenk verbunden. Es existieren Gelenke mit beliebigen Freiheitsgraden, in der Regel werden allerdings nur Gelenke mit einem Freiheitsgrad (engl. *degrees of freedom*, DOF) verwendet, also Rotations- oder Schubgelenke. Zur Beschreibung der Bewegung eines Körpers bezüglich seines Bezugskörpers werden Gelenkvariablen q_i eingeführt. Gelenkvariablen sind Größen, die die Lage der Basis eines Teilkörpers gegenüber der Lage der Basis des Bezugskörpers messen. Für Rotationsgelenke gibt q den Winkel, für Schubgelenke die Translation an. Der Freiheitsgrad f des Gelenkes legt die Anzahl der Gelenkvariablen $q_1 \dots q_f$ fest. Das MKS insgesamt wird mit dem Inertialsystem durch ein Gelenk verbunden. Dies ist meist ein „freies Gelenk“, d.h. ein Gelenk mit sechs Freiheitsgraden das als Referenz zum Inertialkoordinatensystem dient. Zur Vervollständigung der Beschreibung werden noch die äußeren eingeprägten Kräfte benötigt. Hierzu zählt vor Allem die Angabe des Schwerkraftvektors im Inertialkoordinatensystem, es sind aber auch noch andere Kräfte und Momente denkbar. Sind im realen System angetriebene Gelenke vorhanden, so müssen im Modell zeitabhängige Kraft- und Momentverläufe zwischen den beiden beteiligten Körpern definiert werden. Die Angabe der Kräfte (für Schubgelenke) und Momente (für Rotationsgelenke) erfolgt dabei im körpereigenen Bezugskoordinatensystem.

2.2 Dynamik

Die Bewegung von Mehrkörpersystemen setzt sich natürlicherweise zusammen aus den Bewegungen der einzelnen Teilkörper. Im folgenden Abschnitt sollen kurz die wesentlichen Grundlagen der Dynamiksimulation von MKS gegeben werden. Hierbei wird zuerst die Dynamik von punktförmigen Massen erläutert. Darauf aufbauend folgt die Definition der mathematischen Grundlagen der Dynamik starrer Körper. Anschließend wird die Dynamiksimulation von MKS beschrieben.

Für eine detaillierte Einführung in die Thematik sei an dieser Stelle z.B. auf [31, 143] verwiesen.

Bewegung punktförmiger Körper

Die Bewegung eines starren Körpers kann in erster Näherung als die Bewegung eines punktförmigen Körpers (Partikel) betrachtet werden. Hierbei beschreibt die Funktion $x(t)$ den Ort des Partikels im Inertialkoordinatensystem zur Zeit t. Die Funktion $v(t) = \dot{x}(t) = \frac{d}{dt}x(t)$ beschreibt die Geschwindigkeit des Partikels zum Zeitpunkt t. Der *Zustand* $Y(t)$ eines Partikels zum Zeitpunkt t wird also durch die Angabe der aktuellen Position und der aktuellen Geschwindigkeit vollständig beschrieben (siehe Abbildung 2.2). Der Zustandsvektor $Y(t)$ des Systems ist also definiert als

$$Y(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix}. \tag{2.1}$$

Da die Position und die Geschwindigkeit im dreidimensionalen Raum beschrieben werden, ist $Y(t)$ ein sechs-Tupel

$$Y(t) = \begin{pmatrix} x_x(t) \\ x_y(t) \\ x_z(t) \\ v_x(t) \\ v_y(t) \\ v_z(t) \end{pmatrix}. \tag{2.2}$$

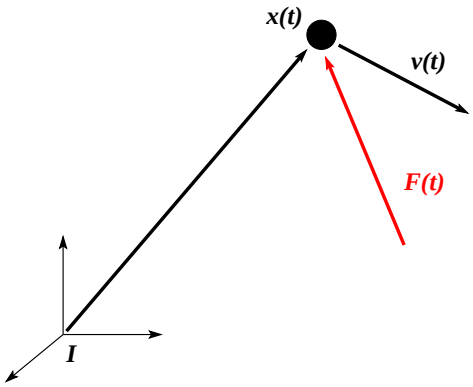


Abbildung 2.2: Der Zustand $Y(t)$ eines Partikels wird durch die Position $x(t)$ und die Geschwindigkeit $v(t)$ beschrieben. Für die Berechnung der Partikelbewegung ist zusätzlich die Angabe der zum Zeitpunkt t wirkenden Kräfte $F(t)$ nötig.

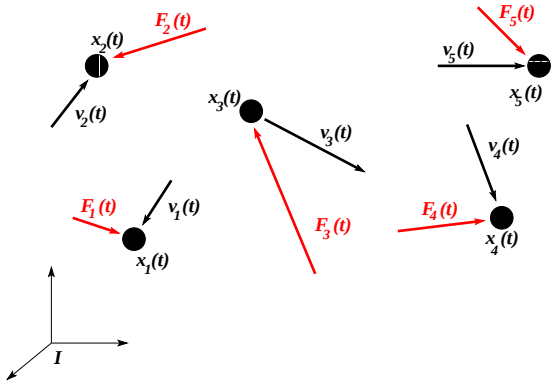


Abbildung 2.3: Ein System mit mehreren Partikeln. Der Zustand $Y(t)$ wird durch Angabe der Positionen $x_i(t)$, Geschwindigkeiten $v_i(t)$ und wirkenden Kräfte $F_i(t)$ aller n Partikel vollständig beschrieben.

Um die Bewegung eines Partikels tatsächlich beschreiben zu können, ist die Angabe der auf das Partikel wirkenden Kräfte notwendig (siehe Abbildung 2.2). Hierbei wird $F(t)$ definiert als die Summe aller zum Zeitpunkt t auf das Partikel wirkenden Einzelkräfte (Gravitation, Wind, Federkräfte, etc.). Besitzt das Partikel eine Masse m, so kann nun die Änderung des Zustandes $Y(t)$ über die Zeit (die Dynamik) beschrieben werden durch

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ F(t)/m \end{pmatrix}. \tag{2.3}$$

Bei beliebigem Anfangswert $Y(t)$ beschreibt Gleichung (2.3) die Änderung des Zustandes zum Zeitpunkt t. Ein System mit n Partikeln (siehe Abbildung 2.3) wird analog dazu beschrieben durch

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x_1(t) \\ v_1(t) \\ \vdots \\ x_n(t) \\ v_n(t) \end{pmatrix} = \begin{pmatrix} v_1(t) \\ F_1(t)/m \\ \vdots \\ v_n(t) \\ F_n(t)/m \end{pmatrix}. \tag{2.4}$$

Dieses System von Differentialgleichungen kann nun mithilfe eines numerischen Verfahrens gelöst werden. Hierfür ist die Angabe eines Anfangszustandes $Y(0)$ notwendig. Für ein System mit n Partikeln muss

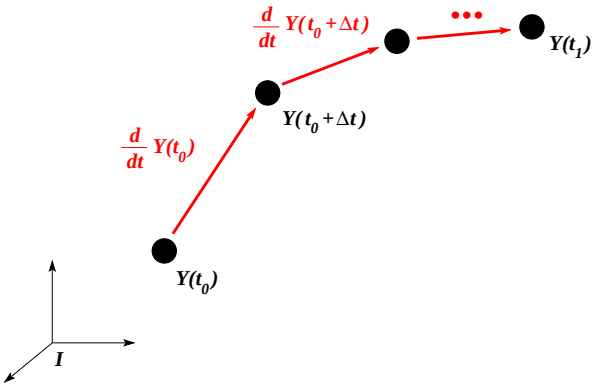


Abbildung 2.4: Numerische Berechnung der Bewegung eines Partikels. Ausgehend von einem Anfangswert $Y(t_0)$ werden die Zustände $Y(t_0+\Delta t)$ berechnet, die jeweils die Grundlage für den nächsten Berechnungsschritt darstellen.

also ein Differentialgleichungssystem mit $6n$ Gleichungen numerisch gelöst werden. Hierfür wird der Zustandsvektor $Y(t)$ zwischen Anfangszeitpunkt t_0 und Endzeitpunkt in kleinen Intervallen Δt berechnet (siehe Abbildung 2.4). Die numerische Lösung von Gleichung (2.4) ist bei geeigneter Schrittweite $h = \Delta t$ und ausgehend von $Y(t_0) = y_0$ und $\frac{d}{dt}Y(t_0) = Y_0$ beispielsweise über das rekursive Runge-Kutta-Verfahren 4. Ordnung möglich:

$$\begin{aligned} Y_{i+1} &= Y_i + h(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4) \text{ mit} \\ k_1 &= f(y_i, Y_i) \\ k_2 &= f(y_i + \frac{h}{2}, Y_i + k_1\frac{h}{2}) \\ k_3 &= f(y_i + \frac{h}{2}, Y_i + k_2\frac{h}{2}) \\ k_4 &= f(y_i + h, Y_i + hk_3). \end{aligned} \tag{2.5}$$

Hierbei wird zu jedem Zeitpunkt $t_0 + i \cdot h$ ein Näherungswert Y_i berechnet. Durch die vier Stützstellen des Verfahrens wird eine relativ hohe Genauigkeit erzielt. Das Runge-Kutta-Verfahren ist ein Standardverfahren zur numerischen Integration von Differentialgleichungssystemen. Durch die Verwendung von fehlergrößengesteuerten Schrittweiten kann die Genauigkeit des Verfahrens gesteigert werden, wobei die zusätzliche Laufzeit gegen den Gewinn an Genauigkeit abgewogen werden muss

Die Bewegung starrer Körper

Für die Simulation tatsächlicher starrer Körper (beispielsweise der Teilkörper eines MKS) wird der Zustandvektor $Y(t)$ erweitert, d.h. er enthält mehr Informationen über das System. Zusätzlich wird die Ableitung von $Y(t)$ komplizierter. Die grundlegende Beziehung aus Gleichung (2.3) bleibt allerdings bestehen.

Position und Orientierung

Die Position eines Partikels im Inertialkoordinatensystem wird vollständig durch den Ortsvektor $x(t)$ beschrieben. Die Beschreibung von Körpern erfordert zusätzlich die Angabe einer Orientierung. Für die vollständige Beschreibung eines Körpers im Inertialkoordinatensystem benötigt man also $x(t)$, die Translation des Körpers, und die Orientierung $R(t)$, eine 3×3-Matrix, die die Rotation des Körpers um die drei Raumachsen beschreibt.

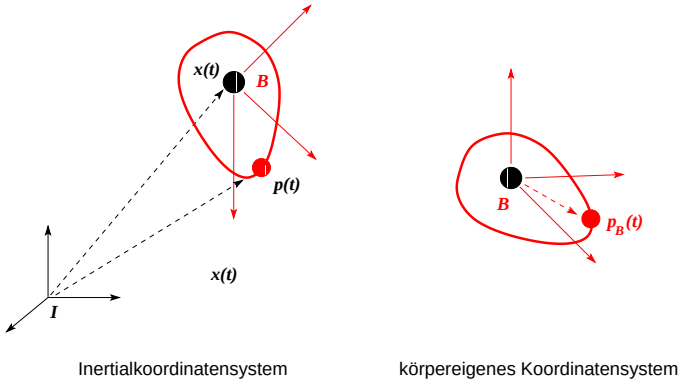


Abbildung 2.5: Teilkörper besitzen ein eigenes Koordinatensystem B bezüglich dem die Position aller Punkte des Körpers angegeben werden. Der Ursprung von B fällt mit dem Schwerpunkt des Körpers zusammen.

Ein Körper nimmt ein bestimmtes Volumen ein und besitzt eine definierte Form. Die Form eines Körpers wird bezüglich eines zum Körper gehörenden Koordinatensystems angegeben (siehe Abbildung 2.5), d.h. die Beschreibung der Position aller Punkte des Körpers erfolgt in lokalen Koordinaten. Dies ist möglich, da starre Körper nur verschoben und rotiert werden können, die Relativpositionen der Punkte des Körpers aber unverändert bleiben. Der Ursprung des körpereigenen Koordinatensystems wird so definiert, dass er mit dem Schwerpunkt des Körpers zusammenfällt. Der Schwerpunkt des Körpers hat also die Position (0,0,0) im körpereigenen Koordinatensystem.

Die Position einzelner Punkte p_B des Körpers im Inertialkoordinatensystem I wird also berechnet durch

$$p_I(t) = R(t)p_B + x(t). \tag{2.6}$$

Dadurch, dass der Schwerpunkt des Körpers im Ursprung des körpereigenen Koordinatensystems liegt, wird seine Position im Inertialkoordinatensystem direkt durch $x(t)$ beschrieben. Die Spalten der Rotationsmatrix

$$R(t) = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{zx} & r_{zy} & r_{yy} \end{pmatrix} \tag{2.7}$$

beschreiben die Richtung der Achsen des körpereigenen Koordinatensystems im Inertialkoordinatensystem.

Beispiel

Die x-Achse (1,0,0) des Körperkoordinatensystems B wird durch $R(t)$ transformiert nach

$$R(t)\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix}. \tag{2.8}$$

Zusammengefasst können $x(t)$ und $R(t)$ also Position und Orientierung eines Körpers genannt werden.

Translatorische und rotatorische Geschwindigkeit

Analog zu Position $x(t)$ und Orientierung $R(t)$, die für die vollständige Beschreibung der Lage eines Körpers notwendig sind, wird die Geschwindigkeit eines Körpers ebenfalls durch Angabe zweier Größen $\dot{x}(t)$ und $\dot{R}(t)$ vollständig beschrieben.

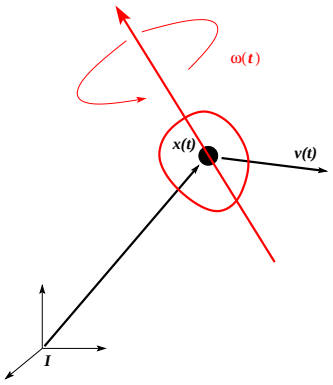


Abbildung 2.6: Translatorische Geschwindigkeit $v(t)$ und Rotationsachse $\omega(t)$ bzw. Winkelgeschwindigkeit $|\,\omega(t)\,|$ eines Körpers.

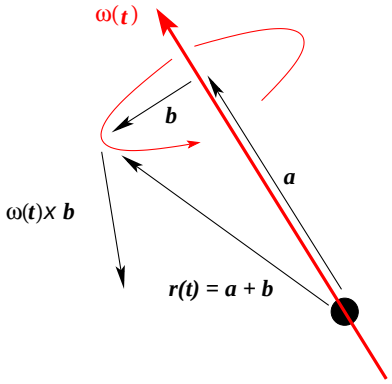


Abbildung 2.7: Rotation des Vektors $r(t)$ um $\omega(t)$.

Da $x(t)$ die Position des Schwerpunktes im Inertialsystem beschreibt, wird mit $v(t) = \dot{x}(t)$ die translatorische Geschwindigkeit des Schwerpunktes bezeichnet. Zusätzlich zur linearen Bewegung des Schwerpunktes kann ein Körper rotieren. Diese Drehbewegung wird mit dem Vektor $\omega(t)$ bezeichnet. Die Richtung von $\omega(t)$ gibt die Richtung der Achse an, um die sich der Körper dreht. Der Betrag des Vektors, $|\,\omega(t)\,|$, beschreibt die Geschwindigkeit der Rotation (siehe Abbildung 2.6). $|\,\omega(t)\,|$ wird Winkelgeschwindigkeit genannt. Die Änderung der Rotationsmatrix, $\dot{R}(t)$, lässt sich aus $\omega(t)$ auf die im Folgenden beschriebene Weise berechnen.

Die Rotation $\omega(t)$ eines Körpers bedeutet, dass die Vektoren des körpereigenen Koordinatensystems B ebenfalls um $\omega(t)$ gedreht werden. Für die Berechnung der Änderung $\dot{r}$ eines beliebigen Vektors r, der um $\omega(t)$ mit der Geschwindigkeit $|\,\omega(t)\,|$ rotiert, wird r in zwei Vektoren a und b aufgeteilt

$$r(t) = a + b, \tag{2.9}$$

wobei a den Anteil an r beschreibt, der parallel zu $\omega(t)$ verläuft, und b senkrecht zu $\omega(t)$ steht (siehe Abbildung 2.7). Das Ende des Vektors $r(t)$ rotiert mit einer Geschwindigkeit von $|\,\omega(t)\,| \cdot |\,b\,| = |\,\omega(t) \times b\,|$ um $\omega(t)$. Diese Geschwindigkeit wird bezeichnet als die Tangentialgeschwindigkeit eines Punktes, der mit Abstand b um $\omega(t)$ rotiert. Die Richtung der Bewegung ist durch das Kreuzprodukt $\omega(t) \times b$ gegeben, d.h.

die Bewegung ist orthogonal sowohl zu $\omega(t)$ als auch zu b. Hierdurch ergibt sich

$$\begin{aligned} \dot{r}(t) &= \omega(t) \times b \\ &= \omega(t) \times b + \omega(t) \times a \qquad \text{(wegen } \omega(t) \times a = 0\text{)} \\ &= \omega(t) \times (b + a) \\ &= \omega(t) \times r. \end{aligned} \tag{2.10}$$

Für die Geschwindigkeit von R kann man nun schreiben

$$\dot{R} = \left(\omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \qquad \omega(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \qquad \omega(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right). \tag{2.11}$$

Das Kreuzprodukt von $\omega(t)$ und den Spaltenvektoren r von R kann auch als eine Matrix-Vektor-Multiplikation der Matrix $\tilde{\omega}(t)$ und dem Vektor r beschrieben werden, wobei $\tilde{\omega}(t)$ definiert wird durch folgende Beziehung:

$$\omega(t) \times r = \begin{pmatrix} \omega_y(t) r_z - r_y \omega_z(t) \\ -\omega_x(t) r_z + r_x \omega_z(t) \\ \omega_x(t) r_y - r_x \omega_y(t) \end{pmatrix} = \begin{pmatrix} 0 & -\omega_z(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{pmatrix} r = \tilde{\omega}(t) r, \tag{2.12}$$

so dass nun Gleichung (2.11) auch geschrieben werden kann als

$$\dot{R}(t) = \tilde{\omega}(t) \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix}. \tag{2.13}$$

Die Matrix auf der rechten Seite ist $R(t)$, deshalb kann für $\dot{R}(t)$ vereinfacht geschrieben werden

$$\dot{R}(t) = \tilde{\omega}(t) R(t). \tag{2.14}$$

Durch die Angabe von translatorischer Geschwindigkeit $v(t) = \dot{x}(t)$ und der Winkelgeschwindigkeit $\omega(t)$ bzw. der Matrix $\tilde{\omega}(t)$ wird die Geschwindigkeit eines starren Körpers im Inertialkoordinatensystem vollständig beschrieben.

Die Position einzelner Punkte p_I des Körpers im Inertialkoordinatensystem I wird (nach Gleichung (2.6), zur Veranschaulichung nochmals aufgeführt) angegeben mit

$$p_I(t) = R(t) p_B + x(t). \tag{2.15}$$

Äquivalent kann nun die Geschwindigkeit $\dot{p}_I(t)$, d.h. die Geschwindigkeit im Inertialkoordinatensystem, angegeben werden durch

$$\begin{aligned} \dot{p}_I(t) &= \tilde{\omega}(t) R(t) p_B(t) + v(t) \\ &= \dot{R}(t) p_B(t) + v(t). \end{aligned} \tag{2.16}$$

Diese Geschwindigkeit setzt sich aus einer linearen Komponente $v(t)$ und einer durch die Körperrotation verursachten Komponente $\dot{R}(t) p_B(t)$ zusammen.

Für die Beschreibung des Zustandes $Y(t)$ eines starren Körpers muss $Y(t)$ also um die Beschreibung der Rotation $R(t)$ und der Winkelgeschwindigkeit $\omega(t)$ erweitert werden, so dass gilt:

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ v(t) \\ \omega(t) \end{pmatrix}. \tag{2.17}$$

Die Änderung des Zustandes $\dot{Y}(t)$ wird analog beschrieben durch

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ v(t) \\ \omega(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \tilde{\omega}(t)R(t) \\ \dot{v}(t) \\ \dot{\omega}(t) \end{pmatrix}. \tag{2.18}$$

Die Bestimmung der Ableitungen von $v(t)$ und $\omega(t)$ nach der Zeit, d.h die Bestimmung der linearen und der Winkelbeschleunigung, wird im folgenden Abschnitt erläutert.

Masse und Schwerpunkt starrer Körper

Die Gesamtmasse M eines Körpers kann vereinfacht angenommen werden als

$$M = \sum_{i=1}^{N} m_i, \tag{2.19}$$

d.h als die Summe der Massen aller N Partikel p_i, aus denen der Körper besteht. Der Schwerpunkt des Körpers im Inertialkoordinatensystem ist dann definiert als

$$x(t) = \sum_{i=1}^{N} m_i p_i(t), \tag{2.20}$$

wobei die Position der Partikel bezüglich des Inertialkoordinatensystems angegeben wird.

Kräfte und Momente

Eine Kraft $f_i(t)$, die auf einen starren Körper einwirkt (Gravitation, Windkraft, etc.), greift an einem bestimmten Punkt p_i an. Sei $F(t)$ die Summe aller auf den Körper einwirkenden Kräfte $f(t)$ zum Zeitpunkt t

$$F(t) = \sum_{i=1}^{n} f_i(t), \tag{2.21}$$

so gilt

$$\dot{v}(t) = \frac{F(t)}{M}, \tag{2.22}$$

d.h. der Körper erfährt eine zu den wirkenden Kräften $\sum f_i(t) = F(t)$ proportionale lineare Beschleunigung. Zusätzlich wird durch $F(t)$, genauer gesagt durch die Verteilung der einzelnen f_i bzw. deren Angriffspunkte p_i über den Körper, ein Moment erzeugt, das definiert ist als

$$\tau(t) = \sum_{i=1}^{n} (p_i - x(t)) \times f_i(t), \tag{2.23}$$

d.h. als die Summe der durch die einzelnen Kräfte und Kraftangriffspunkte verursachten Drehmomente (siehe Abbildung 2.8).

Dieses Gesamtmoment $\tau(t)$ erzeugt eine rotatorische Beschleunigung $\dot{\omega}(t)$ des Körpers. Durch die Geometrie des Körpers und die Verteilung der Massen der Partikel (also die Dichte) des Körpers werden orts- und masseverteilungsabhängige Beschleunigungen erzeugt. Dies bedeutet anschaulich, das die gleiche Kraft unterschiedliche Winkelbeschleunigungen erzeugt, wenn sie an unterschiedlichen Stellen am Körper angreift. Die Beschreibung dieser Relation von Kraft und resultierender Beschleunigung wird Trägheitstensor genannt. Der (rotationsabhängige) Trägheitstensor $I(t)$ ist definiert als

$$I(t) = \begin{pmatrix} I_{xx} & I_{yx} & I_{zx} \\ I_{xy} & I_{yy} & I_{zy} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix}. \tag{2.24}$$

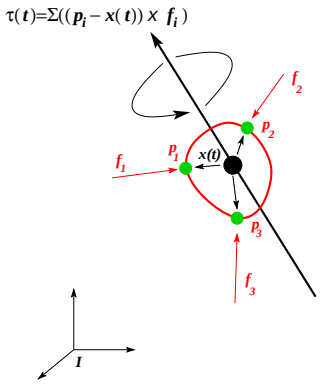


Abbildung 2.8: Durch die angreifenden Kräfte f_i wird ein Drehmoment $\tau(t)$ erzeugt.

Die Matrixeinträge in $I(t)$ sind definiert als

$$I_{xx} = M \int_V (y^2 + z^2)\, dV \text{ (Diagonalenelemente,)} \tag{2.25}$$

$$I_{xy} = -M \int_V xy\, dV \text{ (sonst).} \tag{2.26}$$

Die Winkelbeschleunigung $\dot{\omega}(t)$ ist also nicht nur abhängig von $\omega(t)$ sondern auch von $I(t)$, so dass gilt

$$I(t)\frac{d}{dt}\omega(t) = \tau(t). \tag{2.27}$$

Der Zustand eines starren Körpers kann nun vollständig beschrieben werden durch

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ v(t) \\ \omega(t) \end{pmatrix}. \tag{2.28}$$

Die Änderung des Zustandes $\dot{Y}(t)$ wird analog beschrieben durch

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ v(t) \\ \omega(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \tilde{\omega}(t)R(t) \\ F(t)/M \\ \tau(t)I^{-1}(t) \end{pmatrix}. \tag{2.29}$$

Die Bewegung von Mehrkörpersystemen

Die Dynamik eines MKS resultiert aus den Einzelbewegungen der im MKS enthaltenen starren Körper. Für jeden Teilkörper wird die Dynamik vollständig durch Angabe von $\dot{Y}(t)$ nach Gleichung (2.29) beschrieben. Die in der Struktur des MKS enthaltenen geometrischen und physikalischen Kopplungen, die die Bewegungsmöglichkeiten der Teilkörper relativ zueinander beschreiben, müssen für die Angabe von $Y(t)$ jedoch zuvor geeignet extrahiert werden und in $Y(t)$ bzw. $\dot{Y}(t)$ einfließen.

Die Struktur des MKS aus Gliedern und Gelenken wird kinematische Kette genannt. Je nach Konfiguration der Gelenke wird zwischen offenen und geschlossenen Ketten unterschieden (siehe Abbildung 2.9). Es ist

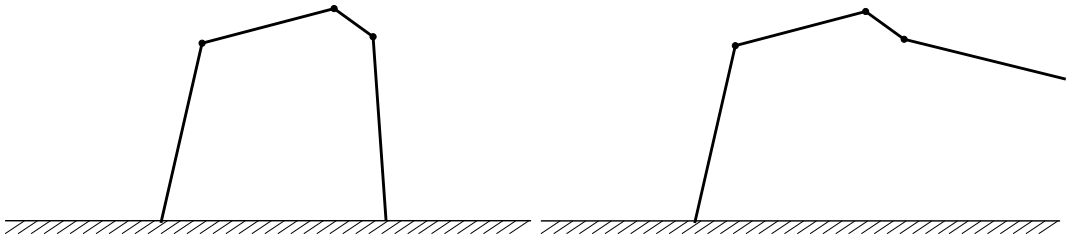


Abbildung 2.9: **Links:** Beispiel einer geschlossenen kinematischen Kette. **Rechts:** Offene kinematische Kette.

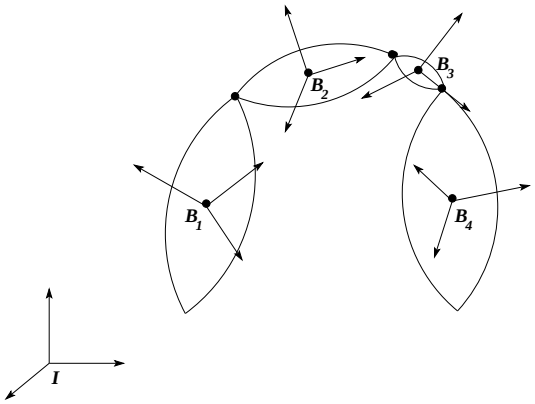


Abbildung 2.10: Schematische Darstellung eines MKS. Eingezeichnet sind die Basissysteme der Teilkörper sowie die Verbindungspunkte zwischen den Körpern.

für die Simulation von erheblicher Bedeutung, ob geschlossene oder offene kinematische Ketten vorliegen. Geschlossene kinematische Ketten verursachen wegen der Bindung der Enden der Kette eine Erhöhung der Komplexität, die sich vor Allem in veränderter (verschlechterter) Laufzeit der Dynamiksimulation bemerkbar macht. In der Regel werden geschlossene kinematische Ketten intern in offene kinematische Ketten mit zusätzlichen Kopplungen der Endglieder und zusätzlichen Randbedingungen überführt. Bei Laufmaschinen mit mehreren Extremitäten existieren auch mehrere kinematische Ketten, die je nach Laufsituation geschlossen oder offen sein können.

Die Anzahl der zur eindeutigen Beschreibung der Lage aller Körper eines MKS mit kinematischer Baumstruktur erforderlichen Variablen heißt Freiheitsgrad des MKS. Die Gelenkvariablen aller Körper des MKS bilden eine solche eindeutige und vollständige Beschreibung. In Abbildung 2.10 ist ein MKS mit den Basissystemen der Teilkörper abgebildet.

Der Zustand $Y_{MKS}(t)$ des MKS, also die Position und Orientierung sowie die lineare und die Winkelgeschwindigkeit der einzelnen Teilkörper, wird nun bestimmt, indem der Zustand $Y_{Ref}(t)$ eines Referenzkörpers im Inertialkoordinatensystem angegeben wird. Relativ zu diesem Teilkörper wird nun der Zustand $Y_{Ref+1}(t)$ des in der kinematischen Kette folgenden Teilkörpers berechnet und in das Inertialkoordinatensystem transformiert. Entsprechend wird mit den nächsten Körpern verfahren, bis man an das Ende der Kette gelangt.

Für die Kopplung der Teilkörper wird die kinematische Kette an den Gelenken freigeschnitten, d.h. die Kette wird in ihre Glieder zerlegt. Die an den Schnittpunkten wirkenden Kräfte werden mit jeweils umgekehrten Vorzeichen für beide Körper angetragen (siehe Abbildung 2.11).

Die Bestimmung der Winkel und Winkelgeschwindigkeiten aufeinanderfolgender Teilkörper kann rekursiv

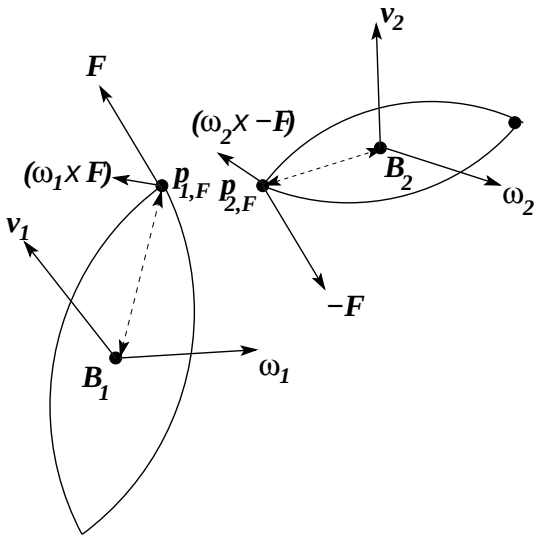


Abbildung 2.11: Zwei Teilkörper werden am Gelenkpunkt freigeschnitten. v_i, v_2 beschreiben die linearen Geschwindigkeiten der Teilkörper, ω_1, ω_2 beschreiben die Winkelgeschwindigkeiten. Die durch das Freischneiden einzutragenden Kräfte F bzw. $-F$ erzeugen unterschiedliche Momente ($\omega_1 \times F$ bzw $\omega_2 \times -F$).

erfolgen, denn es gilt

$$\begin{aligned} \omega_i(t) &= \omega_{i-1}(t) + \omega_{rel} \quad \text{und} \\ \omega_{Ref} &= \omega_{rel}, \end{aligned} \tag{2.30}$$

d.h. die Winkel und Winkelgeschwindigkeit eines Teilkörpers setzt sich zusammen aus den entsprechenden Werten des vorherigen Körpers und einer relativen Verdrehung T. Da $R_i(t)$ und $R_{i-1}(t)$ die Basen der Teilkörper i und $i-1$ in das Inertialsystem transformieren (siehe den entsprechenden Abschnitt zu Position und Orientierung), ist die relative Rotation $T_i(t)$ des Körpers i zum Körper $i-1$ also gegeben durch

$$\begin{aligned} R_i(t) &= R_{i-1}(t)T_i(t), \text{ d.h.} \\ T_i(t) &= R_i(t)R_{i-1}^{-1}(t). \end{aligned} \tag{2.31}$$

Die Position zweier aufeinanderfolgender Teilkörper lässt sich im Inertialkoordinatensystem darstellen als

$$x_i(t) = x_{i-1}(t) + y_i(t), \tag{2.32}$$

wobei $y_i(t)$ den Verbindungsvektor zwischen den jeweiligen Koordinatensystemen darstellt. Für $y_i(t)$ gilt nun

$$y_i(t) = R_i(t)v_i(t) + R_{i-1}(t)w_{i-1}(t). \tag{2.33}$$

Die Vektoren $v_i(t)$ und $w_{i-1}(t)$ stellen dabei die Lage des Gelenkpunktes in den jeweiligen lokalen Koordinatensystemen dar. Für $x_i(t)$ gilt nun

$$x_i(t) = x_{i-1}(t) + R_i(t)v_i(t) + R_{i-1}(t)w_{i-1}(t). \tag{2.34}$$

Analog lässt sich für die lineare Beschleunigung $\dot{x}(t)$ schreiben

$$\dot{x}_i(t) = \dot{x}_{i-1}(t) + \dot{R}_i(t)v_i(t) + \dot{R}_{i-1}(t)w_{i-1}(t). \tag{2.35}$$

Die Berechnung der Position, Orientierung, Geschwindigkeit und Beschleunigung der einzelnen Teilkörper auf diese Weise wird *Vorwärtstransformation* genannt.

Für die Dynamiksimulation müssen die Trägheitstensoren der einzelnen Teilkörper berechnet werden. Bei einfachen geometrischen Objekten sind die Trägheitsmomente aus Tabellenwerken entnehmbar, bei komplexen Körpern müssen sie über ein Volumenintegral berechnet werden. Die Angabe der Hülle eines Körpers kann beispielsweise über eine VRML[1]-Datei erfolgen [14].

Abhängig von der aktuellen Konfiguration werden die Bewegungsgleichungen (Differentialgleichungen) generiert, die anschließend im Zeitbereich gelöst werden. Dies erfolgt durch numerische Integration der Gleichungssysteme. Die Bewegungen der Teilkörper des MKS sind allein abhängig von den Freiheitsgraden der Gelenke, die vor dem jeweiligen Körper in der kinematischen Kette liegen. Roboter besitzen meist nur einen Freiheitsgrad pro Gelenk, so dass für die Beschreibung der Bewegung aller *i* Teilkörper im MKS nur *i* Variablen benötigt werden, d.h. das Gleichungssystem zur Beschreibung von $\dot{Y}_{MKS}(t)$ besitzt *i* Gleichungen.

Für die Lösung der Bewegungsgleichungen sind Anfangsbedingungen notwendig. Die Angabe kann einerseits durch Setzen expliziter Werte für jede Variable erfolgen. Oft erfolgt die Festlegung der Anfangsbedingungen durch Angabe absoluter Positionen (im Inertialkoordinatensystem) beliebiger körperfester Punkte aller Teilkörper des MKS zum Anfangszeitpunkt. Hieraus werden dann die Anfangswerte der Gelenkvariablen berechnet. In diesem Falle werden die Anfangswerte der Ableitungen (also die Geschwindigkeiten der einzelnen Körper) auf Null gesetzt, d.h. das Gesamtsystem befindet sich in Ruhelage.

2.3 Kollisionserkennung

Beim Start der numerischen Integration der Bewegungsgleichungen wird der Körper in Ruhelage angenommen. Die eingeprägten äußeren Kräfte (vor Allem die Schwerkraft) bewirken eine Beschleunigung des MKS. Auf das MKS wirken jedoch noch andere Kräfte/Momente ein, die die einzelnen Körper unterschiedlich beschleunigen oder abbremsen. Hierzu zählen vor Allem Kräfte/Momente, die durch aktive Gelenke (also Motoren im realen Vorbild) auf die angrenzenden Körper ausgeübt werden, aber auch durch Massenträgheit verursachte Gegenkräfte bei Beschleunigungen. Von besonderer Bedeutung sind Kräfte, die durch Kollisionen einzelner Körper des MKS mit anderen Objekten erzeugt werden. Hierzu zählen Kollisionen mit dem Boden oder auch Kollisionen von Körpern des MKS untereinander. Kollidieren die Oberflächen zweier Objekte nicht parallel zu ihren Normalenvektoren, entstehen Reibkräfte. Auch Reibkräfte wirken sich auf die Bewegungen des Gesamtsystems aus (gerade bei Laufmaschinen wird der Vortrieb des Roboters durch reibschlüssige Kraftübertragung zwischen Extremitäten und Inertialsystem erzeugt) und müssen erkannt und berücksichtigt werden.

Die Kollisionserkennung wirkt sich deutlich auf die Laufzeit der Dynamiksimulation aus, da für jeden Körper des MKS zu jedem Integrationsschritt überprüft werden muss, ob eine Kollision mit einem anderen Objekt vorliegt. Hierfür ist theoretisch eine vollständige Überprüfung der gesamten Oberfläche jedes Körpers notwendig. In einigen Programm-Bibliotheken werden laufzeitoptimierte Kollisionserkennungsalgorithmen verwendet. DynaMechs [115] beispielsweise überprüft vor jedem Integrationsschritt nur alle auf den Hüllkörpern definierten Punkte. Eine Kollisionserkennung wird also abhängig vom Grad der Detailliertheit der einzelnen Körper genauer oder ungenauer. Diese Abweichung von der physikalischen Realität macht sich in den resultierenden Bewegungen in der Regel nicht bemerkbar, wenn die Körper feinkörnig genug modelliert werden. Kollisionskräfte werden über die kinematischen Ketten des MKS propagiert und wirken sich unmittelbar auf die Bewegungen aller Körper des Gesamtsystems aus. Kollisionen werden in der Dynamiksimulation unterschiedlich gehandhabt, je nach verwendetem Stoßmodell (z.B. impulsbasiert [118] oder durch Feder-Dämpfer-Systeme [115]).

Ein der Kollision ähnlicher Fall ist die Einschränkung der Beweglichkeit von Gelenken. Gelenke werden mit einem Arbeitsraum definiert, d.h. unter Angabe der minimalen und maximalen Auslenkung aus der

[1] Eine Beschreibung der Hüllkörper einzelner Glieder in VRML (*virtual reality markup language*) wird z.B. im SIGEL-System (Abschnitt 6.2) verwendet

(zusätzlich anzugebenden) Nullposition. Wird nun ein Gelenk über die zulässige Auslenkung bewegt, werden Reaktionskräfte wie bei einer Kollision berechnet. Gleichzeitig muss ein Überschwingungsdämpfungsmoment berechnet werden, das ein Schwingen des Gelenkes um den Anschlag dämpft. Die Reaktionskräfte werden gemäß den Feder- und Dämpferkonstanten des Gelenkes berechnet.

2.4 Einschränkungen

Die hauptsächlichen Einschränkungen bei der Dynamiksimulation werden durch die verwendeten numerischen Verfahren zur Lösung der Bewegungsdifferentialgleichungen hervorgerufen. Die Lösung der Gleichungssysteme, deren Dimension von der Anzahl der Körper im MKS und den globalen Zwangsbedingungen abhängt, mit numerischen Verfahren bedeutet immer eine Fehlerinjektion durch mathematische Ungenauigkeiten. Es ist möglich, dass diese Fehler schwerwiegende Auswirkungen haben, die bis zum völligen Versagen des Verfahrens führen können. Fehlerrobuste Verfahren, die beispielsweise eine geringe Fehlerordnung besitzen, erfordern allerdings im Gegenzug immer kleinere bzw. adaptive Schrittweiten des Lösers. Auf diese Weise sind die Lösungen der Bewegungsgleichungen zwar stabil, der Zeitfortschritt steht allerdings in keinem Verhältnis zum Rechenaufwand (siehe hierzu auch Kapitel 9). Sollen mehrere voneinander unabhängige mechanische Systeme simuliert werden, bedeutet dies die parallele Lösung mehrerer Gleichungssysteme und einen daraus resultierenden Anstieg des Rechenaufwands. Der Aufwand steigt entsprechend, wenn mehrere MKS in Kollisionen miteinander verwickelt sind.

Zusätzlich erschwerend wirken sich große Kräfte/Momente in den Gelenken aus, die durch Antriebe bzw. Überdrehungen von Gelenken erzeugt werden. Auch in diesem Fall hängt die Qualität der Simulation entscheidend vom verwendeten numerischen Verfahren ab. Dies ist von besonderer Bedeutung für den im weiteren Verlauf der Arbeit beschriebenen Fall der zufällig erzeugten Kontrollprogramme. Hier sind eratisch wirkende Kräfte/Momente in den Gelenken eines MKS über kurze Zeitperioden zu erwarten, die durch die Struktur der zufälligen Programme erzeugt werden und die nicht zu numerischen Instabilitäten führen dürfen.

Die realitätsnahe Simulation der Dynamik von Mehrkörpersystemen ist ein Wissenschaftsgebiet für sich, das mit der Zunahme der verfügbaren Rechnerleistung in den letzten Jahren erheblich an Bedeutung gewonnen hat. Hauptziel ist hierbei, effiziente Algorithmen und Datenstrukturen für hochdimensionale Matrixoperationen zu entwickeln, die die Laufzeitordnungen der Kräftetransformationen entlang der kinematischen Ketten verringern ([61, 143]).

Im nächsten Kapitel folgt eine Einführung in die Genetische Programmierung, die in Verbindung mit der Dynamiksimulation von Laufrobotern verwendet wird, um Kontrollprogramme für beliebige Laufroboterarchitekturen zu evolvieren.

Kapitel 3

Genetische Programmierung

> *Genetic Programming is fundamentally different from other approaches to artificial intelligence, machine learning, adaptive systems, automated logic, expert systems, and neural networks in terms of (i) its representation (namely, programs), (ii) the role of knowledge (none), (iii) the role of logic (none), and (iv) its mechanism (gleaned from nature) for getting to a solution within the space of possible solutions.*
>
> J. R. Koza, in [108]

> *First and foremost we will consider the induction of computer programs by evolutionary means [and] we do not limit our definition of GP to include only systems that use (expression) trees to represent programs. Instead, all means of representing programs will be included. As long as there is a population of programs or algorithms [...] and as long as some kind of indeterminism is applied to generate new variants, [a system] can legitimately [be called] a genetic programming system.*
>
> W. Banzhaf, P. Nordin, R. E. Keller und F. Francone, in [30]

Genetisches Programmieren erzeugt Computerprogramme oder Algorithmen in einem evolutionären Prozess und stellt damit eine Variante des maschinellen Lernens dar. Der Lernprozess wird in der Regel als überwachtes Lernen realisiert, d.h. die Anpassung der Parameter wird durch die Beurteilung anhand einer externen Gütefunktion gesteuert. Genetisches Programmieren ist eine der vier bekannten Hauptarten von Evolutionären Algorithmen (EA), den Genetischen Algorithmen (GA), den Evolutionsstrategien (ES), dem Evolutionären Programmieren (EP) und der Genetischen Programmierung (GP), die im Folgenden kurz umrissen werden.

3.1 Evolutionäre Algorithmen

Evolutionäre Algorithmen sind eine Klasse von Algorithmen, die Aspekte der natürlichen Evolution simulieren. Sie werden überwiegend als Optimierverfahren eingesetzt, bei dem schrittweise ein durch Angabe eines globalen Gütekriteriums definiertes Optimum angenähert wird. Es existieren vielfältige Varianten von EA, die jedoch alle bestimmte Eigenschaften gemeinsam haben.

Population Grundlage eines EA ist eine Menge von Lösungen des Problems. Eine einzelne Lösung wird in Annäherung an die Biologie Individuum bzw. Genotyp genannt. Die Population ist im Suchraum verteilt und realisiert eine parallele Suche nach dem Optimum.

Gütekriterium Die Individuen stellen unterschiedliche Lösungen der Problemstellung dar. Jede Lösung besitzt daher eine eigene Güte (der sog. Phänotyp). Die Population kann anhand dieser Gütemaße geordnet werden.

Selektion Eine gemäß dem Gütekriterium geordnete Population erlaubt es, bessere von schlechteren Individuen zu unterscheiden. Schlechtere Individuen werden aus der Population entfernt und durch Varianten der besseren Individuen ersetzt.

Variationsoperatoren Die Variation[1] der besseren Individuen erfolgt durch Operatoren, die analog zu den natürlichen Vorbildern über punktuelle Veränderung (Mutation) bzw. über massiven Austausch von genotypischer Information (Rekombination) funktionieren. Über Frequenz und Grad der Wirkung der Variationsoperatoren lässt sich die Suche im Suchraum kontrollieren.

Algorithmus 3.1.1 stellt einen typischen EA in Pseudocode dar. Eine Population von Individuen (Datenstrukturen) wird initialisiert und im Verlaufe der Evolution durch wiederholte Anwendung von Auswertung, Selektion, Rekombination und Mutation im Sinne der Fitnessfunktion optimiert. Die Populationsgröße bleibt hierbei in der Regel konstant. Die Variable t misst den Fortschritt der Zeit in Generationen.

Die Initialisierung der Population erfolgt im Allgemeinen zufällig, es ist jedoch möglich, dem EA durch eine spezielle Initialisierung Informationen über die Problemstellung zur Verfügung zu stellen. In den Kapiteln 8 und 9 wird von dieser Möglichkeit zur Beschleunigung der Evolution Gebrauch gemacht. Während der Evolution wird die Fitness eines Individuums anhand eines Gütekriteriums berechnet. Die Auswertung kann sowohl sehr einfach durch das Berechnen einer mathematischen Fitnessfunktion erfolgen, die Fitness kann jedoch auch durch Ausführung einer komplexen Simulation oder durch manuelle Bewertung gewonnen werden.

Die Selektion erfolgt meist zweistufig. Zuerst werden die Eltern und die Anzahl der Nachkommen bestimmt. Durch Rekombination der Eltern, wobei genetische Information vermischt, und anschließende Mutation, bei der die genetische Information zusätzlich verrauscht wird, werden die Nachkommen erzeugt. Die Güte der Nachkommen wird durch Auswertung berechnet und im folgenden zweiten Selektionsschritt werden diejenigen Individuen bestimmt, die die nächste Generation bilden.

Vier Hauptrichtungen der EA sind zu unterscheiden, wobei es bei der Vielfalt der Ausprägungen auch zu Überschneidungen bzw. hybriden Algorithmen kommt. In den folgenden Abschnitten werden die Unterschiede kurz erläutert, für eine detaillierte Darstellung sei auf [30] verwiesen.

3.1.1 Genetische Algorithmen

Genetische Algorithmen sind vermutlich die häufigste Ausprägung eines EA. Sie sind Mitte der siebziger Jahre des vorherigen Jahrhunderts von Holland [79] entwickelt worden. GA unterscheiden sich von anderen Ausprägungen durch die radikale Einfachheit, in der Lösungen für die Problemstellung repräsentiert werden: der Genotyp besteht aus einer Sequenz von Binärzahlen, die eine feste Länge besitzt. Die Variationsoperatoren für diese Struktur sind ebenso einfach, wobei bei GA das Hauptaugenmerk auf dem Rekombinationsoperator liegt [69].

Das Hauptproblem bei der Verwendung eines binären Genotyp besteht darin, eine effiziente Kodierung der Suchraumvariablen in einen Binärstring fester Länge zu finden. Der Algorithmus eines GA ist identisch mit dem allgemeinen Algorithmus 3.1.1.

[1] Variationsoperatoren werden auch *Suchoperatoren* genannt.

Algorithmus 3.1.1

```
1  t = 0;
2  /* Intialisierung der Individuen in der Population P*/
3  for i = 1 to popsize do
4      init(P_i(t));
5      evaluate(P_i(t));
6  od
7  /* Hauptschleife der Evolution */
8  while ¬terminate(); do
9          t = t + 1;
10         parents = selectParents(P(t));
11         offspring = recombine(parents);
12         mutate(offspring);
13         for j = 1 to numberOfOffspring do
14             evaluate(offspring[j]);
15         od
16         P(t + 1) = selectBest(P(t), offspring);
17 od
```

3.1.2 Evolutionsstrategien

Die Evolutionsstrategien zählen, zusammen mit EP, zu den frühesten EA. Mitte der sechziger Jahre des vorherigen Jahrhunderts von Schwefel und Rechenberg entwickelt [150, 140], wurden sie für die experimentelle Optimierung von hydrodynamischen Problemstellungen verwendet. Aus jener Zeit stammt auch die ursprünglichste Form der ES, mit nur einem Individuum in der Population, von dem durch Mutation nur ein Nachkomme erzeugt wurde. ES unterscheiden sich durch drei Besonderheiten von anderen EA [151]: (i) Die Repräsentation der Lösungen. Diese erfolgt durch reellwertige Vektoren fester Länge. (ii) Die Selbst-Adaptation der Strategieparameter. Mutationsschrittweiten beispielsweise können als Teil des Genoms selbst dem evolutionären Prozess unterworfen sein, wodurch Individuen mit besseren Lösungen und gleichzeitig besseren Strategieparametern für die Evolution bevorzugt werden. (iii) Die $(\mu+, \lambda)$-Selektion. Diese Art der Selektion wird auch Selektion durch Überproduktion genannt. Aus einer Menge von λ Nachkommen der μ Individuen der Population (wobei λ ein Vielfaches der Populationsgröße μ sein kann) wird die Nachfolgepopulation gebildet.

Es existieren zwei Varianten der Selektion. In der ersten Variante werden die Individuen der vorherigen Generation immer ersetzt. Bei dieser sogenannten Komma-Strategie wird Zeile 16 in Algorithmus 3.1.1 durch die entsprechende Zeile in Algorithmus 3.1.2 ersetzt. Hierdurch wird eine permanente Bewegung der Population im Suchraum erreicht, da die Elterngeneration komplett durch die μ besten Nachkommen ersetzt wird. Bei der zweiten Variante, der Plus-Strategie, werden aus den Individuen der vorherigen Generation

Algorithmus 3.1.2

```
...
/* Hauptschleife der Evolution, Komma-Strategie */
while ¬terminate(); do
    ...
    P(t + 1) = selectBest(offspring);
od
```

(den Eltern) und den Nachkommen die besten Individuen für die nächste Generation selektiert. Zeile 16 in Algorithmus 3.1.1 wird in diesem Fall durch die entsprechende Zeile in Algorithmus 3.1.3 ersetzt. Auf diese Weise verbleiben gute Lösungen in der Population, bis bessere gefunden werden. Mit einem weite-

ren Parameter, κ, kann die „Lebensdauer“ der Individuen bei dieser Methode begrenzt werden, um eine Stagnation zu vermeiden.

Algorithmus 3.1.3 ...

```
/* Hauptschleife der Evolution, Plus-Strategie */
while ¬terminate(); do
    ...
    P(t+1) = selectBest(P(t) ∪ offspring);
od
```

3.1.3 Evolutionäres Programmieren

Evolutionäres Programmieren zählt zu den frühsten Varianten eines EA und wurde Mitte der sechziger Jahre von Fogel entwickelt [63]. In EP werden Individuen als Endliche Automaten repräsentiert, d.h. als Graphen. EP verwendet ausschließlich Mutation, wobei verschiedene Mutationsoperatoren für unterschiedliche Teile des Graphen verwendet werden. Der Basisalgorithmus von EP ist in Algorithmus 3.1.4 als Pseudocode dargestellt

Algorithmus 3.1.4 ...

```
/* Hauptschleife der Evolution, EP */
while ¬terminate(); do
    t = t + 1;
    offspring = selectParents(P(t));
    mutate(offspring);
    for j = 1 to numberOfOffspring do
        evaluate(offspring[j]);
    od
    P(t+1) = selectBest(P(t) ∪ offspring);
od
```

Für die Fitnessbewertung werden die Endlichen Automaten ausgeführt, wobei ausgehend von aktuellem Zustand und aktueller Eingabe sequenziell neue Zustände angenommen werden und Ausgaben in Form von Symbolen geschrieben werden. Die Ausgabe ist also eine Symbolfolge. In zahlreichen Anwendungen von EP ist die Fitness eines Individuums deswegen auch als der Abstand der generierten Symbolfolge zu einer bekannten Folge definiert worden. EP ist relativ gering verbreitet.

3.2 Repräsentation von Programmen in GP

Genetische Programmierung ist die jüngste der vier Hauptarten von EA und wurde Anfang der neunziger Jahre etabliert [108]. GP unterscheidet sich im Wesentlichen in der Repräsentation von anderen EA-Spielarten. Wie oben erwähnt, steht bei GP die direkte Evolution von Programmen bzw. Algorithmen im Vordergrund, was unmittelbar Auswirkung auf die Kodierung des Problems hat. Durch die Tatsache, dass Programme bzw. Rechenvorschriften auf verschiedene Art und Weise repräsentiert werden können, werden innerhalb von GP allerdings auch verschiedene Formen der Kodierung verwendet. Der Basisalgorithmus bei GP ist, wie auch bei den GA, identisch mit dem Grundalgorithmus 3.1.1.

Baum-GP

Die ursprünglich eingeführte Repräsentation von Programmen durch Bäume [105, 106, 108] ist die am weitesten verbreitete. Bäume, deren innere Knoten Operatoren, äußere Knoten (Blätter) Konstanten darstellen, können sehr einfach in Post- oder Präfixausdrücke (wie z.B. LISP-Ausdrücke) umgewandelt werden. Ein Beispiel eines einfachen Baumausdruckes ist in Abbildung 3.1 dargestellt.

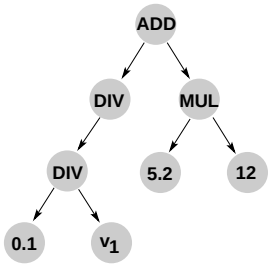


Abbildung 3.1: Ausschnitt aus einem GP-Individuum mit Baumstruktur. Operatoren stehen auf inneren Knoten, Konstanten oder Variablen (beispielsweise für die Eingabe von Problemvariablen) bilden die Blätter des Baumes.

Bäume werden also aus der Kombination von Elementen zweier Mengen gebildet, der Menge der Operatoren (*function set*) und der Menge der Konstanten (*terminal set*). Bei der Rekombination werden Teilbäume aus den beteiligten Bäumen extrahiert und ausgetauscht. Bei der Mutation werden Knoten gelöscht oder eingefügt. Es kann auch der Knotentyp verändert werden, was eventuell Einfluss auf die Anzahl der Nachfolgeknoten im Baum hat. Werden Knoten eingefügt, die Nachfolgeknoten besitzen (beispielsweise Operatoren), so werden diese ebenfalls in den bestehenden Baum eingefügt.

Linear-GP

Die lineare Repräsentation von Programmen [30, 129] orientiert sich an der linearen Struktur von Assembler- bzw. Maschinenspracheprogrammen. Ein Beispiel eines linearen GP-Programmes ist in Abbildung 3.2 dargestellt. Zusätzlich zu den bei Baum-GP bereits eingeführten Operator- und Terminalmengen wird bei

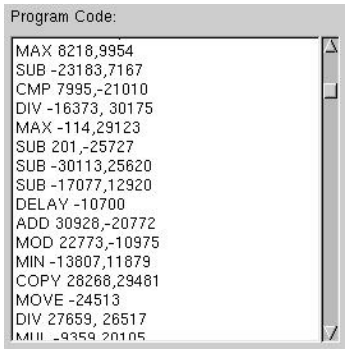


Abbildung 3.2: Ausschnitt aus einem GP-Individuum mit linearer Struktur, wie sie im System SIGEL (Kapitel 6.2) verwendet werden.

Linear-GP eine Menge von Speicherplätzen benötigt. Diese Register dienen zur Speicherung von Zwischenergebnissen, der Versorgung des Programms mit Eingabewerten und dem Speichern des Gesamtergebnisses.

Eine Erweiterung der Wirkung von Befehlen wird durch die Verwendung von Registern für die Speicherung von Ergebnissen erreicht. Anders als bei Baum-GP, bei dem nur lokale Wirkung möglich ist, erstreckt sich die Wirkung hier auch auf Befehle, die vom Ort der Entstehung weiter als eine Position entfernt sind. Diese globalen Variablen und durch Sprungbefehle bedingte variable Ausführungsreihenfolgen erhöhen die Komplexität der Programme bei linearem GP gegenüber reinem Baum-GP erheblich.

Graph-GP

Eine weitere Variante der Repräsentation von Programmen in GP wurde z.B. von [168] vorgestellt. Bereits die Baum- bzw. die lineare Repräsentation von Programmen sind spezielle Varianten von Graphen. Reines Graph-GP allerdings vermeidet die strukturellen Einschränkungen dieser Varianten und erlaubt beliebige (verbundene) Graphen als Repräsentation von Programmen. Ein Graph-GP-Individuum ist schematisch in Abbildung 3.3 dargestellt.

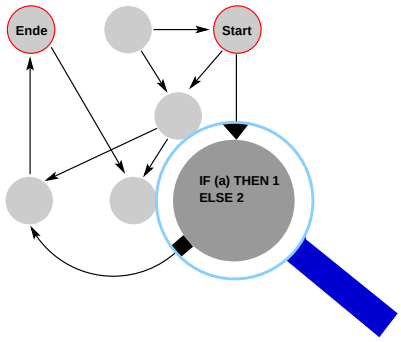


Abbildung 3.3: Ausschnitt aus einem GP-Individuum mit Graphstruktur. Codesegmente in den Knoten entscheiden über den Programmfluss.

Die Kanten im Graphen spielen hierbei die Rolle des Programmzeigers, der die nächste auszuführende Instruktion eines Programms angibt. Instruktionen sind in Knoten kodiert und werden aus den schon bekannten Funktions- und Terminalmengen gebildet.

Für die eindeutige Ausführung eines Programms werden ein definierter Start- und Endknoten benötigt. Die Ausführung wird bei Erreichen des Endknotens terminiert (oder bei Erfüllung eines anderen Terminierungskriteriums). Während der Ausführung wird in jedem Knoten (i) ein Befehl ausgeführt und (ii) der nächste Knoten für die Ausführung bestimmt. Es kann nicht im Vorhinein festgelegt werden, ob der Endknoten erreicht wird, deshab wird in der Regel nach einer bestimmten Zeit bzw. nach einer festgelegten Anzahl von Instruktionen die Ausführung des Programms abgebrochen.

Hybride Ansätze

Es existieren Arbeiten, die Eigenschaften verschiedener Repräsentationen vereinen. Hierzu zählt beispielsweise ein Ansatz, der Baum-GP verwendet, innerhalb der Knoten jedoch gekapselt lineare Strukturen ausführt [97]. Ein hybrides linear-baum-Individuum ist in Abbildung 3.4 beispielhaft dargestellt. Die Kapselung von linearen Programmstrukturen in Knoten ist prinzipiell auch für Graph-GP möglich. Für die einzelnen Teile des Individuums (die übergeordnete Baum- bzw. Graphstruktur und die lineare Sequenz von Instruktionen innerhalb von Knoten) werden dann unterschiedliche Variationsoperatoren verwendet, so dass die Komplexität des Algorithmus durch die Repräsentation gegenüber dem Basisalgorithmus deutlich erhöht wird.

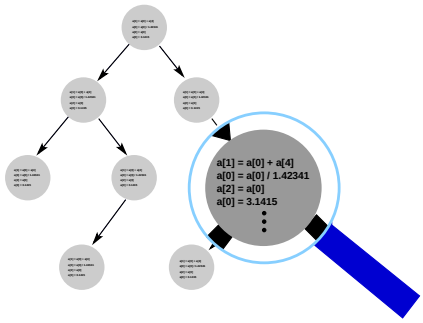


Abbildung 3.4: Ein GP-Individuum mit hybrider Struktur. Lineare Codesegmente werden in den Knoten den Baumes ausgeführt.

3.3 Variationsoperatoren in GP

Das Spektrum der möglichen Ausprägungen der Individuen wird erheblich durch die Tatsache erweitert, dass GP nicht nur die direkte Evolution von Computerprogrammen, sondern auch die Evolution beliebiger, als Programm, Algorithmus oder Rechenvorschrift interpretierbarer Strukturen umfasst. In Abhängigkeit von der gewählten Repräsentation werden dann die auf den jeweiligen Datenstrukturen wirkenden Variationsoperatoren entworfen. Man unterscheidet jedoch in allen Fällen zwischen Mutations- und Rekombinationsoperatoren, wobei die Ersteren Veränderungen in nur einem Individuum der Population bewirken, Letztere aus zwei oder mehr Individuen neue Individuen erzeugen. Die Arbeitsweise der Operatoren ist dabei so vielfältig wie die Menge der möglichen Repräsentationen in GP.

Die Variationsoperatoren verändern die Individuen und erzeugen auf diese Weise eine Bewegung der Population im Suchraum. Manchmal wird auch der Reproduktionsoperator zu den Variationsoperatoren gezählt. Dieser erzeugt identische Kopien von Individuen und wird an dieser Stelle allerdings nicht näher erläutert. Die Wahrscheinlichkeiten der Anwendung des Mutations- und Crossoveroperators sind wichtige Parameter eines GP-Systems, die starke Auswirkung auf die Performance des Systems haben können. Sie werden vor jedem Lauf festgelegt. In Abschnitt 7.3 wird der Einfluss der Operatorwahrscheinlichkeiten auf die Evolution mit GP detailliert untersucht.

Rekombination

Bei Rekombinations- bzw. Crossoveroperatoren werden Teilmengen der Struktur der Individuen ausgetauscht. Je nach Repräsentation können das Teilbäume bzw. Knoten (Baumrepräsentation), Sequenzen von Instruktionen (lineare Repräsentation) oder Teilgraphen (Graphrepräsentation) sein. Die Anzahl der ausgetauschten Teilmengen, die Größe der Teilmengen, die Schnittpunkte bei den Individuen und selbst die Anzahl der beteiligten Individuen sind Parameter, die bei der Implementierung bzw. bei der Anwendung des Operators angegeben werden müssen.

Die Rekombination von genetischer Information bedeutet in der Mehrzahl aller Fälle eine Verschlechterung der Fitnesswerte der Nachkommen gegenüber den Eltern [30]. Um das Potenzial der Rekombination, das vor Allem in der Exploration des Suchraumes liegt, trotzdem weiter ausschöpfen zu können, ist der Rekombinationsoperator weiter entwickelt worden. Ein Beispiel für die Erweiterung der Rekombination ist *homologes Crossover*. Hierbei wird im Gegensatz zur Standardrekombination, bei der die ausgetauschten Teilmengen an genetischer Information rein zufällig bestimmt werden, darauf geachtet, strukturell oder funktional ähnliche Teile des Genoms auszutauschen, was auch biologisch plausibler ist [30].

Mutation

Mutationsoperatoren wirken nur auf ein Individuum. Oft werden Mutationsoperatoren nach dem Rekombinationsoperator angewendet. Viele EA verwenden auch nur die Mutation als Suchoperator. Die Wirkweise einer Mutation ist, wie beim Rekombinationsoperator, abhängig von der gewählten Repräsentation. Beim Baum- und Graph-GP werden einzelne Knoten bzw. Kanten der Individuen mutiert, d.h. verändert, bei Linear-GP sind es Instruktionen, die verändert werden. Gegebenenfalls muss noch weiter differenziert werden: Beim Linear-GP beispielsweise besteht eine Instruktion aus mehreren Bestandteilen, die entweder einzeln oder insgesamt Gegenstand der Mutation sein können.

3.4 Selektion in GP

Nachdem die Population bewertet worden ist, wird mit Hilfe der Selektion bestimmt, welche Individuen in der Population verbleiben, welche Individuen Gegenstand von Rekombination bzw. Mutation sind und welche Individuen aus der Population entfernt werden. Durch ein geeignetes Verhältnis der Anzahl von verbleibenden und von zu ersetzenden Individuen wird ein skalierbarer Selektionsdruck in Richtung Optimum erzeugt. Die gebräuchlichsten Selektionsverfahren sind fitness-proportionale Selektion (*fitness-proportional selection*), rangbasierte Selektion (*ranking selection*) und $(\mu+, \lambda)$-*Selektion*.

Fitness-Proportionale Selektion

Die Individuen werden mit einer Wahrscheinlichkeit p_i selektiert, die proportional zur Fitness in Relation zur Fitness der Gesamtpopulation ist. Es gilt dann

$$p_i = f_i \frac{1}{\sum_{i \in P} f_i}. \tag{3.1}$$

Nachteil dieser Methode ist, dass Individuen mit sehr guter Fitness im Vergleich zu den anderen Individuen der Population sehr häufig selektiert werden, was zwangsläufig zu einem starken und schnellen Abfall der Diversität innerhalb der Population und zu einem Fokussieren der Suche auf einen begrenzten Ausschnitt des Suchraumes führt. Separate Maßnahmen zur Diversitätserhaltung können dem entgegenwirken.

Rangbasierte Selektion

Die rangbasierte Selektion wählt ein Individuum mit einer Wahrscheinlichkeit, die nicht auf der tatsächlichen Fitness, sondern auf der Ordnung der Individuen basiert. Für die Berechnung von p_i wird die Position r_i innerhalb der nach Fitnesswerten geordneten Population und eine beliebige Funktion g verwendet. Für p_i gilt

$$p_i = g(r_i). \tag{3.2}$$

Besonders häufig wird für g eine lineare oder eine Exponentialfunktion verwendet. Mit der rangbasierten Selektion kann beispielsweise die Dominanz einer guten Lösung vermieden werden, wie sie bei der fitness-proportionalen Selektion auftritt.

$(\mu+, \lambda)$-Selektion

Diese Form der Selektion stammt aus dem Gebiet der Evolutionsstrategien. Aus einer Anzahl von μ Eltern werden λ Nachkommen generiert. Je nach Strategie bilden die μ besten Individuen aus der gemeinsamen

Menge der Eltern und Nachkommen (Plus-Strategie), bzw. nur aus den Nachkommen (Komma-Strategie), die nächste Generation. Über das Verhältnis von μ und λ kann der Selektionsdruck variiert werden. Auch die $(\mu+, \lambda)$-Selektion ist, wie die rangbasierte Selektion, unabhängig von den tatsächlichen Fitnesswerten der Individuen.

Steady-State Selektion

Generell wird zwischen generationsbasierter Selektion, bei der die Population der nächsten Generation in einem Schritt aus der vollständig bewerteten aktuellen Generation gebildet wird, und den *steady-state* Mechanismen unterschieden, bei der Teile der aktuellen Population ersetzt werden und im nächsten Selektionsschritt sofort zur Verfügung stehen. Der Generationsbegriff kann dann nicht mehr im wörtlichen Sinne verwendet werden und wird mathematisch bestimmt[2].

Ein Beispiel für einen steady-state-Algorithmus ist die Turnierselektion. Hierbei werden nur Teilmengen der Population miteinander verglichen, wobei die „Gewinner“ des Turniers, d.h. diejenigen Individuen mit der besseren Fitness, die „Verlierer“ ersetzen. Soll Rekombination stattfinden, ist die minimale Turniergröße vier, wird nur Mutation verwendet, müssen mindestens zwei Individuen am Turnier teilnehmen. Über die Größe des Turnieres kann der Selektionsdruck variiert werden. Kleine Turniergrößen verursachen hierbei einen geringeren Selektionsdruck als größere Turniere, da bei kleinen Turnieren die Wahrscheinlichkeit eines Individuums, ausschließlich mit schlechteren Individuen verglichen zu werden, größer ist, als bei umfangreicheren Turnieren (in denen der „Champion“ der Population eher auftauchen kann).

In den folgenden Kapiteln wird GP mit linearer Repräsentation und als Selektionsmethode durchgehend Turnierselektion verwendet. Lineares GP hat den Vorteil, besonders effizient implementierbar zu sein und hat sich in früheren Arbeiten als gut geeignet für Problemstellungen aus dem Gebiet der autonomen Roboter erwiesen [131, 132].

[2] Wenn eine Anzahl von Individuen ausgewertet worden ist, die der Populationsgröße entspricht, wird die Generationenzahl erhöht. Es kann dabei vorkommen, dass einige Individuen der Population nicht bewertet worden sind, andere hingegen mehrmals.

Teil II

Evolution von Robotersteuerungen mit Genetischer Programmierung

Kapitel 4

Evolution und Robotik

Die Steuerung autonomer Roboter ist eine komplexe Aufgabe. Abhängig von der Ausstattung des Roboters mit Aktorik und Sensorik und nicht zuletzt von der zur Verfügung stehenden Prozessorleistung muss ein Kontrollprogramm entworfen und implementiert werden, das allen aufgabenspezifischen Anforderungen genügt, die an den Roboter gestellt werden. Gerade bei autonomen mobilen Robotern ist der Anspruch an die Robotersteuerung besonders hoch, da eine flexible und anpassungsfähige Steuerung benötigt wird, um auch in unbekannten Situationen ein Funktionieren des Roboters gewährleisten zu können. Hierfür dürfen keine starren Kontrollalgorithmen verwendet werden, die Aufgrund der prinzipiellen Unvorhersehbarkeit von Umweltsituationen und vom aktuell vorherrschenden Zustand des Roboters immer unzureichend bleiben müssen. Die Erhöhung des Grades an Kontrollautonomie in mobilen Robotern ist Gegenstand aktueller Forschung weltweit.

Die klassische deliberative Architektur autonomer mobiler Roboter basiert auf einer weitgehend sequenziellen Reihenfolge der Informationsverarbeitung in hierarchischen Kontrollsystemen (siehe Abbildung 4.1, links). Dieser Aufbau ist gut geeignet für strukturierte und bekannte Umgebungen, scheitert aber, wenn der Roboter in unbekannten und dynamischen Situationen schnell und geeignet reagieren soll. Ein nach diesem Prinzip gesteuerter Roboter, der Handlungen nach einem internen Weltmodell plant, ist vom Funktionieren dieses Weltmodells bereits zu Beginn des Einsatzes abhängig. Diese Information muss also a priori bekannt sein.

Der Nachteil, dass vor dem Betrieb des Roboters bereits ein Modell der Umwelt bzw. ein Konzept der Handlungsplanung existieren muss und die Notwendigkeit zur „Reaktion“ auf schnelle Veränderungen in der Umwelt haben zur Abkehr von diesem Modell geführt. Das Studium lebender Vorbilder aus der Biologie macht deutlich, dass bereits simple neuronale Strukturen oder sogar rein chemische Kontrollsysteme es primitiven Kleinstlebewesen erlauben, autonom zu agieren und dabei rudimentäre Intelligenz zu zeigen. Die Analyse der neuronalen bzw. chemischen Informationsverarbeitung in diesen Lebewesen hat zur verhaltensbasierten Robotik (*behavior based robotics*) geführt.

Die Architektur der Kontrollsoftware für verhaltensbasiert gesteuerte autonome Roboter spiegelt die rein reaktiven Verhaltensmuster primitiver Lebewesen wider (siehe Abbildung 4.1, Mitte). Die einzelnen Verhaltensmodule operieren unabhängig voneinander, so dass auch bei einem Versagen eines Moduls der Roboter nicht vollständig ausfällt. Ein Roboter kann also seine Umgebung erkunden, um eine Karte zu erstellen, die zur Navigation zu einem späteren Zeitpunkt verwendet wird, und gleichzeitig etwaigen Hindernissen ausweichen oder dynamischen Veränderungen der Umwelt durch eine bereits funktionierende primitive Verhaltenssteuerung begegnen. Das Gesamtverhalten des Roboters wird erzeugt durch die parallele Ausführung der Module. Diese verhaltensbasierte, reaktive Architektur wird gleichzeitig in reale Roboter eingebettet (*embodiment*), wodurch Anpassungen an Artefakte der künstlichen, simulierten Umgebungen vermieden werden sollen [40, 41]. Es ist mit dieser Technik zwar prinzipiell möglich, durch Kombination einfacher

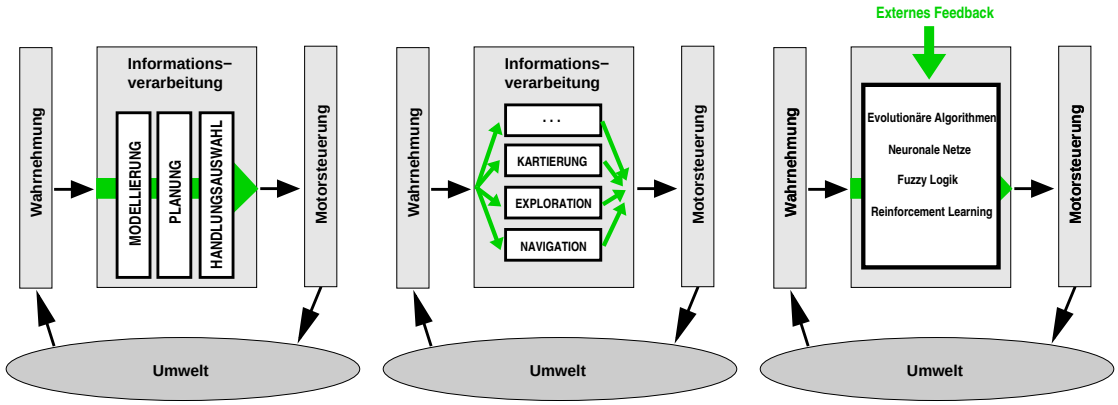


Abbildung 4.1: **Links:** Klassische Architektur der Kontrollsoftware **Mitte:** Aufbau eines reaktiven Kontrollsystems. **Rechts:** Automatisches Lernen von Kontrollsystemen. Über externes Feedback wird der Lernprozess gesteuert (überwachtes Lernen).

Regeln vielschichtige Verhaltensweisen zu erzeugen (ein klassisches Beispiel hierfür sind Braitenbergs „Vehikel“ [35] oder Brooks’ „subsumption architecture“ [39]), größter Nachteil dieser Methode ist allerdings, dass es nicht möglich ist, ein Handlungsschema zu formulieren, dessen Ausführung nicht in irgendeiner Form reaktiv ist. Alle Verhaltensebenen, die zur Ausführung eine Planung von Handlungen benötigen, sind mit diesem Ansatz nicht realisierbar.

Ob die Kontrollsoftware nun auf einem klassischen Ansatz beruht oder eine verhaltensbasierte Software den Roboter steuert, allen Ansätzen gemein ist, dass die Ableitung von Motorkommandos aus der sensorisch erfassten Umwelt (und eventuell aus internen Variablen) einem manuell entwickelten Schema folgt. Die Entwickler der Kontrollsoftware müssen also Handlungen für jede beliebige Situation vorsehen, beziehungsweise, da ja ein Höchstmaß an Autonomie für den Roboter erreicht werden soll, Strategien implementieren, die es dem Roboter erlauben, zu jedem Zeitpunkt eine geeignete Handlung (im besten Falle natürlich eine optimal zur Situation passende Handlung) auszuwählen und auszuführen.

Um dieses Dilemma zu umgehen, wurde Anfang der neunziger Jahre vorgeschlagen, die Kontrollstrukturen von autonomen Robotern nicht mehr per Hand zu kodieren, sondern in einem evolutionären Prozess automatisch zu generieren (Abbildung 4.1, rechts). Der evolutionäre Prozess wird hierbei über externes Feedback gesteuert, das die Qualität der automatisch generierten Kontrollstrukturen beschreibt. Das Feedback kann hierbei automatisch extrahiert werden (z.B. durch Kameraüberwachung des Roboters und automatische Bildverarbeitung) oder auch manuell erfolgen. Hierdurch wird erreicht, dass nicht mehr die Frage der Implementierung von Kontrollstrukturen im Vordergrund steht, sondern der Schwerpunkt auf der Beschreibung von „gutem“ bzw. „schlechtem“ Verhalten liegt. Vorbild hierbei war die in der Biologie untersuchte Evolution von „intelligentem“ Verhalten, das schon bei vergleichsweise primitiven Lebewesen wie Insekten beobachtet werden kann [40].

Ansätze, die sich mit der Evolution von Kontrollstrukturen für autonome mobile Roboter befassen, werden allgemein unter dem Begriff *Evolutionary Robotics* zusammengefasst [84]. Angefangen bei Simulationen von mobilen Robotern (beispielsweise in [34, 45]) über Experimente mit realen Robotern (z.B. [91, 107, 108, 117, 128]) werden in diesem Gebiet automatisch Kontrollstrukturen mit Hilfe einer künstlichen Evolution erzeugt, die autonome mobile Roboter bestimmte Aufgabenstellungen durchführen lassen. Für die Durchführung der Experimente werden typischerweise Aufgabenstellungen aus dem Gebiet der Bahnplanung bzw. Navigation gewählt und entsprechende Gütefunktionen formuliert, anhand derer dann die Kontrollstrukturen optimiert werden. Einen detaillierten Überblick über das Gebiet Evolutionary Robotics geben z.B. [90] oder [127].

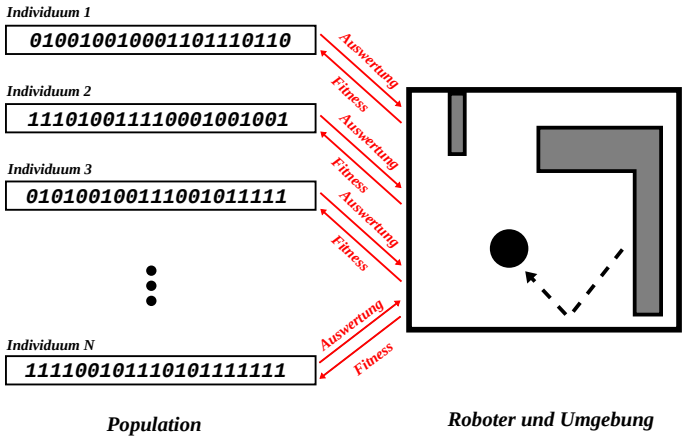


Abbildung 4.2: Evolutionary Robotics: Die Individuen (hier als Binärstrings dargestellt) werden in entsprechende Kontrollstrukturen übersetzt und steuern einen Roboter in einer definierten Umgebung. Der Roboter und die Umwelt können simuliert oder auch real sein. Die Fitness des Individuums wird durch das Verhalten des Roboters bestimmt.

4.1 Ziel der Evolution

Bei der Evolution von Kontrollprogrammen wird vorausgesetzt, dass es prinzipiell möglich ist, jedes beliebige Verhalten eines Roboters mit Hilfe der künstlichen Evolution zu generieren, wenn erstens eine hinreichend präzise Fitnessfunktion für die Bewertung der Individuen während der Evolution vorhanden ist und zweitens eine hinreichend große Anzahl von Generationen evolviert wird.

Ziel bei der Evolution von Kontrollprogrammen von autonomen mobilen Robotern ist es beispielsweise, mehr oder weniger komplexe Navigationsaufgaben zu lösen. Ein typisches Problem ist hier die Steuerung eines Roboters in einem Labyrinth von einer gegebenen Startposition zu einer definierten Endposition, wobei die Güte einer gefundenen Trajektorie durch den Grad der Erfüllung verschiedener vorher definierter Kriterien gebildet wird. Hierzu zählen beispielsweise eine möglichst geringe Anzahl von Kollisionen mit Hindernissen oder eine möglichst kurze Wegstrecke, die der Roboter zur Erfüllung der Aufgabe zurücklegt, oder auch eine Kombination beider Kriterien (siehe Abbildung 4.2). Die Randbedingungen, zu denen ein Modell des Roboters sowie eine mehr oder weniger detailliert modellierte Umwelt zählen, sind allerdings maßgeblich für das Resultat der Evolution von Kontrollprogrammen. Für simplifizierte, künstliche Umgebungen simulierter oder realer Roboter sind einfache, rein reaktive Verhaltensweisen evolviert worden. Ist die Aufgabenstellung des Roboters komplexer, sind rein reaktive Kontrollprogramme nicht geeignet und müssen um nicht-reaktive Elemente (beispielsweise eine Art „Gedächtnis“) erweitert werden, um die Aufgabenstellung hinreichend gut erfüllen zu können. Durch immer weiter steigende Komplexität der Aufgabenstellung und durch eine immer dynamischere Umwelt des Roboters sollen auf diese Weise immer komplexere Kontrollprogramme evolviert werden.

Die eigentlichen Rahmenbedingungen für die Evolution von Kontrollprogrammen werden allerdings durch die zur Verfügung stehenden Sensorinformationen auf der einen Seite und die Möglichkeit zur Bewegung auf der anderen Seite gestellt. Ein Kontrollprogramm kann nur in dem Maße auf eine sich verändernde Umwelt reagieren, wie es die Änderungen mit entsprechenden Sensoren zeitnah erfassen und die entsprechenden Konsequenzen durch Ansteuerung der Aktoren realisieren kann. Die Schwierigkeiten bei der Simulation von Robotern werden dadurch offensichtlich: Kontrollprogramme, die in simulierten Umgebungen evolviert wurden, sind an die in der Simulation gegebenen Rahmenbedingungen angepasst. Ein Transfer der Ergebnisse auf reale Roboter ist deshalb nur bedingt möglich [91]. Durch Simulationsysteme, die die physikalischen Gegebenheiten der realen Welt modellieren, kann diese Hürde verkleinert werden, es bleibt

allerdings immer ein prinzipieller Unterschied bestehen.

So wie das Kontrollprogramm eines simulierten Roboters in einer simulierten Umwelt erfolgreich evolviert werden kann, und auch reale Roboter in Laborumgebungen mit Hilfe von evolvierten Kontrollprogrammen gesteuert werden können, so kann auch der Roboter selbst Gegenstand eines evolutionären Prozesses werden. Die Untersuchungen hierzu werden durch Erkenntnisse der Biologie motiviert: Die Zusammenhänge zwischen Form und Funktion, die bei Lebewesen offensichtlich sind, sollen auch in einer künstlichen, parallelen Evolution der Morphologie und der Kontrollstrukturen von Robotern entstehen.

Bei einem Experiment zur gemeinsamen Evolution von Morphologie und Kontrollprogrammen konnten Roboter mit einer an die simulierten physikalischen Gegebenheiten angepassten Struktur und mit einem sowohl für die eigene Struktur als auch bezüglich der Umwelt und der Interaktion mit anderen Robotern optimierten Kontrollprogramm evolviert werden [94, 153, 154, 155, 156]. Ein ähnliches System ist Framsticks [104]. Hierbei konkurrieren die „Roboter“ (in diesem Falle werden sie Creatures genannt) zusätzlich um Ressourcen.

Erst in letzter Zeit konnte das Experiment von der Simulation in die Realität verlagert werden [42]. Ein Evolutionsexperiment wurde durchgeführt, bei dem ein Rapid-Prototyping-System die in der Simulation evolvierte Struktur eines Roboters realisiert. Die in der Simulation gleichzeitig mit der Struktur der Roboter evolvierten Kontrollprogramme wurden dann auf die realen Roboter transferiert [112]. Das identische Kontrollprogramm war in der Lage, sowohl in der Simulation als auch in der Realität gleiche Bewegungen des Roboters durchzuführen. In einer anderen Arbeit wurden beispielsweise Geometrie und Gangmuster eines humanoiden Roboters erfolgreich parallel evolviert [58].

4.2 Gegenstand der Evolution

Die künstliche Evolution ist in der Lage, jede Kontrollarchitektur, die bestimmte Bedingungen erfüllt, hinsichtlich einer Fitnessfunktion zu optimieren. Minimale Bedingung ist, dass ein Kontrollprogramm für einen autonomen Roboter in einer Datenstruktur repräsentiert werden kann, die es dem evolutionären Algorithmus erlaubt, die üblichen Variationsoperatoren (zumindest aber Mutation) durchzuführen. Auf diese Weise ist ein EA nicht auf eine bestimmte Form der Kontrollarchitektur beschränkt und durch die unterschiedlichen Varianten von EA ebenfalls frei bei der Repräsentation der schließlich gewählten Architektur. Zusätzlich muss die Güte einer Kontrollstruktur bestimmbar und quantifizierbar sein, um eine Rangfolge bilden zu können, die Grundlage für das Funktionieren der Selektionsmechanismen ist.

Es wurden und werden unterschiedliche Repräsentationen verwendet, die jeweils besondere Vor- bzw. auch Nachteile besitzen. Im Folgenden ist eine Auswahl von Arbeiten aufgeführt, in denen jeweils eine bestimmte Form einer Kontrollarchitektur gewählt und speziell kodiert wurde.

Kodierung der Kontrollstruktur

Die am häufigsten verwendete Kontrollstruktur ist ein künstliches neuronales Netz (eine detaillierte Einführung in das Gebiet der neuronalen Netze gibt z.B. [178]). Für rein reaktive Verhaltensmuster werden hierbei Feedforward-Netze wie beispielsweise Multilayer-Perzeptrons oder Kohonenkarten eingesetzt, die für die Berechnung der Ausgabe des Netzes (beispielsweise den Motorkommandos) ausschließlich die Eingaben verwenden. Sind die zu evolvierenden Verhaltensweisen komplexer, kommen rekurrente neuronale Netze zum Einsatz, die durch Rückkopplung von Signalen auch interne Zustände berücksichtigen. Unterliegt das Verhalten eines Roboters zusätzlich einer zeitlichen Dynamik, so kann dies durch zeitabhängige Aktivierungsfunktionen der einzelnen Neuronen realisiert werden [45, 84, 90]. Über die normalen Lernverfahren hinaus, die ein neuronales Netz an die Anforderungen der Problemstellung anpassen, können

mit Hilfe der Evolution beispielsweise die Übertragungsfunktionen, die Gewichte, die Netztopologie oder ähnliche Charakteristika der Netze variiert werden.

Eine weitere Möglichkeit, die Steuerung eines autonomen Roboters zu kodieren, sind Classifier Systems [80]. In ihrer ursprünglichen Form werden dabei die Eingangsvariablen $e_1, \ldots, e_i$ durch eine Menge von Regeln (den Klassifizierern) der Form

$$\begin{array}{ll} \text{Regel 1:} & \text{if } (pre_1(e_1,\ldots,e_i)) \text{ then } (post_1(a_1,\ldots,a_j)), \text{strength} = s_1 \\ \text{Regel 2:} & \text{if } (pre_2(e_1,\ldots,e_i)) \text{ then } (post_2(a_1,\ldots,a_j)), \text{strength} = s_2 \\ & \vdots \\ \text{Regel N:} & \text{if } (pre_N(e_1,\ldots,e_i)) \text{ then } (post_N(a_1,\ldots,a_j)), \text{strength} = s_N \end{array}$$

logisch miteinander verknüpft und die Werte der Ausgangsvariablen $a_1, \ldots, a_j$ der jeweiligen Regel entsprechend gesetzt. Jeder Regel ist ein Wert zugeordnet, der die „Stärke“ der Klassifikation beschreibt. Bei der zweiten Variante der Classifier Systems werden die Klassifizierer in Form von $if-then$-Regeln durch neuronale Netze ersetzt [122, 123].

Gegenstand der Evolution bei Classifier Systems (die in diesem Falle *learning classifier systems* genannt werden) können beispielsweise die einzelnen Parameter der Regeln bzw. ganze Regeln an sich sein. Bilden neuronale Netze die Klassifizierer, so finden die oben erwähnten Adaptionsmechanismen Anwendung.

Das Kontrollsystem eines autonomen Roboters kann neben der Repräsentation durch ein neuronales Netz oder durch ein Classifier System auch durch ein entsprechendes Steuerungsprogramm dargestellt werden. Hierfür ist per definitionem die Methode des Genetischen Programmierens sehr gut geeignet. Die Programme werden hierbei über Eingangsvariablen, die Teil der Terminalmenge sind, mit Eingaben versorgt und berechnen daraus die Werte der Ausgangsvariablen (beispielsweise die Umdrehungszahl der Motoren). Der Gegenstand der Evolution sind die Programme an sich, mit den im vorherigen Kapitel aufgeführten Variationsoperatoren.

Genotyp-Phänotyp-Mapping

Eine wichtige Rolle bei der Evolution von Kontrollstrukturen spielt die Übertragung der im Genom der Individuen gespeicherten Information auf den Roboter, d.h. um die Extraktion des Phänotyps. Dies ist deshalb von besonderer Bedeutung, weil die Fitness eines Individuums durch den Phänotyp definiert wird, Gegenstand der Variationsoperatoren und der Selektionmechanismen hingegen der Genotyp ist. Während der künstlichen Evolution werden die Mechanismen, die die Konversion zwischen Genotyp und Phänotyp bestimmen, in der Regel durch den Experimentator definiert und sind nicht - wie beispielsweise in der Natur - Gegenstand der Evolution selbst. Geschieht die Umsetzung von Genotyp zum Phänotyp während der Bestimmung der Fitness des Individuums und nicht schon vorher, so wird von der „Entwicklung“ des Individuums gesprochen.

Bei der künstlichen Evolution wird in der Regel die Konversion des Genotyps in den Phänotyp durch eine Eins-zu-Eins Umwandlung der Datenstruktur in Charakteristika des Phänotyps realisiert. Die Art der Konversion ist dabei natürlich abhängig von der gewählten Repräsentation innerhalb des EA.

Neuronale Netze

Bei neuronalen Netzen existieren mehrere Varianten der Transformation des Genotyps in den Phänotyp. Im einfachsten (und gebräuchlichsten) Fall wird ein neuronales Netz mit fester Größe durch die im Genotyp kodierten Werte parametrisiert. Jeder Parameter des Netzes hat dabei seinen festen Platz im Genom des Individuums, das also z.B. durch einen reellen Vektor dargestellt werden kann. Mit Hilfe dieser Parameter können unterschiedliche Eigenschaften des Netzes beschrieben werden, beispielsweise die Vernetzung

oder auch die Gewichte innerhalb der Neuronen. Diese Art des Genotyp-Phänotyp-Mappings wird *direct encoding* genannt. Viele Arbeiten verwenden eine GA-Repräsentation (also Binärstrings) der Gewichte künstlicher neuronaler Netze und *direct encoding*, um eine Kontrollarchitektur für die Navigation mobiler Roboter zu repräsentieren und zu evolvieren (z.B. [34, 45, 110, 128]).

Wenn die Komplexität der Problemstellung hoch oder sogar unbekannt ist, werden Netze mit variablen Eigenschaften eingesetzt. Bei diesen Netzen ist meist die Anzahl der Neuronen und/oder der Grad der Vernetzung veränderbar. Die Variabilität des Phänotyps kann nur durch ebenfalls variable Genotypen erreicht werden, d.h. mit Individuen unterschiedlicher Länge. Dies hat gleichzeitig Konsequenzen für die verwendeten Variationsoperatoren, die in diesem Falle auch Methoden zur Manipulation der Anzahl der Neuronen bzw. der synaptischen Verbindungen bereitstellen müssen. Direktes kodieren der Vernetzung (z.B. Neuron 1 wird mit Neuron 7 verbunden) ist in diesem Falle nicht möglich, da es durch die variable Größe des Netzes keine Garantie für die Existenz des adressierten Neurons gibt. Für die Umsetzung des Genotypes in den Phänotyp muss also eine kompliziertere, relative Adressierung definiert werden.

Eine elegante Methode, neuronale Netze mit variabler Topologie und Größe zu kodieren ist das sogenannte *cellular encoding* [73]. Hierbei werden Lisp-Ausdrücke in einer Baumstruktur kodiert (ähnlich der Baumrepräsentation bei GP). Diese Ausdrücke stellen *graph rewriting rules* dar. Hierbei wird eine Startstruktur des Netzes, die mit einem einzelnen Neuron die Eingangs- mit den Ausgangsknoten verbindet, durch die im Genom kodierten Transformationen (z.B. Knotenduplikationen oder das Einfügen einer rekurrenten Verbindung) sukzessiv in seine endgültige Form überführt. Durch die Ähnlichkeit der Repräsentation der Rewriting rules zu Baum-GP können die dort bekannten Variationsmechanismen verwendet werden. Cellular encoding ist in der Lage, neuronale Netze mit beliebigen Strukturen zu erzeugen und ist bereits erfolgreich für die Evolution von Laufrobotersteuerungen eingesetzt worden [73, 90].

Classifier Systems

Das Mapping vom Genotyp zum Phänotyp bei Classifier Systems hängt natürlich davon ab, ob die Klassifizierer regelbasiert oder durch neuronale Netze realisiert werden. In der regelbasierten Darstellung besteht ein Individuum aus mindestens zwei Teilen (den sog. Chromosomen), die jeweils die Eingabe bzw. die Ausgabe der Regel darstellen [54]. Üblicherweise werden binäre Ein- und Ausgänge verwendet, so dass eine Regel durch einen Binärstring dargestellt werden kann:

$$I = \underbrace{\{0,1,*,1,}_{condition}\underbrace{1,1,0\}}_{action}.$$

Eine Regel wird dann ausgeführt, d.h. die Ausgänge des Classifier Systems werden entsprechend des *action* Teiles der Regel gesetzt, wenn die Werte der Eingänge des Classifier Systems mit der *condition* der entsprechenden Regel übereinstimmen, wobei das „*“-Symbol bedeutet, dass ein Eingang irrelevant ist (*don't care*-Symbol).

Werden die Classifier durch neuronale Netze repräsentiert, so wird das Genotyp-Phänotyp-Mapping mit den oben erwähnten Techniken durchgeführt. Oft wird hierbei die gesamte Regelbasis des klassischen Ansatzes in einem Netz kodiert [122, 123]. Neuronale Netze bieten zusätzlich den Vorteil, sowohl reellwertige Eingaben zu akzeptieren als auch reelle Ausgaben produzieren zu können. Auch Classifier Systems sind erfolgreich eingesetzt worden, um autonome mobile Roboter zu steuern [46, 122, 149].

Genetisches Programmieren

Bei GP wird die Art des Genotyp-Phänotyp-Mappings durch die gewählte Repräsentation bestimmt (siehe Kapitel 3). Bei den in der Vergangenheit durchgeführten Untersuchungen zur Evolution von Kontrollprogrammen für autonome mobile Roboter fanden nahezu alle bekannten Arten der Darstellung von Programmen Anwendung: In den Arbeiten von Koza und Rice werden beispielsweise Baumstrukturen verwendet,

die Lisp-Ausdrücke darstellen [108, 107]. Das Genotyp-Phänotyp-Mapping, das bei den oben aufgeführten anderen Methoden aufwändige Berechnungen benötigt, wird in diesem Fall durch simples Auswerten des Lisp-Ausdruckes erreicht.

Andere Arbeiten mit GP verwenden statt dessen lineare Repräsentationen, die eher einem Assembler-Programm ähneln [130, 131, 132, 133, 134]. Auch bei diesen Ansätzen werden Programme direkt kodiert, d.h. sie werden in einem Interpreter ausgeführt und wirken direkt auf den Roboter.

4.3 Evolution von Laufalgorithmen

Bei Experimenten mit Laufrobotern ist der Gegenstand der Evolution meist der Laufalgorithmus zur koordinierten Bewegung der Beine des jeweiligen Roboters. Höhere Kontrollebenen wie z.B. die Navigation, die Bahnplanung oder auch die Task-Koordination werden für Laufroboter in der Regel nicht untersucht, da hierfür auf Resultate anderer Arbeiten zurückgegriffen werden kann, die sich mit den entsprechenden Fragestellungen beschäftigen.

Im Bereich der Evolution von Laufrobotersteuerungen existieren neben den Arbeiten, die manuell implementierte neuronale Netze verwenden (z.B. [32, 47, 64]), auch Ansätze, bei denen Parameter neuronaler Netze evolviert werden [44, 65, 110, 111]. In anderen Arbeiten wird der Programmcode direkt mit GP-Systemen evolviert (z.B. [82, 135, 161, 177], um nur Einige zu nennen), wobei die Programme aus zusammengesetzten Befehlen (sogenannten *Makrobefehlen*) bestehen, die beispielsweise komplette Beinbewegungen kodieren. Die resultierenden Programme können Aufgrund dessen leicht evolviert und analysiert werden.

Es existieren darüber hinaus weitere Ansätze, die die Kontrollstrukturen für autonome Laufroboter mit Hilfe anderer Methoden des Maschinellen Lernens erzeugen. Hierzu zählen unter anderem Varianten, die Methoden des Reinforcement Learning verwenden um Beinbewegungen zu koordinieren (beispielsweise [136, 164, 167]). Andere verwenden Fuzzy-Methoden kombiniert mit evolutionären Algorithmen ([137]) oder Neuro-Fuzzy Ansätze ([116, 172]). Learning Classifier Systems werden ebenfalls für die Evolution von Laufrobotersteuerungen eingesetzt [43].

Allein durch die Anzahl der unterschiedlichen Ansätze wird deutlich, dass ein breites Spektrum an Kontrollarchitekturen und Repräsentationen untersucht wurde, wobei bei nahezu allen Ansätzen, die GP verwendet haben, Makrobefehle in den Operatormengen enthalten waren. Dabei wird jeweils vorausgesetzt, dass Routinen existieren, die die jeweiligen Makrobefehle in Gelenkbewegungen umsetzen, wofür genaue Kenntnisse über die kinematische Struktur des Roboters notwendig sind. Ein Beispiel ist die Verwendung eines *Swing*-Befehls, der ein Bein aus der hinteren Aufstandsposition nach vorne schwingt und wieder aufsetzt. Hierfür muss eine entsprechende Trajektorie für den Fußpunkt generiert werden, entlang der der Fußpunkt bewegt werden muss. Es sind inverse Transformationen an Zwischenpunkten auf der Trajektorie notwendig, um aus der Sollposition des Fußpunktes im Inertialkoordinatensystem die erforderlichen Gelenkwinkel des Beines abzuleiten. Diese aufwändigen Berechnungen, die die hohe Komplexität der Laufmustergenerierung verdeutlichen, werden von den allermeisten Ansätzen jedoch als gegeben vorausgesetzt, so dass dort eigentlich nur die zeitliche Koordination bereits bekannter Bewegungsabläufe evolviert wird.

Hinzu kommt, dass nahezu alle gezeigten Ansätze nur auf einen Roboter bzw. eine Roboterarchitektur angewendet worden sind, so dass keine verallgemeinerten Schlussfolgerungen aus den Resultaten gezogen werden können.

In Teil III dieser Arbeit werden Experimente vorgestellt, die weder auf Makrobefehlen beruhen noch sich mit nur einer Roboterarchitektur beschränken. Durch die Verwendung einer linearen Repräsentation von Programmen innerhalb des GP-Systems und durch die Konzentration auf rudimentäre Befehle zur Datenoperation und zur Robotersteuerung wird die Kreativität der Evolution möglichst wenig eingeschränkt. Dieser reduktionistische Ansatz wird konsequent weiterverfolgt, indem zusätzlich auf die Integration von

Modellwissen verzichtet wird, um keinerlei Einschränkungen hinsichtlich der möglichen Roboterarchitekturen aufzuwerfen. Modellwissen umfasst an dieser Stelle sowohl die kinematischen und geometrischen Eigenschaften des Roboters als auch Erkenntnisse über Eigenschaften des Laufens an sich. Es wird gezeigt, dass mit diesem Ansatz Programme für die unterschiedlichsten Laufroboter evolviert werden können.

Kapitel 5

Fortbewegung autonomer Roboter

Die Aufgabenstellungen autonomer mobiler Roboter basieren fast immer auf der Möglichkeit der Roboter zur Fortbewegung. Die große Mehrheit aller mobilen Roboter verwendet hierfür radgebundene Antriebe, es existieren jedoch auch gehende, schwimmende, fliegende und sogar in der Schwerelosigkeit schwebende Roboter. Unabhängig von der verwendeten Fortbewegungsart sind für mobile Roboter zwei Problemstellungen interessant:

1. Wie bewegt sich der Roboter bei gegebener Ansteuerung der Antriebe? Dieses Problem ist als Vorwärtskinematik bekannt.

2. Wie müssen die Antriebe angesteuert werden um eine gegebene Bewegung zu erreichen? Dieses Problem ist wird als Inverse Kinematik bezeichnet.

Die für einen mobilen Roboter verwendete Fortbewegungsart ist vom Einsatzgebiet abhängig. Es existieren vier Kategorien von Einsatzgebieten, die spezielle Fortbewegungsarten implizieren (nach [55]):

Bodengebunden Bodengebundene Roboter bewegen sich auf festem Grund fort. Der Vortrieb wird hierbei durch reibschlüssige Kraftübertragung gewonnen. Bei ununterbrochenem Bodenkontakt werden Radantriebe verwendet. Ist Bodenkontakt nur punktuell vorhanden, werden Kettenantriebe (bei kleinen Abständen zwischen Kontaktpunkten) oder Laufantriebe (bei größeren Abständen) verwendet. Die Mehrzahl aller Roboter ist bodengebunden.

Wasser Wassergebundene Roboter operieren entweder an der Oberfläche oder unter Wasser. Der Vortrieb wird hauptsächlich durch Schrauben- oder Strahlantriebe gewährleistet. Einsatzgebiete sind beispielsweise die Exploration von schwer zugänglichen Unterwasseregionen oder die Inspektion von Schwimmkörpern (Schiffen, Ölplattformen) oder Tanks während des Betriebes (siehe z.B. [120, 148]). Schwimmende Roboter sind meist nicht vollständig autonom, weil sie über eine flexible Verbindung mit Energie oder Druckluft versorgt werden. Diese Verbindung bietet den zusätzlichen Vorteil, den Roboter in schwierigen Situationen auch aus der Entfernung steuern zu können (Teleoperation).

Luft Luftgebundene Roboter sind in der Regel auf existierenden Fluggeräten (z.B. Helikoptern, Flugzeugen, Blimps oder Ballons) aufgebaut. Die Dynamik des Fliegens macht die Navigation dieser Roboter extrem komplex [146]. Militärische Anwendungen haben zu großem Fortschritt auf dem Gebiet in den letzten Jahren geführt. Gemeinsame Eigenschaft von Flugrobotern und Unterwasserrobotern ist, dass zur Aufrechterhaltung des aktuellen Zustandes Energie aufgewendet werden muss.

Schwerelosigkeit Diese Kategorie von Robotern wird der Vollständigkeit halber aufgeführt, zur Zeit existieren nur Prototypen von Robotern dieser Kategorie. In der Schwerelosigkeit operierende Roboter sollen nach Planungen der NASA[1] beispielsweise für die Wartung oder auch innerhalb der Internationalen Raumstation ISS eingesetzt werden [66].

Für nahezu alle Fortbewegungsarten autonomer mobiler Roboter in den dargestellten Einsatzgebieten wird eine Bewegung des Roboters durch simples Einschalten des Antriebs erzeugt. Bis auf eine Ausnahme: Laufroboter stellen Systeme dar, bei denen die Bewegung des Roboters nur durch eine zeitlich koordinierte Ansteuerung aller Motoren möglich ist. Erfolgt die Bewegung der Antriebe unkoordiniert, wird ein Laufroboter sofort versagen. Diese Tatsache hat dazu geführt, dass bei Laufrobotern die Fortbewegung selbst Gegenstand der Forschung ist, wohingegen bei Robotern mit anderen Arten von Antrieben die Fortbewegung als hinreichend untersucht gilt und selbstverständlich verwendet wird[2]. Ist ein Laufroboter allerdings in der Lage, sich kontrolliert fortzubewegen, d.h. die Lösungen zu den oben erwähnten Problemen der Vorwärts- und der Inversen Kinematik können als bekannt vorausgesetzt werden, so kann ein Laufroboter wie jeder andere bodengebundene mobile Roboter verwendet werden.

Die für die Bewegung der Laufroboter notwendige Koordination der Gelenke steht also im Mittelpunkt der folgenden Kapitel dieser Arbeit.

5.1 Laufen

Laufroboter stellen einen Spezialfall bodengebundener mobiler Roboter dar. Während bei der Forschung mit Robotern mit Rädern oder Kettenantrieben die Navigation der Roboter im Vordergrund steht, d.h. die kontrollierte Bewegung des Gesamtsystems in der Umwelt, liegt bei Laufrobotern das Hauptaugenmerk auf der Fortbewegung an sich. Fragen nach der Struktur des Laufens, nach der Regelung und Steuerung und den damit verbundenen Problemstellungen auf den Gebieten der Sensorik bzw. Aktorik, nach der Dynamik des Laufens und auch die Konstruktion von Laufrobotern sind aktuell Gegenstand der Forschung.

Laufen ist die Fortbewegungsart nahezu aller Tiere auf festem Boden und hat sich im Verlaufe der Evolution als besonders geeignet für verschiedenste Domänen gezeigt. Die Erfindung des Rades und die darauf folgende Anpassung der Umwelt des Menschen haben die radgetriebene Fortbewegung perfektioniert. Abseits der „Kulturlandschaft“, in natürlicher Umgebung, versagt diese Art der Fortbewegung jedoch völlig. Der Expansionsdrang des Menschen und die Gewohnheit, unzureichende eigene Fertigkeiten mit Hilfe von ausgefeilten technischen Hilfsmitteln zu erweitern, haben das Interesse an der natürlichen Fortbewegung, dem Gehen, geweckt. Die Vielfalt der untersuchten biologischen Formen, Zweibeiner, Vierbeiner, Sechsbeiner, Achtbeiner und die jeweiligen Gangmuster haben zu einer entsprechenden Vielfalt der künstlichen Laufmaschinen geführt.

Laufen wird in diesem Zusammenhang als „eine gerichtete Fortbewegung auf geradem oder unebenen Terrain sowie Klettern in moderater Form“[3] verstanden, wobei die Fortbewegung mit Hilfe der Extremitäten der Roboter erfolgt. Die Konstruktion von Laufrobotern ist relativ einfach: Die Beine von Laufrobotern sind in ihrer Struktur identisch mit den aus industriellen Anwendungen bekannten Manipulatorsystemen (siehe Abbildung 5.1). Auch die minimale Anzahl von Freiheitsgraden eines Beines ergibt sich direkt aus den Bewegungen, die ein Bein ausführen soll: die Beine werden periodisch vom Boden abgehoben und wieder gesenkt sowie nach vorne und hinten geschwungen. Ein minimales Bein besitzt also zwei DOF. Bei einem Bein mit nur zwei DOF verlaufen die Trajektorien der Fußpunkte in Kreisbögen. Um ein lineares Verfahren

[1] *National Aeronautics and Space Administration*, die Luft- und Raumfahrtbehörde der USA.

[2] Die Bewegung von Robotern in der Schwerelosigkeit ist ebenfalls hochkomplex. Aufgrund der Tatsache, dass Anwendungsgebiete für diese Roboter rar und die Kosten hoch sind, sind Experimente hierzu allerdings nicht durchgeführt worden.

[3] Zitat aus: Zielsetzung des Schwerpunktprogramms *Autonomes Laufen* der Deutschen Forschungsgemeinschaft (DFG). Die Homepage des SPP ist unter der URL `http://www.fzi.de/ids/dfg_schwerpunkt_laufen/german/research_themes/index.htm` zu finden.

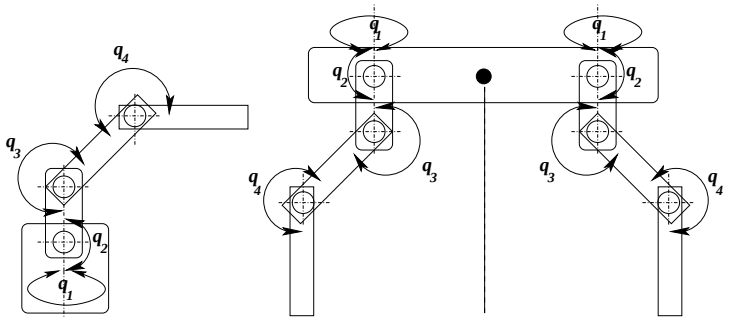


Abbildung 5.1: **Links:** Schematische Darstellung eines Manipulatorroboters mit vier DOF ohne Endeffektor. **Rechts:** Schematische Darstellung eines Laufroboters, dessen Beine kinematisch identisch sind mit dem Manipulatorsystem links.

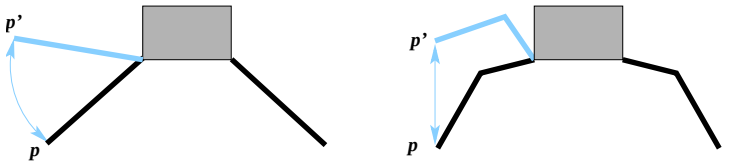


Abbildung 5.2: **Links:** Die Fußpunkte der Beine werden in Kreisbögen bewegt. **Rechts:** Durch Hinzufügen eines Gelenkes kann der Fußpunkt linear verfahren werden.

des Fußpunktes zu erreichen, muss ein weiteres Gelenk eingefügt werden (siehe Abbildung 5.2). Werden beispielsweise an den Fußpunkt eines Beines Anforderungen an die Orientierung gestellt, erhöht sich die notwendige Anzahl der Freiheitsgrade entsprechend auf mindestens sechs[4].

Die für die Bewegung der einzelnen Beine erforderlichen Transformationen (Vorwärts- und Inverse Kinematik) sind durch die industrielle Praxis bekannt und können ohne Einschränkung verwendet werden. Für die Kontrolle der Beine wird auf Methoden der Bahnplanung von Manipulatorrobotern zurückgegriffen, bei denen der Endpunkt des letzten Robotergliedes (der sog. *tool center point*) entlang einer vorgegebenen Trajektorie geführt wird. Für Laufroboter bedeutet dies also, dass der Fuß entlang einer Trajektorie geführt werden kann. Für die - klassische regelungstechnische - Kontrolle eines Laufroboters ist es also notwendig, die Trajektorien der Fußpunkte aller Beine des Roboters so zu berechnen, dass der Zentralkörper des Roboters in die gewünschte Richtung bewegt wird, ohne dass die Stabilität des Roboters beeinträchtigt wird.

5.2 Stabilität

Mit jedem Bein und jedem Gelenk steigt die Komplexität der Steuerung des Laufroboters an. Aus kontrolltechnischer Sicht ist also ein Roboter mit nur einem Bein optimal[5]. Für einen statisch stabilen Stand muss ein Laufroboter allerdings mindestens drei Beine besitzen. Eine Konfiguration der Gelenkwinkel der Beine gilt dann als statisch stabil, wenn die Projektion des Schwerpunktes des Roboters innerhalb des Aufstandspolygons liegt (siehe Abbildung 5.3). Das Aufstandspolygon wird gebildet durch die konvexe Hülle der Fußpunkte der Beine. Soll der Roboter sich fortbewegen, sind mindestens vier Beine notwendig (ein

[4] Drei Freiheitsgrade für die Positionierung im Raum, drei zusätzliche Freiheitsgrade für die Orientierung des Fußpunktes
[5] Es existieren tatsächlich einbeinige Roboter, die sich hüpfend fortbewegen, z.B. der *Hopping Robot* in [139].

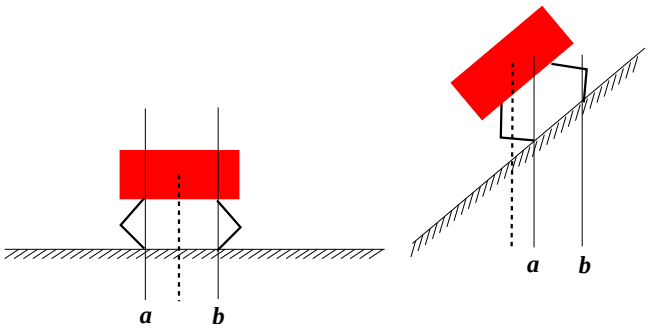


Abbildung 5.3: Stabilität des Gehens. **Links:** Die Projektion des Schwerpunkt liegt innerhalb der durch die Projektion der Aufstandspunkte gebildeten Region *a-b*. Der Roboter steht stabil. **Rechts:** Die Projektion des Schwerpunktes des Roboters liegt außerhalb der Region *a-b*, der Roboter beginnt zu kippen.

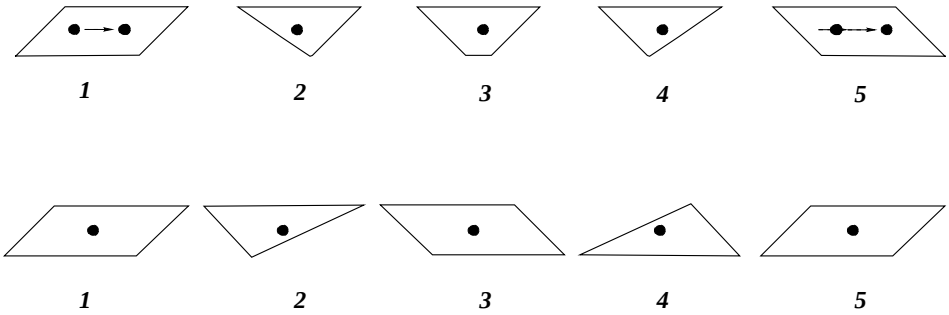


Abbildung 5.4: Schematische Darstellung der Aufstandspolygone während des Gehens. Die Ecken der Aufstandspolygone werden durch die Fußpunkte des Roboters gebildet. **Oben:** Der Schwerpunkt eines Vierbeiners muss aktiv verschoben werden (Phase 1 und 5), um Beine setzen zu können. **Unten:** Sechsbeinige Roboter im Tripod-Gang sind in jeder Situation statisch stabil. Der Schwerpunkt muss nicht vor Bewegen eines Beines verschoben werden.

Bein in der Luft, die anderen drei auf dem Boden). Sechs Beine vergrößern die Stabilität bereits erheblich. Durch weiteres Hinzufügen von Beinpaaren wird nur ein geringer Zuwachs an Stabilität gewonnen, wohingegen sich die Komplexität der Steuerung weiter erhöht. Generell wird zwischen statischer und dynamischer Stabilität von Laufrobotern unterschieden (oft wird auch synonym von statisch oder dynamisch stabilem Laufen gesprochen). Statische Stabilität wird erreicht, indem die Projektion des Schwerpunktes zu jedem Zeitpunkt innerhalb des Aufstandspolygons liegt (Abbildung 5.3). Die Menge der Laufmuster, die statisch stabiles Gehen ermöglichen, wird durch diese Bedingung sehr eingeschränkt. In Abbildung 5.4 wird deutlich, dass ein vierbeiniger Roboter seinen Schwerpunkt aktiv verschieben muss, z.B. durch eine Relativbewegung des Körpers gegenüber dem Boden, um ein Bein anheben zu können (z.B. zwischen Phase 1 und Phase 2 der Bewegung), wohingegen der Schwerpunkt des sechsbeinigen Roboters immer innerhalb des Aufstandspolygons liegt und nicht verschoben werden muss.

Dynamisches Gehen hingegen bedeutet, dass der Schwerpunkt zeitweise nicht innerhalb des Aufstandspolygons liegt. Wenn sich der Schwerpunkt nicht mehr über der Standfläche befindet (beispielsweise durch das Anheben eines Fußes), wird ein zweibeiniger Roboter instabil und beginnt zu kippen. Deshalb muss beim dynamischen Gehen der Schwingfuß so vorgeschoben werden, dass der während der Schwingphase entstehende Fall durch das Wiederaufsetzen des Fußes aufgefangen wird. Die Aufrechterhaltung der Balance wird also durch eine geplante Fußplatzierung erreicht. Die Grundlage für die dynamische Balance durch geplante Fußplatzierung ist die Berechnung des Zero Moment Point [49, 101]. Wird der Roboter nicht beschleunigt,

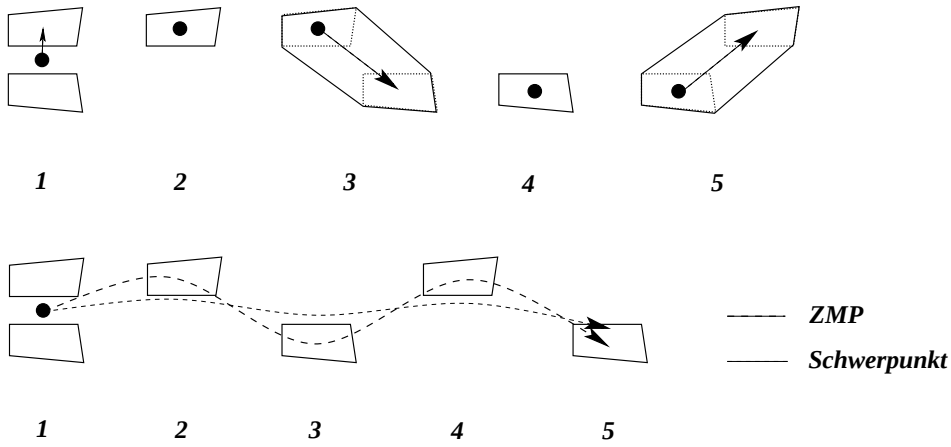


Abbildung 5.5: Bewegung des Schwerpunktes beim zweibeinigen Gehen. **Oben:** Beim statisch stabilen Gehen befindet sich die Projektion des Schwerpunktes immer innerhalb des Aufstandspolygons. **Unten:** Bewegung des Schwerpunktes und des ZMP während des dynamisch stabilen Ganges.

liegt der ZMP vertikal unter dem Schwerpunkt. Der ZMP muss innerhalb des Aufstandspolygons liegen, damit das Laufmuster dynamisch stabil ist.

5.3 Architekturvielfalt

Es existiert eine Fülle von verschiedenen Laufrobotern, die für die verschiedensten Zielsetzungen entworfen und konstruiert wurden. Eine kleine Auswahl der verschiedenen Architekturen ist in Abbildung 5.6 dargestellt. Die Vielzahl der Roboter und ihre stark variierende Qualität machen es schwierig, einen Überblick über das Feld zu geben. Im Internet existieren eine Reihe von Katalogen[6], die versuchen, alle relevanten Roboterarchitekturen, die jeweilige Institution und die Zielsetzung der Arbeitsgruppe zu dokumentieren.

Bei den Laufrobotern unterscheidet man verschiedene Architekturformen, wobei das häufigste Klassifikationskriterium die Anzahl der Beine des jeweiligen Roboters ist. Gängige Ausprägungen von Laufrobotern besitzen eine gerade Anzahl von achsensymmetrisch zum Zentralelement (auch Rumpf oder Torso genannt) angeordneten Beinen. Häufig orientiert sich die Anordnung der Beine an einem biologischen Vorbild, wobei allerdings fast immer die technische Realisierbarkeit als Randbedingung wirkt. Nur selten werden konstruktive Maßnahmen ergriffen, um biologische Architekturprinzipien exakt umzusetzen, wie z.B. bei dem vierbeinigen Roboter BISAM (siehe Abbildung 5.6), dessen Architektur in enger Zusammenarbeit von Ingenieuren, Informatikern und Biologen entworfen worden ist [86, 176].

Besonders häufig werden Sechsbeiner konstruiert, die aufgrund ihrer hohen Stabilität - es können immer wenigstens drei Beine auf dem Boden bleiben - und auch wegen ihrer morphologischen Verwandtschaft zu Insekten besonders interessant sind. Etwas seltener sind Roboter mit vier Beinen. Die geringere Stabilität während des Gehens lässt sie für die Steuerung schwieriger werden, denn selbst wenn nur ein Bein in der Luft bewegt wird, ist das Aufstandspolygon, also die Fläche, die von den Kontaktpunkten der Beine umfangen wird, in Bezug auf den Schwerpunkt des Roboters ungünstig gelegen (siehe Abbildung 5.4). Die morphologische Verwandtschaft zu - je nach Baugröße und Beingeometrie - Nagetieren und Kleinsäugern macht sie trotzdem zu einem interessanten Forschungsobjekt. Besonders schwierig sind zweibeinige Laufroboter, deren Architektur dem Menschen nachempfunden ist, da zu bestimmten Zeitpunkten nur ein Fuß den Boden

[6] Die wahrscheinlich umfangreichste Sammlung zu Aktivitäten mit Laufrobotern ist im Internet unter der URL `http://www.fzi.de/ids/WMC/WMCindex.html` zu finden.

berührt. Die Dynamik des zweibeinigen Gehens stellt daher eine große Herausforderung an Mechanik und Steuerung dar. Zweibeiniges Gehen ist von besonders hoher regelungstechnischer Komplexität, so dass die Anzahl der Roboter dieser Bauform bislang gering war, allerdings in den letzten Jahren stark angestiegen ist. Schwerpunkte der Forschung mit humanoiden Robotern liegen in der Erforschung der Laufdynamik von Zweibeinern, im Bereich der Mensch-Roboter-Schnittstelle und auf dem Gebiet der Regelungstechnik.

Es existieren viele Variationen von zwei-, vier- und sechsbeinigen Robotern, die sich in der Größe, der Beingeometrie, der Anzahl der Freiheitsgrade, dem Grad der Autonomie, der Sensorik etc. unterscheiden (eine kleine Auswahl unterschiedlicher Bauformen ist in Abbildung 5.6 dargestellt). Zusätzlich zu dieser Vielfalt existieren auch Roboter, die zu den Laufrobotern gezählt werden, aber eine ungewöhnliche Anzahl von Beinen besitzen. Hierzu zählen Einbeiner, Dreibeiner, Achtbeiner oder auch Roboter mit mehr als acht Beinen. Es existieren sogar Roboter ohne Beine, die Schlangen nachempfunden sind. Auch hybride Architekturen sind Gegenstand aktueller Forschung. Hier werden Roboter untersucht, die sowohl Beine als auch Räder zur Fortbewegung verwenden.

5.4 Nachteile dieser Architekturvielfalt

Von den meisten Robotern existieren nur wenige, meist sogar nur ein Exemplar. Eine Ausnahme stellt zum Beispiel der Aibo von Sony (Abbildung 10.1) dar, von dem bis Ende des Jahres 2000 weltweit ca. 50.000 Exemplare abgesetzt wurden. Wird von vornherein nur ein Prototyp entwickelt, bedeutet dies gleichzeitig, dass bis auf Standardbauteile wie Motoren, Sensoren, Kabel etc. jedes Teil des Roboters handgearbeitet ist. Hinzu kommt eine Vielfalt von Forschungszwecken, zu deren Untersuchung diese Roboter entworfen wurden. Dies können Aspekte der Sensorik, der Antriebstechnik, der Energieversorgung, der Navigation, der Regelung, der Softwarearchitektur, der künstlichen Intelligenz, der Bioinformatik, der Prothetik, der Biomechanik, der Bildverarbeitung, der militärischen Anwendung, der Landwirtschaft, oder auch der Mensch-Maschine-Interaktion sein. Diese Vielfalt der Anforderungen an die Roboter und an die Kontrollprogramme verhindert, dass Programme, die auf einem Roboter untersucht wurden, auf irgendeinem anderen Roboter ausgeführt werden können.

Zusätzlich erschwerend wirkt sich aus, dass während der Konstruktion eines Roboters, zumindest in der Prototyp-Phase, im Allgemeinen zahlreiche Veränderungen an der Hardware des Roboters vorgenommen werden. Dies können beispielsweise Veränderungen der Beingeometrie, die Installation zusätzlicher Hardwarekomponenten oder die Verwendung von Motoren mit unterschiedlichen Leistungscharakteristiken, Abmessungen oder Massen sein. Die Gesamtcharakteristik des Roboters wird durch diese Änderungen modifiziert, was direkten Einfluss auf die Kontrollsoftware hat, die deshalb ebenfalls immer angepasst werden muss.

Gerade Programmteile, die die Koordination der Beine für die Fortbewegung übernehmen, sind von diesen Änderungen stark betroffen und müssen manuell angepasst werden. Eine Möglichkeit, diese Probleme zu umgehen ist es, Laufprogramme mit Hilfe von Techniken des maschinellen Lernens - beispielsweise mit Genetischer Programmierung - automatisch zu generieren. Hierbei wird erstens das manuelle Erstellen von Laufprogrammen vermieden und zweitens die Möglichkeit gegeben, mit der gleichen Technik für eine Vielzahl von verschiedenen Roboterarchitekturen Laufprogramme zu entwickeln, d.h. unabhängig von der konkreten Realisierung eines Laufroboters zu werden. Die Evolution dieser Programme ist deshalb prinzipiell unabhängig von der gewählten Bauform des Roboters, weil die Roboterarchitektur reduziert wird zu einem Parameter des EA. Zusätzlich ist es auch möglich, Personenkreise bei der Entwicklung von Laufrobotersteuerungen einzubeziehen, die ohne diese Technik aufgrund fehlender Kenntnis der mathematischen und ingenieurwissenschaftlichen Grundlagen der Robotik dazu sonst nicht in der Lage wären.

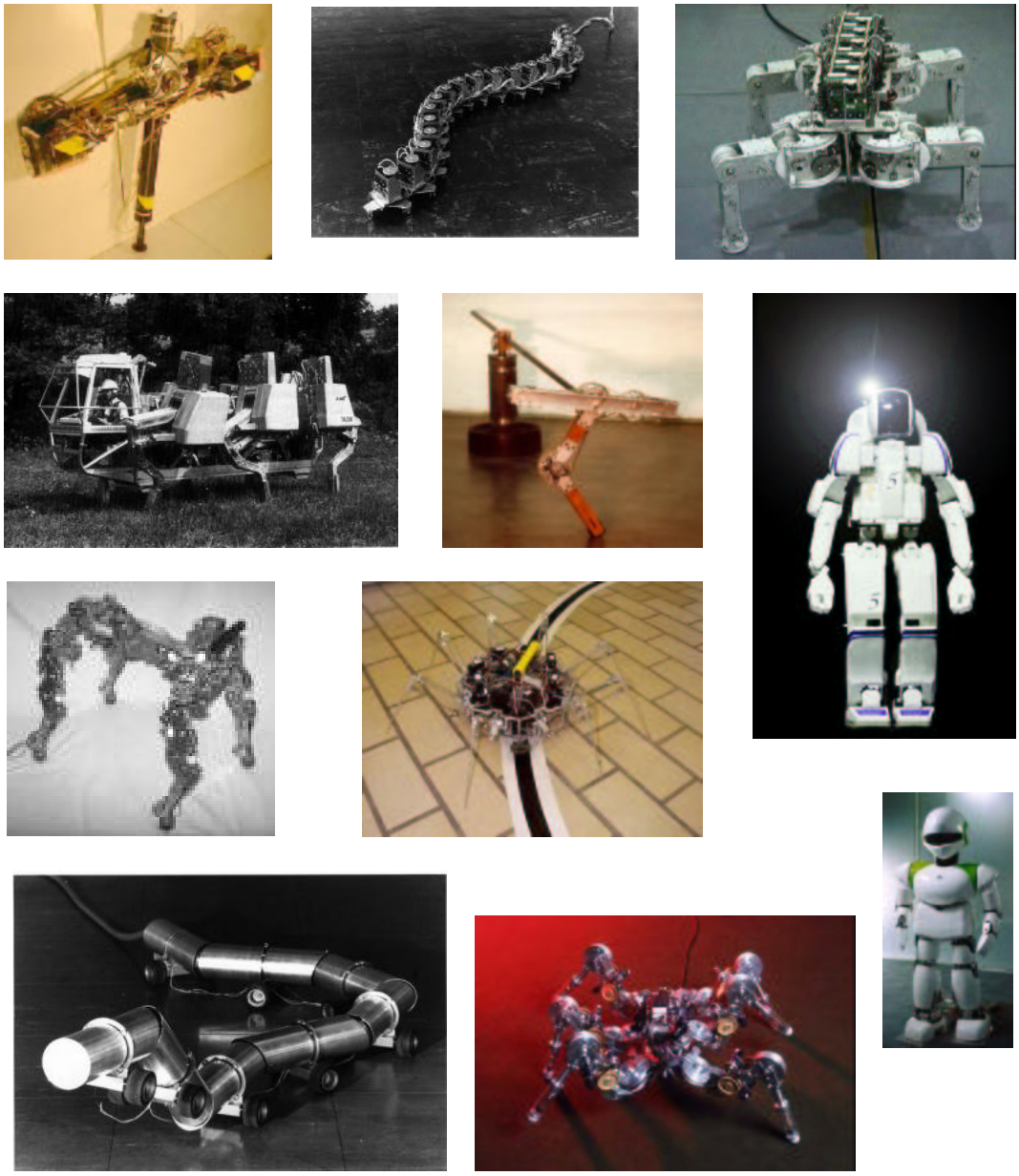


Abbildung 5.6: Beispiele für die Vielfalt der Roboterarchitekturen. Von links oben nach rechts unten: ARL-Monopod, ACM-III, TITAN-VIII, ASV, OLIE, Honda P3, BISAM, Robotic Spider, PINO, OBLIX, LAURON-III[7].

Im folgenden Kapitel wird die Evolution von Laufroboterprogrammen mit GP in ihren Grundzügen erläutert. In Teil III werden dann Ergebnisse der durchgeführten Experimente dargestellt, wobei dort explizit auf die Vielfalt der verwendeten Roboterstrukturen eingegangen wird.

[7] Bilder aus: Walking Machines Catalogue (`http://www.fzi.de/ids/WMC/WMCIndex.html`).

Kapitel 6

Evolution von Laufrobotersteuerungen mit GP

Robotersteuerungen für Laufroboter werden normalerweise speziell für eine Roboterarchitektur mit den Methoden der klassischen Regelungstechnik entworfen. Hierfür wird eine zu regelnde Größe definiert (beispielsweise die Position und Orientierung der Fußpunkte der Beine des Roboters) und ein entsprechender Regler entworfen. Invariante Rahmenbedingungen für die Regelung werden identifiziert (u.a. maximale Drehmomente und Winkelgeschwindigkeiten der Motoren, Massenverteilung, geometrische Abmessungen) und berücksichtigt. Andere Bedingungen werden frei festgelegt (z.B. statisch oder dynamisch stabiles Gehen, Schrittlänge und -frequenz etc.). Aus diesem Reglerentwurf werden Differentialgleichungssysteme abgeleitet, die aus der zu regelnden Größe und der Differenz aus Soll- und Istzustand des Systems die Stellwerte der einzelnen Aktoren berechnen.

Ein Beispiel ist die Führung der Fußpunkte der Beine entlang vorgegebener Trajektorien. Die Sollwinkel der Beine werden dann in regelmäßigen Zeitabständen in Abhängigkeit von der gewünschten Position und Orientierung der Fußpunkte auf der Trajektorie mit Hilfe der inversen Kinematik bestimmt. Aus dem aktuellen Zustand des Systems, d.h. den aktuellen Winkelgeschwindigkeiten und -beschleunigungen der Gelenke und der aktuellen Position und Orientierung der Fußpunkte werden dann die Beschleunigungen der einzelnen Gelenke berechnet, die zum Erreichen der Sollkonfiguration innerhalb eines gegebenen Zeitintervalls notwendig sind. Der Istzustand des Roboters wird mit einer festen Frequenz abgefragt und die jeweiligen Stellgrößen durch Lösen der Differentialgleichungen mit Hilfe von numerischen Verfahren berechnet (siehe Abbildung 6.1). Hierdurch wird eine kontinuierliche Regelung der Größen erreicht. Die Sollwerte der Regelung, im gegebenen Beispiel also die Trajektorien der Fußpunkte, werden normalerweise extern berechnet und vorgegeben.[1]

Eine Laufrobotersteuerung ist demnach im Prinzip nichts anderes als eine in einer Schleife aufgerufene Rechenvorschrift, die aus Eingabeparametern (dem Istzustand des Systems und den extern vorgegebenen oder online berechneten Sollwerten) entsprechende Ausgabeparameter (Motorkommandos) erzeugt. Diese *black-box*-Sicht der Steuerung von Laufrobotern ermöglicht zwei Schlüsse: (i) Jedes beliebige Programm mit geeignetem Ein-/Ausgabeverhalten kann den Roboter gleichermaßen steuern. (ii) Die Abweichung vom vorgegebenen Ein-/Ausgabeverhalten ist ein Kriterium für die Güte einer Robotersteuerung. Die Qualität einer klassischen Laufrobotersteuerung hängt - vorausgesetzt, die berechneten Trajektorien der Fußpunkte ermöglichen ein stabiles Laufen des Roboters - also davon ab, inwieweit die vorgegebenen Sollwerte der Regelung erreicht werden.

[1] Es existieren auch Verfahren zur online-Berechnung der Trajektorien. Hierbei wird beispielsweise nach bestimmten Stabilitätskriterien der optimale Fußpunkt eines Beines berechnet. Diese Bewegung des Roboters wird auch als *free gait* bezeichnet, da sie keinem bestimmten Muster folgt.

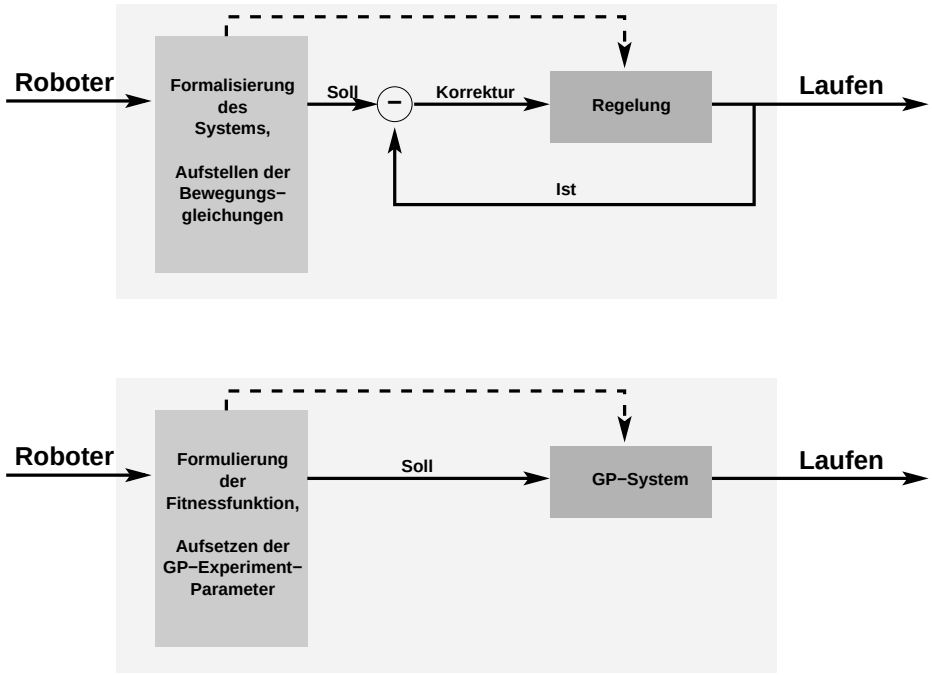


Abbildung 6.1: **Oben:** Klassische Vorgehensweise beim Reglerentwurf. **Unten:** Entwurf eines GP-Systems.

Bei der Evolution einer Laufrobotersteuerung entfällt die mathematisch komplexe manuelle Erstellung eines Regelsystems. Die Vorgabe von expliziten Regelgrößen entfällt ebenfalls. Stattdessen werden Steuerungen evolviert, deren Güte anhand von makroskopischen Qualitätskriterien bestimmt wird. Dies kann beispielsweise durch die Forderung nach einer möglichst schnellen und geradlinigen Fortbewegung des Roboters geschehen. Durch die Evolution von Programmen, die ein geeignetes Ein-/Ausgabeverhalten aufweisen, d.h. von Programmen, die den Istzustand des Roboters messen und geeignete Motorkommandos berechnen, wird eine immer bessere Erfüllung der Anforderungen erreicht. Die Qualitätskriterien für das Gesamtverhalten des Roboters implizieren gewisse Ein-/Ausgabeverhalten der Programme, so dass es theoretisch durchaus möglich ist, Laufprogramme zu evolvieren, die mit einer klassischen Regelung des Laufroboters identisch sind. Notwendig hierfür ist eine entsprechend formulierte Fitnessfunktion.

Eine Robotersteuerung mit Hilfe von GP zu evolvieren bedeutet auf den ersten Blick eine Verschwendung von Rechenzeit, da viel Aufwand auch für die Evaluation von schlechten Individuen getrieben werden muss. Dies ist ein großer Kritikpunkt von Verfechtern der traditionellen, ingenieurmäßigen Herangehensweise. Hierbei wird jedoch übersehen, dass für die Anwendung klassischer Verfahren genaues Modellwissen verfügbar sein muss und darüber hinaus auch die Methoden, auf die hierbei zurückgegriffen wird, in jahrelangen Bemühungen entwickelt worden sind. Das für die Anwendung klassischer Methoden erforderliche „Problemwissen“ stellt außerdem gleichzeitig ein eigenes Problem dar, denn es schränkt den Personenkreis ein, die zur Lösung des Problems fähig sind. Stärke der evolutionären Methoden hingegen ist es, ohne fachspezifisches Problemwissen gute Resultate zu liefern.

6.1 Automatische Erzeugung von Kontrollprogrammen mit SIGEL

Durch die Existenz eines Gütekriteriums für Robotersteuerungen ist es möglich, mit Hilfe von GP in einem evolutionären Prozess immer bessere Steuerprogramme von Laufrobotern zu erzeugen. Hierbei spielen nahezu alle Kriterien, die den manuellen Reglerentwurf für Laufroboter zu einem komplexen und aufwändigen

Prozess machen, keine Rolle. Die mit Hilfe von GP erzeugten Programme steuern „blind“ einen Roboter. Hierbei ist von entscheidender Bedeutung, dass der Grundalgorithmus unabhängig von Parametern des Roboters ist und deshalb auch für *jeden beliebigen Roboter* angewendet werden kann (siehe Abbildung 6.1). Die für das Laufen wichtigen Eigenschaften des Roboters werden im Verlaufe der Evolution automatisch identifiziert und verwendet, wohingegen beim klassischen Reglerentwurf dieses Modellwissen manuell extrahiert und implementiert werden muss.

GP (siehe Abschnitt 3) ist eine Variante eines EA, die für die Evolution von Laufrobotersteuerungen besonders geeignet ist. Individuen in GP werden als Programme repräsentiert, so dass die Übertragbarkeit der Resultate der Evolution auf die zu steuernden Roboter relativ problemlos erfolgen kann. Für die Evolution von Laufrobotersteuerungen mit GP wurde das Tool SIGEL[2] entwickelt.

Die Individuen der mit SIGEL durchgeführten Experimente bestehen aus einer linearen Liste von Programmzeilen (Linear-GP) und werden in einem Interpreter ausgeführt. Dieser Interpreter verbindet die Register und Instruktionen eines GP-Programms mit dem zu steuernden Roboter. Hierfür müssen die Menge der Instruktionen und die Menge der Terminale angepasst werden. Zusätzlich zu der Menge der Standardinstruktionen, die normalerweise die vier grundlegenden arithmetischen Operationen $(+, -, \cdot, /)$ und eventuell trigonometrische Funktionen (sin, cos, tan) umfasst, müssen Instruktionen zur Verfügung gestellt werden, die zur Laufzeit eines Programmes Zugriff auf Sensorinformationen des Roboters (sofern Sensoren vorhanden sind) bzw. Zugriff auf Aktoren eines Roboters erlauben (SENSE, MOVE). Diese Instruktionen, sowie die Instruktionen zur Datenmanipulation (Datenspeicherung, -transfer und -vergleich) und zur Programmkontrolle (Sprunganweisungen, etc.) bilden die Menge der Instruktionen, aus denen ein Roboterkontrollprogramm gebildet werden kann.

6.1.1 Initialisierung

Zu Beginn der Evolution wird eine Startpopulation gebildet. Die Individuen dieser Population müssen vor dem Experiment initialisiert werden. Hierfür wird eine zufällig bestimmte Anzahl von Programmzeilen erzeugt, die jeweils aus einer Instruktion besteht (und ggf. weiteren Parametern, die ebenfalls zufällig bestimmt werden). Es können Einschränkungen bezüglich einer Mindest- bzw. Maximallänge der Programme bestehen, die eingehalten werden müssen. Falls keine andere Verteilung der Wahrscheinlichkeiten für die einzelnen Instruktionen angegeben wird, sind alle Instruktionen mit gleicher Häufigkeit in einem Programm enthalten. Die Population kann aus einer beliebigen Anzahl Individuen bestehen.

In SIGEL können Individuen neben der rein zufälligen Initialisierung auch manuell initialisiert werden. Hierfür ist der Import von manuell erstellten Programmen enthalten, oder auch von Programmen aus anderen Experimenten. Darüber hinaus erlaubt es SIGEL, die Zusammensetzung der Population zu jedem Zeitpunkt zu verändern.

6.1.2 Variationsoperatoren

Die Programme werden mit den Standardvariationsoperatoren verändert. Implementiert sind zwei verschiedene Mutationsoperatoren, wobei der eine Operator Bestandteile einer Programmzeile, der andere ganze Programmzeilen mutiert, wozu in diesem Falle auch das Löschen bzw. Einfügen kompletter Zeilen gehört. Crossover findet zwischen zwei Individuen statt, wobei *one-point* bzw. *two-point-crossover* implementiert worden sind. Bei der Anwendung der Operatoren wird darauf geachtet, eventuelle Längenbeschränkungen oder Mindestlängen nicht zu über- oder unterschreiten. Es ist noch ein weiterer Operator implementiert, der nur eine Kopie des Elternteils anlegt und deswegen Reproduktionsoperator genannt wird. Jedes Individuum besitzt komplette Aufzeichnungen über seinen Werdegang, d.h. über Ort und Zeitpunkt von Mutationen

[2] SIGEL: Simulator für GP-Evolvierte Laufrobotersteuerungen

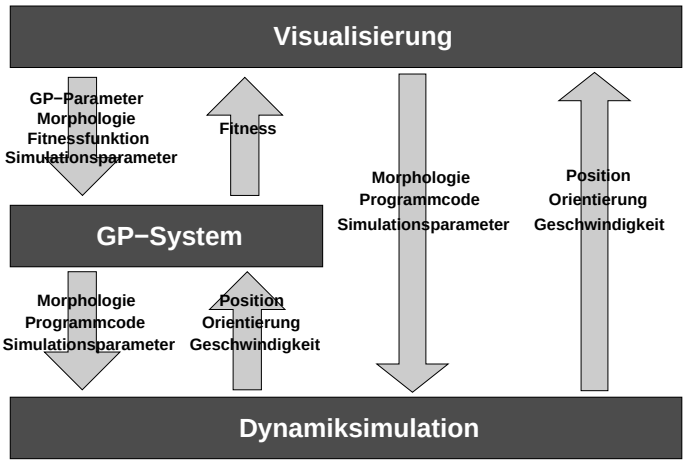


Abbildung 6.2: Der Aufbau des Robotersimulationssystems SIGEL.

und über den Partner bei Crossover etc., so dass die Entwicklung der Population durch die Anwendung der Variationsoperatoren genau verfolgt werden kann.

6.2 Evaluation von Laufprogrammen in der Simulation

Die Evolution von Computerprogrammen mit Genetischer Programmierung erfordert generell eine große Anzahl von Fitnessauswertungen. Die Laufzeit für ein GP-Experiment ist im Wesentlichen durch diese Größe bestimmt. Wenn die Evaluation von Individuen nicht nur mathematische Berechnungen erfordert, sondern zusätzlich mechanische Geräte gesteuert werden, so bedeutet eine hohe Anzahl an Evaluationen gleichzeitig eine hohe Belastung des Materials. Die meisten mechanischen Konstruktionen sind nicht dahingehend konstruiert, häufige Belastungen im Grenzbereich auszuhalten, Belastungen, die zwangsläufig entstehen, wenn durch einen Zufallsprozess erzeugte Steuerungsroutinen ausgeführt werden. Aufwändig zu implementierende Filterprogramme könnten diese Spitzenbelastungen reduzieren, generell bleibt aber die Belastung durch hunderte, wenn nicht tausende Ausführungen von Steuerprogrammen während des evolutionären Prozesses bestehen. Aus diesem Grunde wurde in SIGEL ein System integriert, das Laufroboter in einer physikalischen Umgebung simuliert. In diesem Simulator wurden dann Experimente zur Evolution von Laufroboterprogrammen durchgeführt.

Das Simulationssystem SIGEL besteht aus drei wesentlichen Bestandteilen (siehe Abb. 6.2). Über eine Benutzeroberfläche wird das System gesteuert und parametrisiert. Die Roboter werden dort visualisiert. Das integrierte GP-System ist für die Durchführung von evolutionären Experimenten zuständig. Der Roboter wird gemäß dem in einem Interpreter ausgeführten GP-Individuum in der Dynamiksimulation bewegt. Im Folgenden werden die Funktionen der einzelnen Teile und ihre Vernetzung detaillierter erläutert.

6.2.1 Das GP-System

Im GP-Modul wird die Evolution von Kontrollprogrammen für einen ausgewählten Roboter durchgeführt. Alle Individuen besitzen eine lineare Struktur. Die Fitness eines Individuum wird berechnet, indem es den Roboter in der Dynamiksimulation bewegt. Die Auswertung der Individuen erfolgt parallelisiert über PVM und kann jederzeit unterbrochen werden. Alle Parameter des Roboters, der Dynamiksimulation, des GP-Systems und von PVM können in SIGEL verändert werden.

Zu jedem Zeitpunkt während der Evolution kann in den laufenden Prozess eingegriffen werden. Unter anderem ist es möglich, die Populationsstruktur (Größe, Zusammensetzung) jederzeit zu ändern und auf diese Weise in die Evolution einzugreifen. Individuen können separat gespeichert und geladen werden, so dass es beispielsweise möglich ist, eine Startpopulation nicht zufällig, sondern nach besonderen Gesichtspunkten zusammenzustellen (siehe hierzu auch Kapitel 9). Jeder Parameter eines Experimentes kann zur Laufzeit verändert werden, z.B. Operatorwahrscheinlichkeiten, die Verteilung der Instruktionswahrscheinlichkeiten, Längenbeschränkungen für Individuen etc. Die Interaktivität des GP-Systems erlaubt vielfältige Einflussnahmen des Experimentators zur Laufzeit der Evolution, um auf diese Weise Erfahrungen aus vergangenen Experimenten einfließen lassen zu können.

Der Befehlssatz, der dem GP-System zur Verfügung gestellt wird, umfasst elementare arithmetische Operationen wie Subtraktion, Addition, Multiplikation, etc. sowie Befehle zur Datenmanipulation (Laden, Kopieren und Vergleichen von Registerinhalten sowie Sprünge). Zusätzlich sind zwei Befehle vorgesehen, die zum Auslesen von Sensorinformationen (SENSE) und zur Steuerung der Motoren (MOVE) dienen. Der genaue Umfang der Operatormenge wird vor Beginn eines Experimentes festgelegt.

6.2.2 Die Dynamiksimulation

Die Dynamiksimulation ist das Kernstück des SIGEL-Systems. Hier werden die Bewegungen des Roboters simuliert, wobei auf eine möglichst exakte Übereinstimmung zwischen Simulation und Realität Wert gelegt wird. Die Roboter werden einerseits passiv durch Kräfte bewegt, die durch die Interaktion mit der Umwelt entstehen (zum Beispiel durch die Gravitation, durch Masseträgheiten, durch Kollision von Roboterteilen mit der Umwelt etc). Auf der anderen Seite wird ein Roboter auch aktiv bewegt, indem an den definierten Gelenken Drehmomente aufgebracht werden. Wo (in welchem Motor) und in welchem Ausmaß die Drehmomente aufgebracht werden, wird durch die Ausführung eines GP-Individuums im Interpreter bestimmt. Der Interpreter ist direkt mit der physikalischen Simulation verknüpft.

Die GP-Instruktionen, die während der Simulation ausgeführt werden, besitzen eine definierte Ausführungsdauer. Während dieser Dauer werden die Bewegungen des Roboters simuliert. Anschließend wird die Simulation gestoppt und die nächste Instruktion wird ausgeführt. Über die MOVE-Instruktion ist ein Programm in der Lage, die Simulation zu beeinflussen. Hierfür wurden zwei verschiedene Methoden implementiert.

Momentbasierte Ansteuerung Die MOVE-Instruktionen bewirken, dass in dem durch weitere Parameter angegebenen Gelenk ein bestimmtes Moment geschaltet wird. Über die Dauer der Instruktion, einem globalen Parameter der Simulation, kann auch die Wirkung des Momentes angegeben werden. Der Roboter bewegt sich entsprechend der im Ausführungszeitraum erreichten Beschleunigung der einzelnen Körper.

Steuerung über Stellwinkel Die MOVE-Instruktionen wirken durch die Ansteuerung eines Motors mit einem neuen Stellwinkel. Dieser Stellwinkel wird durch den Interpreter an die Simulation übergeben. Der jeweilige Motor fährt in den folgenden Simulationsschritten die neue Winkelposition an. Durch spezielle Anweisungen kann das ausgeführte Programm die benötigten Sensorinformationen des Roboters „auslesen“. Hierbei werden durch die Instruktion spezielle Register des GP-Systems mit Werten beschrieben, die durch die physikalische Simulation bereitgestellt werden. Den Transfer der Daten von der Simulation zum GP-System übernimmt der Interpreter. Auf diese Weise wird ein bei Robotern in der Realität weit verbreiteter Motortyp modelliert, der Servomotor.

Die in der Roboterbeschreibung definierten Gelenkwinkel und die Raumposition des Torsoelementes beschreiben die Initialposition. Beim Start der Simulation wird der Roboter in dieser Position aufgestellt. Neben der Geometrie und der Masseverteilung des Roboters benötigt die Simulation noch Daten über Reibkoeffizienten. Über die Definition von Materialien und deren Eigenschaften in einer Umgebungsbeschreibung und die Zuordnung von Robotergliedern bzw. des Bodens zu Materialien wird die Simulation mit diesen

Informationen versorgt. Durch den Reibschluss einzelner Glieder des Roboters mit dem Boden wird die dadurch verursachte Vortriebskraft simuliert. Auch der Vektor der Schwerkraft kann frei definiert werden, beispielsweise um das Gehen auf einer geneigten Ebene zu simulieren. Bei Kollision von Robotergliedern mit dem Boden werden Reaktionskräfte berechnet. Hierfür werden Feder- und Dämpferkonstanten angegeben. Bedingt durch die verwendete Bibliothek für die Dynamiksimulation können nur Kollisionen von Objekten mit dem Boden erkannt werden [115]. Es ist also möglich, dass sich Glieder des Roboters selbst durchdringen. Um dies nach Möglichkeit zu vermeiden, werden maximale und minimale Gelenkwinkel definiert, zwischen denen der Arbeitsraum des Gelenks liegt. Wird ein Gelenk über die maximale Auslenkung (definiert in der Roboterbeschreibung) hinaus angesteuert, so werden auch hier Reaktionskräfte gemäß den Gelenk-Feder- und -Dämpferkoeffizienten berechnet, damit der Arbeitsbereich des Gelenkes nicht verlassen wird.

Der Roboter, die Umgebungsbeschreibung, die Reib-, Feder- und Dämpferkoeffizienten sowie die Programm- und deshalb zeitabhängigen Momente in den einzelnen Gelenken werden zusammen in der Dynamiksimulation in ein System von Differentialgleichungen überführt, das numerisch gelöst wird. Es kann zwischen einem Standard-Runge-Kutta-Verfahren[3] mit fixer Schrittweite und einem erweiterten Runge-Kutta-Verfahren mit adaptiver Schrittweite gewählt werden. Es hat sich gezeigt, dass einfache Robotermodelle mit wenigen Gliedern und Gelenken schon mit einer fixen Schrittweite von $h = 0.01$ ohne numerische Probleme zu simulieren sind. Ist das Robotermodell komplexer (siehe Kapitel 9), so muss eine kleinere Schrittweite gewählt werden ($h = 0.0005$). Zusätzlich ist eine Schrittweitenanpassung für das Lösungsverfahren auszuwählen, da sonst das System aufgrund numerischer Ungenauigkeiten und der Steifheit der Differentialgleichungen nicht simulierbar ist. Starke Kräfte in den Gelenken und hohe Beschleunigungen bzw. schnelle Bewegungen können ebenfalls zu numerischen Problemen führen, so dass auch hier zu kleineren und adaptiven Schrittweiten gegriffen wird.

6.2.3 Die Visualisierung

Die Güte von Laufprogrammen kann sowohl im Zeitverlauf als Graph (Fitness über der Zeit in Generationen) als auch durch Visualisierung der Bewegungen des Roboters dargestellt werden. Für den Verlauf der Evolution ist allein die numerische Bewertung der Laufprogramme durch die Fitnessfunktion ausschlaggebend, außerordentlich hilfreich für den Betrachter ist allerdings die Visualisierung der Bewegungen des Roboters. Hierbei kann zwischen der Simulation in Echtzeit und einem Playback der aufgezeichneten Bewegungen gewählt werden. Gerade bei komplexen Robotern und daraus resultierender kleiner Schrittweite des numerischen Verfahrens ist die Playback-Funktion nützlich. Die Teilbewegungen werden als POVRAY-Files abgespeichert und können nach der Simulation offline in qualitativ hochwertige Videos gerendert werden.

Durch die Visualisierung der Bewegungen können die Programme vom Experimentator phänotypisch analysiert und verglichen werden. Gerade bei der Aufstellung einer Fitnessfunktion, die die Güte einer Bewegung abstrakt beschreibt, ist eine Visualisierung der evolvierten Programme sehr nützlich, um die Eignung der mathematischen Formel validieren zu können (siehe hierzu auch Abschnitt 9.2.2).

Laufroboter sind hervorragend geeignete Demonstrationsobjekte für die Anwendung Evolutionärer Algorithmen auf komplexe Problemstellungen. Eine Visualisierung der evolvierten Bewegungsabläufe, auch von Zwischenstadien im Verlaufe der Evolution, bietet die Möglichkeit, den Prozess der Evolution anhand eines Beispieles deutlich vor Augen zu führen.

[3] Das Runge-Kutta-Verfahren ist eine Standardmethode zur numerischen Integration von Differentialgleichungen. Siehe hierzu auch [138].

6.3 Evaluation von Laufprogrammen mit realen Robotern

Unter Umständen ist es wünschenswert, die Güte von Laufprogrammen zu bestimmen, indem sie auf einem realen Roboter ausgeführt werden. Dies kann beispielsweise der Fall sein, wenn für den Roboter kein Computermodell existiert, oder die Ergebnisse der Simulation durch Unzulänglichkeiten des verwendeten Modells nicht auf den realen Roboter übetragbar sind.

Bei Experimenten mit realen Robotern tauchen allerdings mehrere Probleme auf. So ist es schwierig, die Experimente automatisiert durchzuführen. Hierfür notwendig ist zusätzliches Equipment, das den Roboter nach einem Fall wieder aufrichtet und auch dafür sorgt, dass der Roboter das Experimentierfeld nicht verlässt. Außerdem ist eine automatische Erkennung der Fitness einer Bewegung auf bilderkennende Algorithmen und zusätzliche Ausrüstung angewiesen. Gerade in der Anfangsphase der Evolution, während der viele erratische Bewegungen ausgeführt werden, ist es schwierig, graduelle Unterschiede maschinell zu erkennen. Um längere Experimente durchführen zu können, ist eine externe Stromversorgung notwendig. Hierdurch wird ein Nachschleppen des Energieversorgungskabels notwendig, eventuell zusätzlich noch ein Datenkabel für den Transfer der Kontrollprogramme. Ist der Roboter nicht mit einer autonomen CPU ausgerüstet, muss die komplette Steuerung der Motoren und der Transfer der Sensordaten über eine Leitung erfolgen[4].

Das Problem, eine automatische Fitnessbewertung inklusive Bildverarbeitung etc. implementieren zu müssen wird umgangen, wenn stattdessen interaktiv evolviert wird. Hierbei erfolgt die Bewertung der Fitness eines Individuums manuell durch einen Experimentator. Wird turnierbasierte Selektion verwendet, entfällt die Angabe von numerischen Werten zur Beschreibung der Güte von Individuen, die aufgrund der mangelnden Vergleichsmöglichkeiten zum Bewertungszeitpunkt und aufgrund eines fehlenden absoluten Qualitätsmaßstabes problematisch ist. Werden stattdessen die Individuen in Turnieren miteinander verglichen, entscheidet nur der Unterschied zwischen „schlechterem“ und „besserem“ Individuum. Dieser Unterschied ist für einen menschlichen Beobachter mit einer - natürlich subjektiven - Vorstellung von „gutem“ bzw. „schlechtem“ Laufen leicht festzustellen, es ist allerdings schwierig, die entsprechenden Bewertungskriterien mathematisch exakt zu formulieren (siehe auch Abschnitt 9.2.2). Diese sogenannte interaktive Evolution bettet die menschliche Intuition, die Subjektivität, die Wahrnehmung, das Verstehen und Kategorisieren ein in den Evolutionären Algorithmus, um genau von diesen mathematisch nicht definierbaren Größen zu profitieren. Einen Überblick über das Gebiet der interaktiven Evolution gibt beispielsweise [165].

Reale Roboter für die Evolution von Laufprogrammen zu verwenden birgt neben den Nachteilen - Verschleiß, Dauer, personeller und apparativer Aufwand - aber auch Vorteile. So entfällt die unter Umständen komplizierte Erstellung eines Computermodelles des Roboters. Unvermeidliche Ungenauigkeiten bei der Modellierung des Roboters, Einschränkungen der Dynamiksimulation, im Falle der verwendeten Programmbibliothek zum Beispiel das Fehlen von Elastizitäten, und numerische Fehler führen zwangsläufig dazu, dass sich simulierte und reale Roboter prinzipiell unterschiedlich verhalten. Durch aufwändige Modellierung und verfeinerte Simulationsmethoden kann die Differenz (*reality gap*) zwar verkleinert werden, sie bleibt allerdings immer bestehen.

6.4 Hybride Methoden

Wenn nur offline in der Simulation gelernt wird und die Ergebnisse anschließend auf den Roboter übetragen werden, muss ein entsprechender Aufwand für die Erstellung des Modells und für die Dynamiksimulation getrieben werden (siehe z.B. [34, 109]). Kann die Evolution von Laufrobotersteuerungen komplett auf dem realen Roboter erfolgen (entweder vollständig auf dem Roboter oder teils auf dem Roboter, teils auf einem

[4] Bei manchen Robotern existiert auch die Möglichkeit, über *Wireless LAN (WLAN)* Daten bzw. Programme zu übertragen. Diese Tatsache ändert aber nichts an der Notwendigkeit einer externen Energieversorgung bei lang andauernden Experimenten.

angeschlossenen Rechner) und eine korrekte Fitnessbewertung ist gewährleistet, wird der Aufwand durch die Dauer und Anzahl der Ausführungen bestimmt (siehe z.B. [85, 119]).

Es ensteht also immer das Dilemma, Zeit in eine exaktere Simulation zu investieren oder die gleiche Zeit für die Evolution mit einem realen Roboter aufzuwänden. Ein Ausweg besteht darin, beide Möglichkeiten zu verwenden (wie beispielsweise in [117, 119]). Wenn erst mit einem Modell in der Simulation evolviert wird und anschließend die Ergebnisse auf einem realen Roboter den tatsächlichen Gegebenheiten angepasst werden, so wird weniger Zeit für die Modellierung und Simulation aufgewändet (weil das Modell weniger exakt sein muss), gleichzeitig werden weniger Auswertungen auf dem Roboter notwendig sein, da schon relativ gute Individuen in der Startpopulation enthalten sind. Diese hybride Methode löst das Problem, sehr exakte Simulationen oder reale Roboter verwenden zu müssen, indem die Vorteile beider Methoden vereint und die Nachteile weitestgehend vermieden werden. Sie wird in Kapitel 9 angewendet.

Teil III

Experimente

Kapitel 7

Analyse von Methoden zur Verkürzung der Laufzeit

Die Laufzeit für ein GP-Experiment ist im Wesentlichen durch die hohe Anzahl von Auswertungen und deren Dauer bestimmt. Es ist demzufolge wünschenswert, die Zahl der Auswertungen bzw. deren Dauer so weit wie möglich zu verringern, um die Gesamtlaufzeit des Algorithmus entsprechend zu verkürzen. Eine Reduzierung der Zahl der Evaluationen dient außerdem der Schonung des Materials, wenn reale Roboter eingesetzt werden. Simulationen vermeiden zwar den Verschleiß von eventuell sehr teuren Maschinen, bedeuten allerdings aufwändige Rechenvorgänge. Werden komplexe Roboter und ihre Wechselwirkung mit der Umwelt in einer Dynamiksimulation dargestellt, sind rechenzeitintensive Algorithmen notwendig, um mit einer hohen Exaktheit die Bewegungen und andere physikalische Gesetzmäßigkeiten nachzubilden.

7.1 Parallelisierung

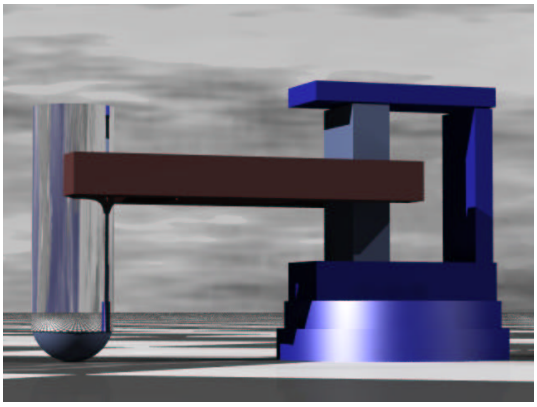
Es ist eine Eigenschaft von EA, dass sie besonders gut parallelisierbar sind. Bei generationenbasierten Verfahren können die Auswertungen aller Individuen einer Generation parallel erfolgen, und auch bei *steady state* Verfahren ist es möglich, Auswertungen von Individuen parallel durchzuführen. Hierzu wird ein Turnierplan $\{T_1, \ldots, T_N\}$ erstellt, der festlegt, welche Individuen in einem Turnier miteinander verglichen werden. Hierzu werden die Turniere T_x geordnet:

$$T_x \prec T_y :\Leftrightarrow \exists i : \text{ Individuum } i \text{ nimmt teil an } T_x \text{ und an } T_y \text{ und es gilt } x < y. \tag{7.1}$$

Unabhängige Turniere, d.h. Turniere an denen Individuen teilnehmen, die nicht vorher an anderen Turnieren beteiligt sind, können parallel ausgeführt werden. Diese Methode garantiert, dass das Experiment unabhängig von Systemparametern wie Netzwerkverkehr oder Rechnergrundlast bleibt und beliebig oft wiederholt werden kann. Für die Parallelisierung wurde einerseits PVM [67] verwendet, andererseits das am Lehrstuhl XI installierte LSF-System. In beiden Fällen erfolgt die Verteilung der auszuführenden Bewertungen auf Rechner im Rechencluster im entsprechenden Scheduler, d.h. im PVM-Master bzw. LSF-Master.

7.2 Vorzeitiger Abbruch

Bei den ersten durchgeführten Experimenten sollte ein Roboterbein (Abb. 7.1, links) von der Ausgangsposition (im Bild dargestellt) zu einem Zielpunkt x_s bewegt werden, wobei die Entfernung des Auftreffpunktes



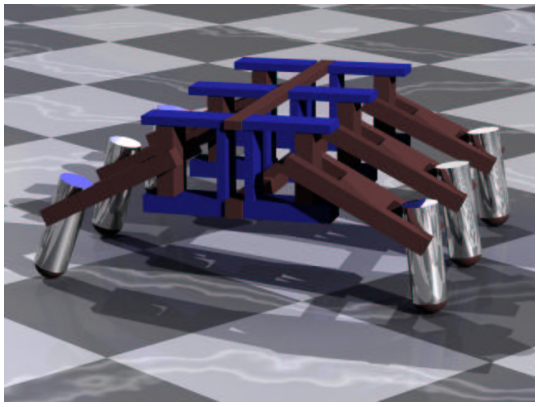


Abbildung 7.1: **Links:** Roboterbein mit drei DOF, aus einfachen geometrischen Objekten modelliert. Das Roboterbein wurde hier auf einen Sockel montiert. **Rechts:** Laufroboter mit 18 DOF, dessen Beine Kopien des links abgebildeten Beines sind.

des Beines auf dem Boden zum Zielpunkt x_s während der Bewegung aufsummiert wird und die Fitness des jeweiligen Kontrollprogramms darstellt. Im Bild ist die Anfangssituation des Roboterbein-Experimentes dargestellt. Das Bein besteht aus drei Rotationsgelenken. Die abgebildete Position ist die Grundstellung (alle Winkelgeschwindigkeiten sind gleich Null) und dient als Ausgangsposition für alle GP-Individuen.

Die Qualität einer Bewegung eines Beines kann nun durch folgende Funktion beschrieben werden:

$$fitness = \int_0^{t_{end}} \big(x(t) - x_s\big)^2 dt. \tag{7.2}$$

Die Tatsache, dass die Zeit in der Simulation in diskreten Schritten h fortschreitet, bedeutet, dass die Fitness tatsächlich berechnet wird durch

$$fitness = \frac{1}{N+1} \sum_{i=0}^{N} \big(x(t+i\Delta t) - x_s\big)^2, \text{ mit } t_{end} = t_0 + N\Delta t. \tag{7.3}$$

Die Fitness eines Individuums kann ebenso definiert werden als die Summe der quadratischen Abweichung des Auftreffpunktes auf der Ebene zu einem Punkt $x_s(t)$ auf einer im vorhinein definierten idealen Trajektorie, die die Sollbewegung des Beines darstellt. In den folgenden Experimenten soll zum Beispiel der Auftreffpunkt des Beines linear von einem Startpunkt zu einem Endpunkt verfahren werden, d.h. es gilt

$$x_s(t) = x_{start} + t\left(\frac{x_{ziel} - x_{start}}{t_{end} - t_0}\right), \text{ mit } t_0 \leq t \leq t_{end}. \tag{7.4}$$

Je näher an der Trajektorie die Bewegung des Beines liegt, desto besser (kleiner) ist die Fitness des Individuums. Es ist hierbei außerdem wichtig, dass das Bein aus dem Stillstand vom Startpunkt zum Endpunkt bewegt und wiederum zum Stillstand gebracht werden soll. Dies bedeutet eine Beschleunigung am Anfang der Bewegung, eine Bewegung mit konstanter Geschwindigkeit und eine abschließende Verzögerung des Beines bis zum Stillstand. Um zu gewährleisten, dass alle diese Phasen auch in die Evolution der Bewegung einfließen, wurde über einen Zeitraum von $t = 3s$ simuliert, lange genug, um selbst ein sehr langsam bewegtes Bein die Bewegung abschließen zu lassen. Die Parameter des GP-Systems sind im Koza-Tableau in Tabelle A.1 abgebildet.

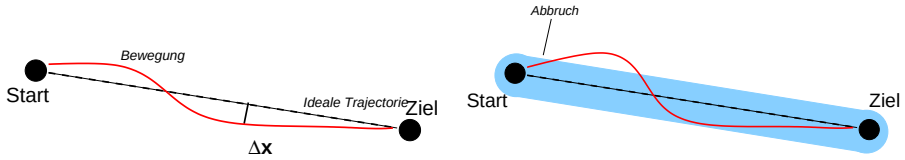


Abbildung 7.2: **Links:** Abstand zwischen der idealen Trajektorie und der Bewegung des Beines. **Rechts:** Vorzeitiger Abbruch der Evaluation bei Überschreiten des maximal erlaubten Abstandes zur optimalen Trajektorie.

Die Fitness einer evolvierten Bewegung hängt stark von der koordinierten Bewegung einzelner Glieder ab. Diese werden durch die zeitlich korrekt synchronisierte Angabe von Gelenkmomenten bewegt. Diese Gelenkmomente werden durch die Sequenz von Instruktionen eines GP-Individuums definiert. Wenn nun zu Beginn einer Bewegung ein Glied des Beines nicht optimal in das Bewegungsmuster des Beines passt, wird die Qualität der gesamten Bewegung schlechter sein, selbst wenn die Bewegung nach der fehlerhaften Instruktion korrekt sein sollte. Es ist also möglich, in einem sehr frühen Stadium der Evaluation des Individuums festzustellen, dass die Simulation abgebrochen werden kann (siehe Abbildung 7.2). Hierdurch kann sehr viel Rechenzeit gespart werden, denn die zur Verfügung stehende Rechenleistung kann nun auf die Simulation vielversprechender Individuen verteilt werden. Diese Methode ist unter dem Namen *„racing algorithm“* bekannt [114].

Die Fitness der Individuen, deren Auswertung zum Zeitpunkt t_{stop} abgebrochen wird, berechnet sich in Anlehnung an Gleichung (7.3) nach Gleichung (7.5) durch

$$fitness = \frac{t_{end}}{t_{stop}} \frac{1}{n+1} \sum_{i=0}^{n} \left(x(t+i\Delta t) - x_s\right)^2, \text{ mit } t_{stop} = t_0 + n\Delta t \text{ und } n < N. \tag{7.5}$$

Auf diese Weise erhalten Individuen, die über den kompletten Zeitraum simuliert werden, die gleiche Fitness wie in Gleichung (7.3). Individuen, deren Auswertung abgebrochen wird, bekommen einen von ihrer Simulationszeit abhängigen Straffaktor zugewiesen. Bei Individuen, die zum gleichen Zeitpunkt abgebrochen werden, entscheidet weiterhin die Qualität der Bewegung.

Bei den durchgeführten Experimenten ist eine Geschwindigkeitssteigerung von bis zu 700% erzielt worden[1]. Gemessen wurde die Anzahl der Auswertungen (bei identischen Startbedingungen), nach der ein definiertes Fitnessniveau erreicht wurde. Zu beachten ist, dass die Evolution zu Beginn der Experimente mit größerer Geschwindigkeit voranschreitet, da mehr schlechtere Individuen existieren, deren Simulation frühzeitig abgebrochen wird. Gegen Ende der Experimente, wenn die Individuen im Mittel besser geworden sind, werden sie auch später abgebrochen (wenn überhaupt noch), so dass die Geschwindigkeit der Konvergenz abnimmt. Es werden also im Verlaufe der Evolution immer weniger Individuum pro Zeiteinheit bewertet, die Auswertung der Individuen dauert dafür immer länger.

7.3 Variation der Parameter des Basisalgorithmus

Um die Performance des Grundalgorithmus zu erhöhen, sind die verwendeten Methoden hinsichtlich ihrer Verbesserbarkeit untersucht worden. Im Bereich der EA hängt die Geschwindigkeit des Algorithmus sehr stark von den Variationsoperatoren ab, die die Bewegung der Population im Suchraum steuern. Aus diesem Grund wurden Messungen und Experimente zur selbst-regulierenden Schrittweitensteuerung der Variationsoperatoren bei der Evaluation von Beinbewegungen durchgeführt.

[1] In den Experimenten in Kapitel 9 wurde sogar eine Geschwindigkeitssteigerung von 900% durch den vorzeitigen Abbruch der Simulation erreicht.

Ziel dieses Ansatzes ist es, die Zahl der Auswertungen zu reduzieren, indem Parameter des EA in einer Weise modifiziert werden, die es erlaubt, im Verlaufe der Evolution einen zu jedem Zeitpunkt optimalen Fortschritt zum Optimum zu erzielen. Zu diesen Parametern gehören zum Beispiel die Mutations- und Rekombinationsrate oder die Populationsgröße. Die zu Grunde liegende Annahme ist, dass zu jedem Zeitpunkt der Evolution unterschiedliche optimale Einstellungen der Parameter existieren. So ist es in manchen Situationen der Evolution vorteilhaft, neue Bereiche des Suchraums zu untersuchen (*Diversifikation*), um nicht in lokalen Optima zu stagnieren, wohingegen während anderer Phasen der Evolution eine verstärkte Konzentration der Suche auf das lokale Umfeld (*Intensivierung*) zu besseren Resultaten führt. Eine zu jedem Zeitpunkt optimale Einstellung der Parameter kann so zu einer potenziellen Verbesserung der Performance führen gegenüber der zu erwartenden Performance mit konventionellen, statischen Einstellungen über die gesamte Laufzeit.

Die positiven Ergebnisse bei Evolutionsstrategien ([140, 151]) mit adaptiven Parametern hat zu verstärktem Interesse an diesem Gebiet im Bereich der Genetischen Algorithmen ([69, 79]) geführt. Schon frühe Arbeiten ([51, 70]) haben sich mit diesem Ansatz beschäftigt, um empirisch optimale Einstellungen zu ermitteln. Übersichtsartikel zu diesem Thema (z.B. [56, 77]) unterteilen die Ansätze in statische Varianten und drei Klassen von dynamischer Anpassung der Parameter, wobei statische Varianten keine Veränderung der Parameter vornehmen und nur der Vollständigkeit halber erwähnt werden:

Deterministisch Deterministische Methoden ändern die Einstellungen der Parameter in Abhängigkeit von der Zeit. Ein klassisches Beispiel hierfür ist Simulated Annealing [102]. Die Evolution wird durch eine stetig sinkende Temperatur gesteuert: bessere Nachkommen ersetzen die Eltern, schlechtere Nachkommen ersetzen die Eltern mit einer temperaturabhängigen Wahrscheinlichkeit.

Adaptiv Bei adaptiven Strategien werden die Parameter anhand einer in der Vergangenheit gemessenen Performance verändert. Die Länge des beobachteten Intervalls ist hierbei ein zusätzlicher Parameter und wird *adaptation window* [50] oder *epoch* [78] genannt. Die Performance wird entweder durch den erreichten Fortschritt oder die Qualität der Individuen seit der letzten Beobachtung bzw. Variation beschrieben (siehe z.B. [50, 125]). Diese Methode kann sich als problematisch erweisen, denn nur direkte Auswirkungen können auf diese Weise gemessen und bewertet werden, verzögerte Effekte haben keinen Einfluss und bleiben unberücksichtigt. In [26] und [174] ist eine detaillierte Kritik zu dieser Problematik aufgeführt. Manche Autoren versuchen, dieses Problem zu umgehen, indem die Auswirkungen vergangener Parameterveränderungen mit in Betracht gezogen werden (z.B. in [50]). Die adaptive Steuerung der Parametereinstellungen wird auch *exogen* genannt.

Selbst-Adaptiv Die Ausdrücke *selbst-adaptiv* oder *endogen* können synonym verwendet werden. Diese Methode basiert auf der Idee, den aktuellen Zustand des evolutionären Prozesses vom Fortschritt des Algorithmus in der Vergangenheit abzuleiten (analog zu den adaptiven Methoden). Wenn die einzustellenden Parameter gleichzeitig auch Teil des evolutionären Prozesses sind, spricht man von selbstadaptiven Methoden. Ein sehr gutes Beispiel hierfür sind die parallel evolvierten Strategieparameter bei den Evolutionsstrategien.

Statisch Die Einstellungen für die Parameter werden vor Beginn des Laufes festgelegt und nicht verändert. Die Performance des Algorithmus ist direkt abhängig von der Auswahl der Werte.

Die Autoren von [56, 77] unterscheiden außerdem zwischen zwei Hauptformen der Parameteranpassung: (i) *parameter tuning* vor dem eigentlichen Experiment (wie sie z.B. in [51, 70] verwendet wird) und *parameter control* während des Experimentes, welches mit bestimmten Initialwerten der einzelnen Parameter begonnen wird. In [56, 77] wird eine umfassende Beschreibung der Arbeiten zu diesem Thema gegeben.

Ein anderer Übersichtsartikel klassifiziert die Arbeiten auf diesem Gebiet nach anderen Gesichtspunkten:

Adaptive Repräsentationen Hierbei wird die interne Darstellung der Individuen verändert, um die Performance des GA zu verbessern. Beispiele hierfür sind in [152] und [159] zu finden.

Adaptive Operatoren Die Adaptation der Operatoren bedeutet, dass sich die Funktionsweise des Operators während der Evolution verändert. In [158, 159, 160] werden Beispiele hierfür gegeben.

Adaptive Parameter Die meisten aller Ansätze verwenden diese Art der Adaptation um die Performance des Algorithmus zu optimieren. Zu den variierten Parametern gehören zum Beispiel die Populationsgröße sowie die Crossover- und Mutationswahrscheinlichkeit.

Eine weitere Unterscheidung der Ansätze ist in [22] zu finden, wobei hier die Unterscheidung durch die Ebene der Adaptation erfolgt. Es wird zwischen *population-level*, *individual-level* und *component-level* unterschieden. Alle zuvor aufgelisteten Klassifikationsmethoden können zusätzlich noch nach diesem Schema klassifiziert werden. Ein weiterer Übersichtsartikel [27] klassifiziert die aufgelisteten Ansätze in folgende Kategorien:

- Ansätze, die die Anpassung der Populationsgröße mit den Operatorwahrscheinlichkeiten kombinieren (z.B. [28, 70, 74]).
- Ansätze, die nur die Anpassung der Populationsgröße untersuchen (z.B. [59, 121, 124]).
- Untersuchungen, die sich nur auf den Einfluss der Operatorwahrscheinlichkeiten beschränken (z.B. [25, 95, 96, 170]).
- Untersuchungen, die eine Kombination von variablen Operatoren (Funktion) und variablen Wahrscheinlichkeiten (Anwendungshäufigkeit) betrachten (z.B. [83]).

Es existieren weitere Modelle, die versuchen, die Anpassung des Algorithmus über Meta-Ebenen innerhalb des GA zu erreichen [62, 171], andere verwenden hierfür Fuzzy-Regelbasen [76]. Für viele Anwendungen ist versucht worden, die Operatorwahrscheinlichkeiten anzupassen, zum Einen als ein Testproblem, zum Anderen als Motivation für eingehende Untersuchungen (z.B. in [157]). Es existieren auch theoretische Ansätze, die die Auswirkungen variabler Parameterwerte analysieren (z.B. in [20, 59, 162]). Dort wird meist anhand von einfachen EA und künstlichen Spielproblemen gezeigt, dass nicht-statische Parametereinstellungen signifikant bessere Ergebnisse (gemessen an der Konvergenzgeschwindigkeit) erzielen als mit den besten statischen Werten erreichbar sind.

Es existieren zu diesem Thema im Bereich GA eine ganze Reihe von Arbeiten, die jedoch kein geschlossenes Bild ergeben. Die Vielzahl der verschiedenen Algorithmen, die unterschiedlichen variablen Teile und verschiedene Testprobleme erlauben es den Autoren nicht, allgemeine und eindeutige Aussagen über die positive Wirkung einer speziellen Methode oder Variante zu machen.

7.3.1 Alternative Ansätze

Es existieren noch andere Ansatzmöglichkeiten, die Leistung der EA zu verbessern, die sich nicht auf die Adaptation von Parametern beschränken. Hierzu zählen zum Beispiel das *Stochastic Sampling* [131, 132], das Verwendung findet, wenn große Datensätze als Trainingsmengen verwendet werden. Mit Hilfe des Stochastic Sampling werden dann zufällige Teilmengen für die Evolution verwendet, so dass immer nur ein Bruchteil des gesamten Datensatzes evaluiert werden muss. Dies spart Laufzeit bei der Auswertung jedes Individuums.

Die Reduzierung der Trainigsdatensätze ist auch das Ziel des *Dynamic Subset Selection*, bei dem die Teilmengen nach Alter und Schwierigkeitsgrad zusammengestellt werden—im Gegensatz zum Stochastic Sampling, bei dem dies zufällig erfolgt. Daten werden mit Priorität in die Trainingsmenge eingefügt, wenn sie schon länger nicht mehr für die Evaluation von Individuuen eingesetzt wurden und wenn sie als ”schwierig” eingestuft worden sind, z.B. falsch klassifizierte Datensätze bei einem Klassifikationsproblem. Andere

Arbeiten unterteilen die Population in mehr oder weniger voneinander getrennte Teilpopulationen (*Demes* [21]), um den Suchraum effizienter durchsuchen zu können.

Trotz der Tatsache, dass adaptive Methoden der Parametervariation seit längerer Zeit in GA, ES und EP analysiert und verwendet werden, existieren nur wenige Arbeiten zu diesem Thema im Bereich GP. Es wurden zum Beispiel die Auswirkungen der Adaptation der internen Operaturstrukturen analysiert [23, 33, 83, 98, 125] und auch der Effekt von variablen Repräsentationen [24, 113, 145, 144] ist Thema von Veröffentlichungen gewesen. Die Zahl der Veröffentlichungen bleibt weit hinter der in anderen Bereichen der EA zurück und auch hier ist das Ergebnis der Untersuchungen nicht eindeutig. Ein unzweideutig positiver Effekt kann von keinem Autor nachgewiesen werden, zumindest aber werden überdurchschnittlich erfolgreiche Experimente mit in allen anderen Fällen durchschnittlich leistungsfähigen Algorithmen gezeigt.

In den folgenden Abschnitten wird eine Serie von Experimenten präsentiert, mit der die Möglichkeit untersucht werden soll, mit Hilfe von adaptiven Elementen die Zahl der Fitnessauswertungen in der Genetischen Programmierung soweit wie möglich zu reduzieren. Hierfür wird der neue, erweiterte GP-Algorithmus an drei Benchmarkproblemen aus unterschiedlichen Problemdomänen getestet und die Ergebnisse präsentiert.

7.3.2 Experimente mit drei verschiedenen Testproblemen

Anhand dreier Testprobleme aus unterschiedlichen Problemdomänen soll der Einfluss von nicht-statischen Parametereinstellungen auf GP untersucht werden. Die verwendeten Probleme sind (i) ein Klassifikationsproblem (*Ringe*), eine Funktionsregression (*Sinus*) und das Problem, ein Laufprogramm für einen (simulierten) sechsbeinigen Laufroboter zu finden (*Hexapod*). Um die Unabhängigkeit der erzielten Resultate von speziellen Problemeigenschaften möglichst zu gewährleisten, sind sie aus drei unterschiedlichen Bereichen gewählt worden.

Die Testprobleme

Ringe ist ein Klassifikationsproblem, das in [36] detailliert beschrieben wird. Es müssen Punkte aus einem dreidimensionalen Raum zwei Klassen zugeordnet werden. Die Klassen formen zwei verkettete Ringe im Raum (Abb 7.3). Die Fitness eines Individuums ist definiert als die Anzahl der falsch klassifizierten Punkte. Dieser Wert muss im Verlaufe der Evolution minimiert werden.

Sinus ist ein Regressionsproblem. Eine Sinuskurve soll mit einer Funktion im Intervall $[0, 2\pi]$ approximiert werden, wobei nur die grundlegenden arithmetischen Operationen $(+, -, \cdot, /)$ zur Verfügung stehen. Die Fitness eines Individuums ist hierbei definiert als die Summe der quadratischen Abweichung an zwanzig zufällig gewählten Punkten in $[0, 2\pi]$. Dieser Wert muss im Verlaufe der Evolution minimiert werden.

Hexapod ist das Problem, ein Robotersteuerungsprogramm für einen simulierten sechsbeinigen Roboter zu evolvieren. Bei diesem Problem besteht die Operatormenge aus den drei Befehlen *swing, stance* und *wait*. Die Befehle haben folgende Bedeutung:

- *swing(x)*: Das Bein *x* wird angehoben, noch vorne geschwenkt und dann wieder abgesenkt. Diese Bewegung dauert t_{sw} Millisekunden. Dieser Abschnitt der Bewegung eines Beines wird Schwingphase genannt.
- *stance(x)*: Das Bein *x* wird auf der Bodenebene nach hinten geschwenkt. Dadurch wird der Körper nach vorne bewegt. Diese Bewegung dauert t_{st} Millisekunden. Dieser Bewegungsabschnitt heißt Stemmphase.
- *wait(n)*: Wartet *n* Millisekunden.

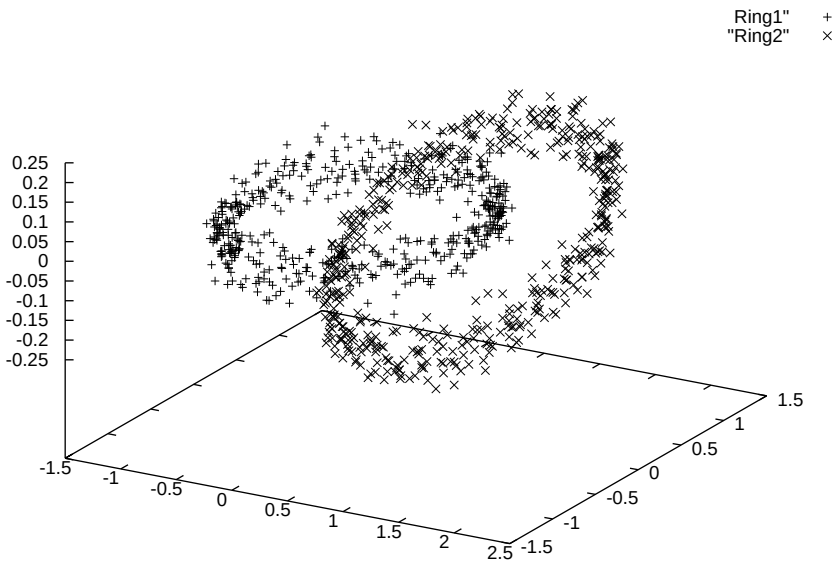


Abbildung 7.3: *Ringe*: Die Punkte bilden zwei verkettete Ringe im dreidimensionalen Raum, die sich nicht berühren (d.h. alle Punkte sind eindeutig zuzuordnen). Alle 1000 Punkte der Ringe haben eine Entfernung vom Mittelpunkt, die normalverteilt ist mit einem Mittelwert von 1.0 (dem mittleren Radius aller Punkte des Umkreises) und einer Varianz von 0.04.

Mit diesem Befehlssatz kann GP Steuerungsprogramme formulieren, die in der Lage sind, einen sechsbeinigen Laufroboter zu bewegen. Die generelle Fähigkeit von GP, autonome Roboter mit unterschiedlicher Zielsetzung zu steuern, wird z.B. in [71, 72, 130, 134] gezeigt. Speziell für Laufroboter ist die erfolgreiche Anwendung von GP in z.B. [53, 111, 161] demonstriert. Es ist aus der Biologie bekannt (siehe z.B. [47]), dass sechsbeinige Insekten eine spezielle Art der Fortbewegung bevorzugt anwenden, den sogenannten *tripod gait*. Hierbei werden jeweils drei Beine synchron bewegt, wobei drei Beine auf dem Boden bleiben, nach hinten schwingen (z.B. die Beine 1,4 und 5 in Abb. 7.4 rechts) und dabei den Körper nach vorne bewegen und gleichzeitig die anderen Beine (2,3 und 6) in der Luft nach vorne schwingen, um in der nächsten Phase dann die Bewegung der Beine 1,4 und 5 durchzuführen. Auf diese Weise stehen immer drei Beine auf dem Boden, das Insekt steht zu jedem Zeitpunkt stabil. Ein Laufprogramm ist eine Sequenz von *swing*, *stance* und *wait*-Instruktionen. Diese Folge der Instruktionen kann dargestellt werden als eine Reihe von Zustandsübergängen in einem Mealy-Automaten (siehe Abbildung 7.4). Die Fitness eines Steuerungsprogrammes wird definiert als der Abstand zwischen der Sequenz, die den *tripod gait* repräsentiert und eine bestimmte Folge von Zustandsübergängen im Automaten verursacht, und dem aktuellen Programm. Dieser Abstand wird gemessen mit Hilfe von gewichteten Kanten im Mealy-Automaten. Die in Abb. 7.4 dargestellten Kanten haben alle den Wert 0 (alle anderen Kanten, von jedem Zustand s_i in alle anderen Zustände s_j, mit $j \neq i$ und $j \neq j+1$, die in der Abb. nicht dargestellt sind, haben den Wert 1). Alle Programme werden nun für eine bestimmte Zeit zyklisch ausgeführt, alle Programme starten in Zustand 0. Ein nicht optimales Programm (ein Programm, das keinen *tripod gait* ausführt) wird zwangsläufig von einem Zustand s_i zu einem Zustand s_{i+k} über eine Kante mit dem Gewicht 1 gelangen. Die Gewichte aller Transitionen während der Ausführung werden summiert und bilden die Fitness des Individuums. Dieser Wert muss im Verlaufe der Evolution minimiert werden.

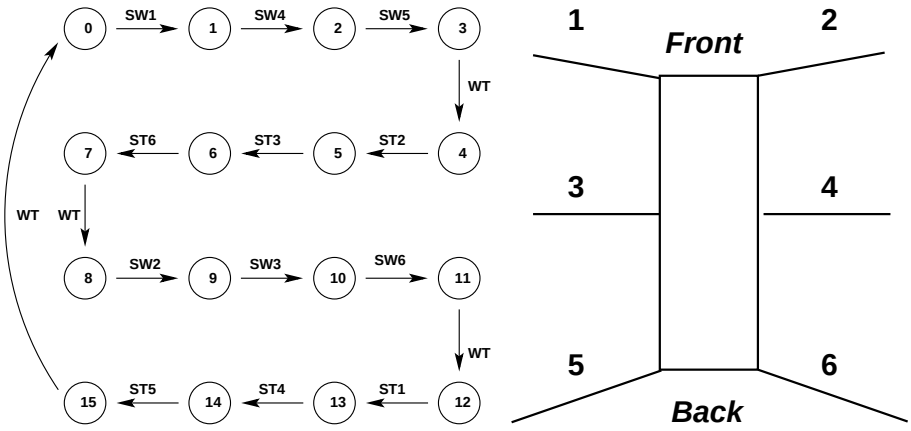


Abbildung 7.4: **Links:** *Hexapod*: Vereinfachtes Darstellung des *tripod gait* eines sechbeinigen Roboters. Die Laufsequenz ist als ein Mealy-Automat [81] mit 16 Zuständen abgebildet. Abkürzungen: ST=Stemmphase (*stance*), SW=Schwingphase (*swing*), WT=Warten (*wait*). **Rechts:** Schematische Darstellung eines Sechsbeiners. Die Beinnummern entsprechen den Übergängen im Mealy-Automaten.

Der verwendete Adaptationsmechanismus im GP-System

Das für diese Voruntersuchungen verwendete GP-System ist das am Lehrstuhl für Systemanalyse entwickelte SYSGP [37]. Es war notwendig, kleine Veränderungen vorzunehmen, um die Parameteradaptation durchführen zu können. Als Repräsentation wurde lineares GP gewählt, wie in allen anderen Experimenten auch, um Ergebnisse leichter übertragbar und vergleichbar zu gestalten.

Es wurden zwei Methoden der Parameteranpassung an den drei Testproblemen untersucht, die deterministische und die adaptive Variation. Die Parameter sind in allen Experimenten nach einer konstanten Anzahl von Generationen variiert worden, das *adaptation window* beträgt in diesem Fall 10 Generationen. Deterministische Veränderungen sind rein zeitabhängig, adaptive Variationen erfordern es, den Fortschritt innerhalb des *adaptation window* zu messen. Dies erfolgt hier global, d.h. es wird nicht nach einzelnen Operatoren getrennt gemessen. Die aktuellen Mutationswahrscheinlichkeiten zum Zeitpunkt t für die deterministische Variante lassen sich nun nach Gleichung (7.6) bestimmen

$$m(t) = m_0 \cdot a_m^{k_m \cdot t}, \tag{7.6}$$

wobei m_0 die Mutationswahrscheinlichkeit zu Beginn $(t = 0)$ des Experimentes angibt. Die Größe der Änderung wird durch $a_m \in [0,1]$ angegeben und $k_m \in \{-1,0,1\}$ beschreibt die Richtung der Anpassung.

Beispiel Ist $m_0 = 0.1$, so sieht der Verlauf der Mutationswahrscheinlichkeit nach Gleichung (7.6) aus wie in Abb. 7.5. Die Geschwindigkeit a_m und die Richtung der Änderung k_m prägen den Verlauf der Kurven maßgeblich. Ist $k_m = 0$, so ändert sich die Mutationswahrscheinlichkeit nicht (statischer Fall). Die Wahrscheinlichkeit des Crossoveroperators berechnet sich analog zu Gleichung (7.6) durch

$$c(t) = c_0 \cdot a_c^{k_c \cdot t}. \tag{7.7}$$

Bei der deterministischen Strategie sind $m(t)$ und $c(t)$ nur monoton änderbar, wohingegen bei der adaptiven Strategie die Parameter nach Gleichung (7.8) auch nicht monoton verändert werden können.

$$m(t) = m_0 \cdot a_m^{k_m \cdot (s(t) - f(t))}, \tag{7.8}$$

wobei $s(t)$ die Anzahl der Verbesserungen, $f(t)$ die Anzahl der Verschlechterungen im beobachteten Intervall angibt. Die Crossoverwahrscheinlichkeit ist analog in Gleichung (7.9)

$$c(t) = c_0 \cdot c_m^{k_m \cdot (s(t) - f(t))} \tag{7.9}$$

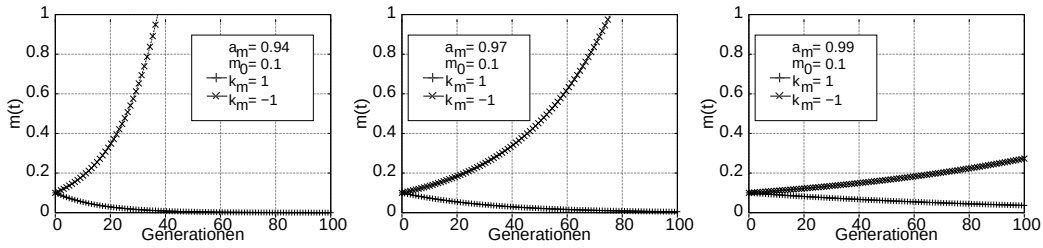


Abbildung 7.5: Verlauf der Mutationswahrscheinlichkeiten $m(t)$ nach Gleichung (7.6). **Links:** $a_m = 0.94$, **Mitte:** $a_m = 0.97$, **Rechts:** $a_m = 0.99$.

Typ	Parameter	Beschreibung
A	$k_c = +1, k_m = +1$	Die Crossoverwahrscheinlichkeit und die Mutationswahrscheinlichkeit werden verkleinert.
B	$k_c = +1, k_m = -1$	Die Crossoverwahrscheinlichkeit sinkt, die Mutationswahrscheinlichkeit steigt.
C	$k_c = -1, k_m = +1$	Die Crossoverwahrscheinlichkeit steigt, die Mutationswahrscheinlichkeit wird verringert.
D	$k_c = -1, k_m = -1$	Beide Wahrscheinlichkeiten werden vergrößert.

Tabelle 7.1: Die vier verschiedenen Typen der Parameteranpassung. Die Anpassung erfolgt jeweils deterministisch oder adaptiv und mit einer bestimmten Geschwindigkeit a_m bzw. a_c.

.

gegeben. Mutations- und Crossoverwahrscheinlichkeit sind auf das Intervall $[0,1]$ beschränkt. Die formale Notation der deterministischen und adaptiven Wahrscheinlichkeitsänderungen nach den Gleichungen (7.6), (7.7), (7.8) und (7.9) ist gleich der selbst-adaptiven Schrittweitenkontrolle bei den Evolutionsstrategien. Der einzige Unterschied besteht darin, dass bei den ES die Exponenten normalverteilte Zufallsvariablen mit adaptierter Varianz [151], in diesem Falle aber entweder statische Werte (deterministische Variante) oder gemessene Erfolgsraten (adaptive Strategie) sind.

Mit Hilfe der Parameter k_m und k_c ist es möglich, neun verschiedene Strategien zur Parameteradaptation zu beschreiben, wobei nur die vier Varianten mit $k_m \neq 0 \wedge k_c \neq 0$ im Folgenden näher betrachtet werden. Alle anderen Varianten, bei denen immer mindestens eine Operatorwahrscheinlichkeit konstant ist, werden hier nicht weiter betrachtet. Die betrachteten Varianten A-D sind in Tabelle 7.1 aufgeführt und näher erläutert.

Strategien und auftretende Probleme

Um entscheiden zu können, ob eine nicht-statische Parametereinstellung die Anzahl der notwendigen Fitnessauswertungen reduziert, ist es notwendig, die durchschnittliche Anzahl der Auswertungen bis zum Erreichen einer vorher definierten Schwelle zu messen. Wenn eine nicht-statische Methode signifikant weniger Auswertungen benötigt, kann sie als besser betrachtet werden. Eine andere Methode, die Qualität einer nicht-statischen Variante zu messen, ist die Fitness nach einer konstanten Anzahl von Auswertungen zu vergleichen. Die letztere Methode ist der ersteren vorzuziehen, da nicht immer garantiert ist, dass ein evolutionärer Prozess eine definierte Fitness in endlicher Zeit erreicht.

	Anzahl Änderungen	Geschwindigkeit	a_m, a_c	c_{min}	c_{max}	m_{min}	m_{max}
Ringe/Sinus	50	schnell	0.93	0.70	1.00	0.70	1.00
	50	langsam	0.97	0.70	1.00	0.70	1.00
Hexapod	100	schnell	0.93	0.50	1.00	0.20	1.00
	100	langsam	0.97	0.50	1.00	0.20	1.00

Tabelle 7.2: Ergebnisse der Voruntersuchungen. Aufgelistet sind die Anzahl der Anpassungen, die Stärke sowie die minimalen und maximalen Werte für $m(t)$ und $c(t)$ für das jeweilige Testproblem.

Die in Abschnitt 7.3.2 aufgeführten Testprobleme besitzen unterschiedliche Schwierigkeitsgrade, so dass bei der Wahl z.B. der Populationsgröße und der maximalen Anzahl von Generationen auf einen vergleichbaren Gesamtrechenzeitverbrauch pro Experiment bei allen drei Problemstellungen geachtet werden muss. Bevor mit den nicht-statischen Experimenten begonnen werden kann, ist es notwendig, genaue Werte der Häufigkeit und der Stärke der Veränderungen zu ermitteln. Geeignete Werte hierfür sind in Vorexperimenten bestimmt worden und in Tabelle 7.2 aufgeführt. Diese Werte bleiben für alle weiteren Experimente konstant. Parameteranpassungen werden nach den Gleichungen (7.6) und (7.7) im deterministischen Fall und nach den Gleichungen (7.8) und (7.9) im adaptiven Fall vorgenommen, wobei jeweils zwei verschiedene Werte für a_m bzw. a_c untersucht wurden (siehe Tabelle 7.2), um den Einfluss der Anpassungsgeschwindigkeit zu untersuchen. Die Wahrscheinlichkeiten nach (7.6), (7.7), (7.8) und (7.9) nehmen unter Umständen Werte an, die zu signifikant schlechteren Ergebnissen geführt haben (z.B $m(t) = 0.0$, d.h. es findet keine Mutation mehr statt), so dass bestimmte Intervalle definiert wurden, in denen sich die Operatorwahrscheinlichkeiten zu bewegen haben (siehe $c_{min}, c_{max}, m_{min}, m_{max}$ in Tabelle 7.2). Alle weiteren Parameter der GP-Experimente sind in Tabelle A.2 aufgeführt.

Resultate

In diesem Abschnitt werden die Ergebnisse der Experimente mit den oben beschriebenen Einstellungen präsentiert. Zuerst allerdings ist es für die objektive Bewertung der Ergebnisse notwendig, Qualitätskriterien zu definieren, die es erlauben, Ergebnisse verschiedener Experimente miteinander zu vergleichen. Hierfür wurden folgende Kriterien aufgestellt:

Eine Strategie mit nicht-statischen Parametern ist genau dann als besser einzustufen, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

(1) Der Mittelwert der Fitness einer nicht-statischen Variante mit optimalen Startwerten für Mutations- und Crossoverwahrscheinlichkeit ist besser als der Mittelwert der Fitness der besten statischen Variante nach einer konstanten Zahl von Auswertungen.

(2) wie (1), nur mit der Bedingung, dass bei gleichem Mittelwert die nicht-statische Variante eine kleinere Varianz besitzt.

(3) Der Mittelwert der Fitness einer nicht-statischen Variante ist besser als der Mittelwert der statischen Variante bei zufälliger Initialisierung der Mutations- und Crossoverwahrscheinlichkeiten[2] nach einer konstanten Zahl von Auswertungen.

(4) wie in (2), nur mit der Bedingung, dass bei gleichem Mittelwert die nicht-statische Variante eine kleinere Varianz besitzt.

[2]Wobei die in Tabelle 7.2 angegebenen Werte die obere und untere Grenze des Intervalls für die Mutations- und Crossoverwahrscheinlichkeit angeben, aus denen dann gleichverteilt m_0 und c_0 bestimmt werden

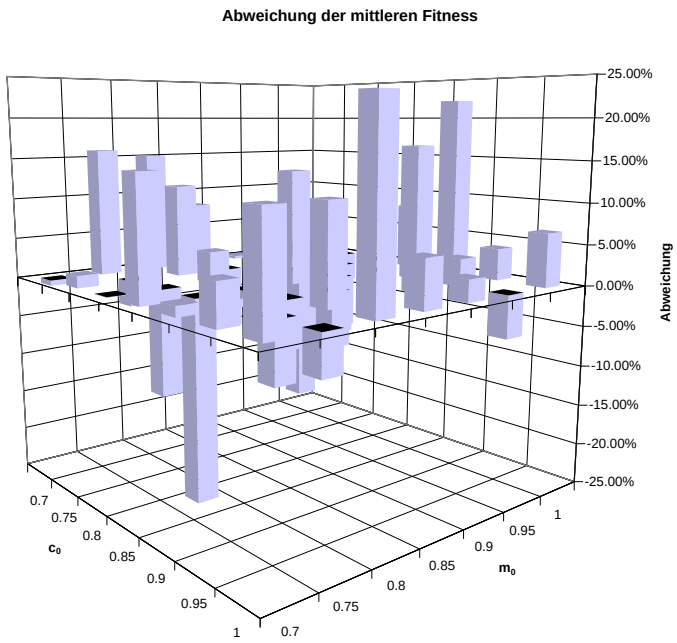


Abbildung 7.6: Stochastische Effekte: Abweichung der mittleren Fitnesswerte von zwei Serien der statischen Variante mit je 200 Experimenten mit unterschiedlichen Seed-Werten des *Ringe*-Problems. Die durchschnittliche Abweichung der mittleren Fitness zwischen beiden Serien beträgt 8.42%, es werden aber auch Abweichungen von bis zu 24% gemessen.

Wenn die Bedingungen (1) oder (3) erfüllt sind, ist das Ergebnis eine verbesserte Qualität der Ergebnisse bei gleicher Anzahl von Auswertungen, wobei die Bedingung (3) eine noch stärkere Aussage bzgl. der Güte der entsprechenden nicht-statischen Variante zulässt. Sind die Bedingungen (2) oder (4) erfüllt, so kann man die nicht-statische Variante als robuster bezeichnen, da die zu erwartende Fitness im Mittel näher an der besten zu erwartenden Fitness nach einer konstanten Zahl von Auswertungen liegt. Ist keine der Bedingungen eindeutig erfüllt, so kann keine positive Aussage bzgl. einer nicht-statischen Variante gemacht werden.

Als Grundlage aller Vergleiche der verschiedenen Varianten dient der Vergleich mit der ”Null-Alternative”, dem statischen Fall. Um gute Vergleichsdaten zu erhalten, wurde für alle Kombinationen von m_0- und c_0-Werten für die initiale Mutations- und Crossoverwahrscheinlichkeit (von 0.7 bis 1.0 in Schritten von 0.05 für das Ringe-Problem bzw. in Schritten von 0.1 für das Sinus- und Hexapod-Problem) zwischen der unteren und der oberen Grenze des in Tabelle 7.2 definierten Intervalls eine Serie von 200 Experimenten mit unterschiedlichen Seed-Werten des Zufallszahlengenerators durchgeführt.

7.3.3 Vergleich von statischen und nicht-statischen Varianten

Für die Durchführung des Vergleichs werden die Parameter $m(t)$ und $c(t)$ gemäß der in Tabelle 7.1 aufgeführten vier Typen A-D deterministisch oder adaptiv mit den in Tabelle 7.2 angegebenen Geschwindigkeiten verändert und der Durchschnitt aus 200 Läufen gebildet. Grundlage der Bewertung der einzelnen Varianten A-D ist der Vergleich mit der durchschnittlichen Fitness aus 200 Experimenten mit statischen Werten $(m(t) = m_0, c(t) = c_0)$. In Abbildung 7.6 wird deutlich, dass bei der Ermittlung der Vergleichsdaten ein problematischer Effekt auftritt: Die durchschnittliche Abweichung zweier Experimentserien der statischen Variante von ca. 8% bedeutet einen erheblichen Unsicherheitsfaktor bei der Bewertung der nicht-statischen Varianten. Im Einzelfall sind sogar Abweichungen von mehr als 20% gemessen worden. Sollte

		Ringe		Sinus		Hexapod	
		Fitness	Varianz	Fitness	Varianz	Fitness	Varianz
Statisch		84.34	8.33	7.06	0.009	3.58	0.172
deterministisch, $a_m = a_c = 0.97$	A:	85.33	17.44	7.07	0.012	-	-
	B:	84.30	16.76	6.99	0.010	-	-
	C:	85.05	18.78	7.06	0.009	-	-
	D:	83.88	17.72	7.07	0.005	-	-
deterministisch, $a_m = a_c = 0.93$	A:	86.94	3.95	7.13	0.012	3.75	0.012
	B:	84.76	5.71	7.02	0.007	2.90	0.014
	C:	85.05	3.86	7.10	0.010	4.80	0.044
	D:	83.93	2.99	7.11	0.004	4.24	0.035
adaptiv, $a_m = a_c = 0.97$	A:	84.34	2.74	7.06	0.012	-	-
	B:	86.20	3.53	7.01	0.007	-	-
	C:	85.01	3.90	7.09	0.013	-	-
	D:	87.35	4.61	7.13	0.005	-	-
adaptiv, $a_m = a_c = 0.93$	A:	84.77	3.51	7.13	0.007	3.59	0.016
	B:	85.73	5.74	7.03	0.004	3.71	0.016
	C:	85.55	2.99	7.12	0.004	3.29	0.034
	D:	87.80	2.35	7.09	0.004	3.78	0.030

Tabelle 7.3: Resultate der Experimente. Mittlere Fitness und Varianz der 200 Experimente sind aufgeführt. Zum Vergleich ist die Leistung der statischen Variante dargestellt. Die Varianzen der schnellen deterministischen Varianten sind beim Ringe-Problem deutlich größer, beim Hexapod-Problem sind die Varianzen durchweg deutlich kleiner als im statischen Fall. Die mittleren Fitnesswerte unterscheiden sich kaum, bis auf die schnelle deterministische Variante B beim Hexapod-Problem, die eine deutlich bessere Fitness aufweist. Die langsamen Varianten sind aufgrund des hohen Rechenzeitaufwandes für das Hexapod-Problem nicht untersucht worden.

der positive oder auch negative Effekt der anderen Varianten kleiner als die Fluktuationen in der Vergleichsmessung sein, so ist es wahrscheinlich, dass er übersehen oder falsch interpretiert wird.

Die aufgeführten Serien von Experimenten benötigten eine immense Rechenzeit: Um alle Kombinationen von m_0 und c_0-Startwerten innerhalb der in Tabelle 7.2 angegebenen Intervalle für $a_m = a_c = 0.93$ und für $a_m = a_c = 0.97$ zu bewerten, müssen 162 Läufe durchgeführt werden. Um die statische Variante und jede der vier nicht-statischen Varianten je 200 mal zu bewerten, sind pro Variante 32.400 Läufe und eine Gesamtzahl von 162.000 Läufen notwendig. Die zeitliche Dauer eines einzelnen Experimentes ist bei den Testproblemen unterschiedlich und hängt von der Last des Rechenclusters ab. Durchschnittlich kann mit einem Zeitbedarf von ca. 1h für ein Experiment gerechnet werden[3].

Die Varianten mit unterschiedlichen Geschwindigkeiten bei der Anpassung der Mutations- und Crossoverwahrscheinlichkeiten sind hier nicht bewertet worden. Die Ergebnisse der Experimente sind für jede Variante A-D in Tabelle 7.3 aufgeführt. Für das Ringe-Problem sind die prozentualen Abweichungen der durchschnittlichen Fitness der adaptiven Varianten von den durchschnittlichen Fitnesswerten der statischen Variante für jede Kombination von Startwerten von Mutations- und Crossoverwahrscheinlichkeit in Abbildung 7.7 dargestellt. Abbildung 7.8 zeigt die deterministischen Varianten. Die prozentuale Abweichung berechnet sich durch

$$x(m_0, c_0) = \frac{f_{statisch} - f_V}{f_{statisch}}, \text{ mit } V = A, B, C, D. \tag{7.10}$$

Die Darstellung der Ergebnisse der Experimente zum Sinus- und zum Hexapod-Problem befindet sich in

[3] Auf einer Sun UltraSparc mit 400 MHz.

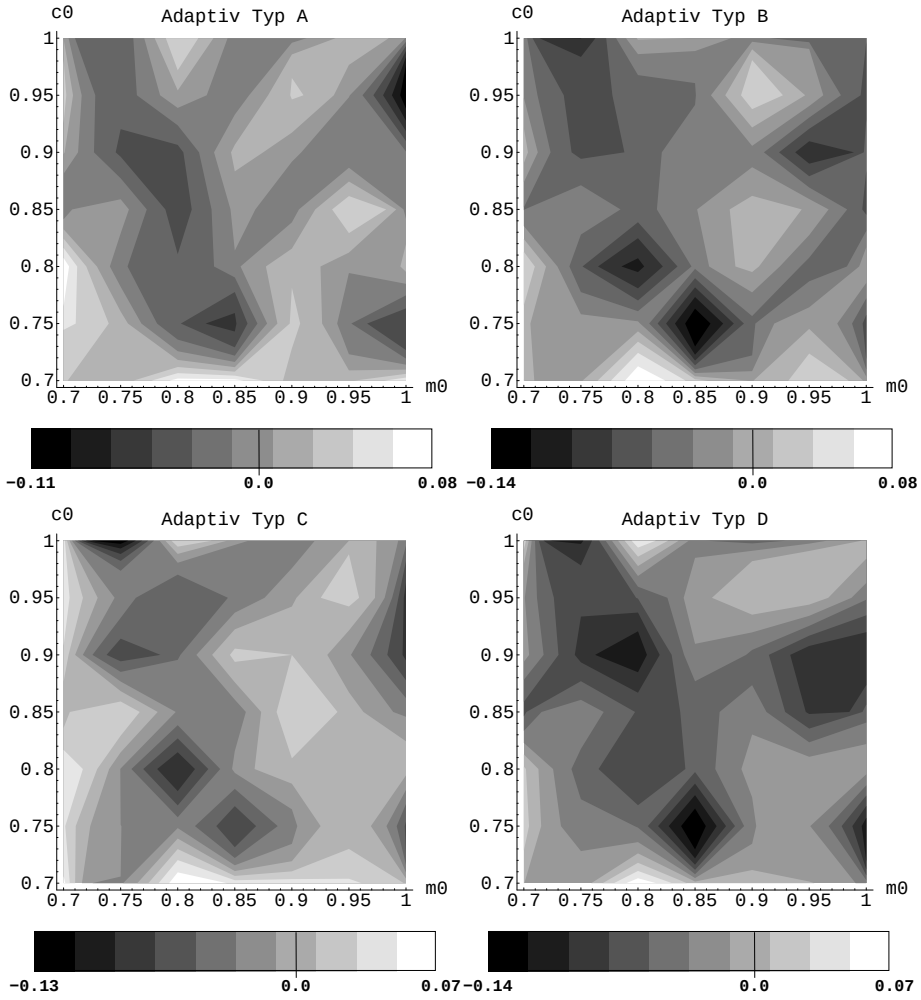


Abbildung 7.7: **Ringe:** Die durchschnittliche Fitness von 200 Experimenten mit adaptiver Anpassung Typ A-D im Verhältnis zur statischen Variante. Helle Bereiche zeigen Verbesserungen, dunkle Bereiche bedeuten Verschlechterungen gegenüber der statischen Variante. Die Legende verdeutlicht den prozentualen Unterschied. $a_m = a_c = 0.93$

Anhang A in den Abbildungen A.1- A.4.

Es wird anhand der Abbildungen und der Ergebnisse in Tabelle 7.3 deutlich, dass es auf den ersten Blick schwierig ist, zu bewerten, ob eine der nicht-statischen Varianten die in Abschnitt 7.3.2 erwähnten Bedingungen erfüllt.

Die Graustufendarstellung der Performance der einzelnen Varianten bezüglich der statischen Varianten in den Abbildungen 7.7 und 7.8 für das Ringe-Problem sowie in den Abbildungen A.1 bis A.4 in Anhang A für das Sinus- und Hexapod-Problem machen deutlich, dass sowohl Verbesserungen als auch Verschlechterungen erzielt werden. Zusätzlich kommt erschwerend hinzu, dass nicht auf den ersten Blick erkennbar wird, ob bestimmte Parameterkombinationen eher zu schlechterer oder verbesserter Performance führen. Damit kann ohne weitere Analyse nicht gezeigt werden, dass die Bedingungen (1) und (3) erfüllt sind.

Die Varianzen der schnellen deterministischen Anpassung beim Ringe-Problem sind zwar größer als im statischen Fall, die Ergebnisse der gleichen Varianten zeigen aber bei den beiden anderen Testproblemen

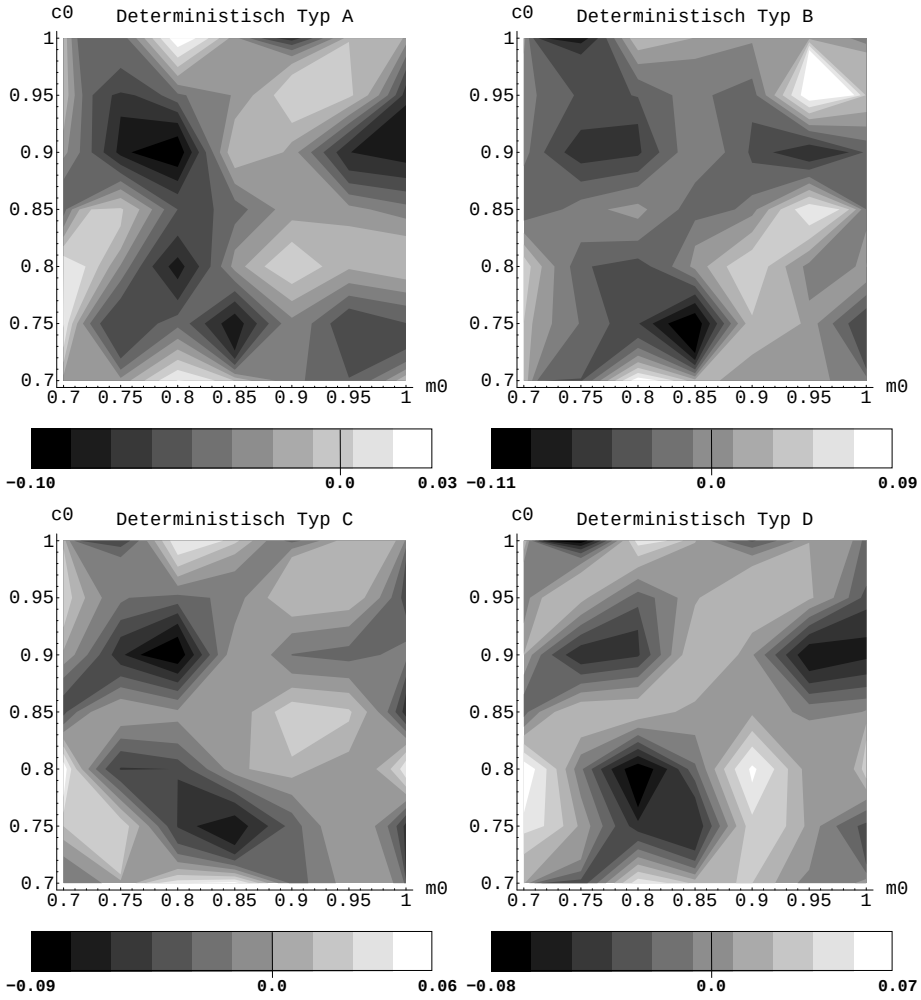


Abbildung 7.8: **Ringe:** Die durchschnittliche Fitness von 200 Experimenten mit deterministischer Anpassung Typ A-D im Verhältnis zur statischen Variante. $a_m = a_c = 0.93$

keine so deutliche Abweichung mehr. Die beim Hexapod-Problem auftretenden kleineren Varianzen sowohl der adaptiven als auch der deterministischen Varianten sind bei den anderen Testproblemen nicht erkennbar (siehe Tabelle 7.3). Dieses uneinheitliche Bild führt dazu, dass auch die Bedingungen (2) und (4) nicht als eindeutig erfüllt gelten können.

Um eine eindeutige Bewertung der Experimente vornehmen zu können ist es somit notwendig, statistische Tests durchzuführen, die aufzeigen, ob eindeutig signifikante Differenzen zwischen der statischen und einer der nicht-statischen Variante bestehen und zwar sowohl positiv eindeutige (signifikant bessere Fitness) als auch negativ eindeutige (signifikant schlechtere Fitness). In den Abbildungen 7.7 und 7.8 sowie A.1- A.4 sind gleichmäßig helle und dunkle Bereiche (also Gebiete mit besserer bzw schlechterer Performance) zu erkennen. Es ist auch in dieser Darstellung schwierig, Kombinationen von Startwerten für Mutations- und Rekombinationswahrscheinlichkeiten zu identifizieren, die in allen drei Problemen gleich gute (oder gleich schlechte) Ergebnisse liefern. Aus den anzutreffenden Einzelfällen mit deutlich besserer bzw. schlechterer Fitness lassen sich allerdings keine generellen Schlüsse über zu favorisierende Parameterkombinationen ziehen.

		Sinus $U_\alpha = 42.45$		Ringe $U_\alpha = 225.18$		Hexapod $U_\alpha = 218.35$
Variante		$a_m = a_c = 0.93$	$a_m = a_c = 0.97$	$a_m = a_c = 0.93$	$a_m = a_c = 0.97$	$a_m = a_c = 0.93$
det.	A	*56.0*	8.0	*675.5*	*237.5*	*266.5*
	B	25.0	41.0	163.5	3.5	**1099.5**
	C	31.0	9.0	177.5	81.5	*1199.5*
	D	**45.0**	10.0	113.5	5.5	*1011.5*
ad.	A	*53.0*	8.0	17.5	19.5	1.5
	B	22.0	30.0	**483.5**	**484.5**	194.5
	C	*54.0*	20.0	183.5	183.5	**1200.5**
	D	29.0	*58.0*	*739.5*	*739.5*	*445.5*

Tabelle 7.4: U-Werte der Wilcoxon-Tests. Alle Tests wurden mit $\alpha = 0.95$ durchgeführt. Ein fettgedruckter Eintrag bedeutet, dass die entsprechende Variante eine bessere Fitness als die statische Variante aufweist, und der Wilcoxon-Test negativ ausfällt ($U > U_\alpha$). Kursiv gesetzte Einträge bedeuten eine signifikant schlechtere Fitness als die statische Variante ($U < U_\alpha$). In allen anderen Fällen fällt der Wilcoxon-Test positiv aus.

Statistischer Vergleich der Strategien

Die Tatsache, dass die zu Grunde liegenden Verteilungen der mittleren Fitnesswerte bei den einzelnen Varianten nicht bekannt und auch mit vertretbarem Zeitaufwand nicht zu bestimmen sind, erfordert es, einen parameterfreien statistischen Test anzuwenden. Mit einem parameterfreien Test wie zum Beispiel dem *Wilcoxon-U-Test* [38] (auch bekannt unter dem Namen *Mann-Whitney-Test*) kann die Hypothese überprüft werden, ob zwei Stichproben X, Y mit jeweils unbekannter Verteilung F_X, F_Y aus derselben Grundgesamtheit stammen oder nicht. Hierfür werden die Werte $x_1, \ldots, x_{n_1}$ und $y_1, \ldots y_{n_2}$ gemeinsam der Größe nach geordnet. Als Testgröße wird die Gesamtzahl U der Inversionen herangezogen, wobei ein Wertepaar (x_i, y_j) eine Inversion bildet, wenn $y_j < x_i$ ist. In diesem Fall also werden die Fitnesswerte der 200 Experimente der statischen und einer nicht-statischen Variante daraufhin geprüft, ob sie aus der selben Grundgesamtheit stammen. Ist dies nicht der Fall, d.h. der Wilcoxon-Test liefert ein negatives Ergebnis, kann davon ausgegangen werden, dass die nicht-statische Variante die Leistung des Algorithmus signifikant verändert.

Beispiel Für $n_1 = 4$ und $n_2 = 5$ ergebe sich folgende Reihenfolge $y_5 x_3 x_4 y_1 y_2 x_2 y_4 y_3 x_1$ nach dem Sortieren. Je eine Inversion bilden x_3 und x_4 mit y_5, x_2 bildet drei Inversionen mit y_2, y_1 und y_5 und x_1 bildet fünf Inversionen (mit allen y_j). Für die Gesamtzahl der Inversionen gilt also $U = 10$. Die Hypothese $F_X = F_Y$ ist wahr, wenn U vom Erwartungswert $E_U = \frac{n_1 n_2}{2}$ nicht allzu stark abweicht. Gilt

$$| U - E_U | = | U - \frac{n_1 n_2}{2} | > U_\alpha \tag{7.11}$$

mit

$$U_\alpha = z_\alpha \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}, \tag{7.12}$$

wobei z_α das $1 - \frac{\alpha}{2}$-Quartil der Standardnormalverteilung ist, dann wird die Hypothese $F_X = F_Y$ abgelehnt und es kann (mit Konfidenz α) angenommen werden, dass X und Y aus verschiedenen Verteilungen stammen. Das Konfidenzniveau der Entscheidung wird durch α festgelegt.

Die Resultate der Analyse der verschiedenen Varianten mit dem Wilcoxon-Test ist in Tabelle 7.4 abgebildet. Es wurde für jeden Vergleich ein Konfidenzniveau von $\alpha = 0.95$ angenommen. Wenn also ein Eintrag in Tabelle 7.4 den kritischen Wert U_α überschreitet, dann besitzen die verglichenen Varianten eine unterschiedliche Verteilung mit einem Konfidenzniveau von 95%. Ist zusätzlich die durchschnittliche Fitness

der Varianten vom Typ A-D in Tabelle 7.3 kleiner (d.h. sie weisen eine bessere Fitness auf), dann ist der entsprechende Eintrag in Tabelle 7.4 **fett**. Ist der Wilcoxon-Test negativ und die mittlere Fitness schlechter, ist der entsprechende Eintrag *kursiv*. Alle anderen Einträge zeigen laut Wilcoxon-Test keine signifikanten Unterschiede ($U < U_\alpha$, d.h. Wilcoxon-Test positiv) zwischen statischen und nicht-statischen Varianten.

Die in fast allen Fällen statistisch signifikant schlechtere Variante A mit zeitabhängig verringerter Mutations- und Crossoverwahrscheinlichkeit im Verlaufe der Evolution und die Variante D mit ansteigender Mutations- und Crossoverwahrscheinlichkeit basierend auf Erfolgsraten zeigen, dass diese Methoden der Parameteranpassung nicht zur Verbesserung der Performance des Algorithmus beitragen, im Gegenteil, die mittlere Fitness ist schlechter (Eintrag kursiv in Tabelle 7.4). Die meisten anderen Varianten zeigen in manchen Fällen Verbesserungen, in anderen Fällen Verschlechterungen, in der Hauptzahl aller Fälle jedoch neutralen Einfluss auf die mittlere Fitness. Herauszustellen sind hier nur die deterministischen und adaptiven Varianten vom Typ B (Anstieg der Mutationwahrscheinlichkeit bei gleichzeitiger Verringerung der Crossoverwahrscheinlichkeit). Sie haben im schlechtesten Fall nur neutrale Auswirkungen, wohingegen in anderen Fällen eine Verbesserung zu beobachten ist. Eine dieser Varianten als Parameteranpassungsmethode zu wählen bedeutet also im schlechtesten Fall keine Verbesserung, im optimalen Fall aber eine signifikante Verbesserung der Performance des Grundalgorithmus. Alle anderen Varianten bergen die Gefahr einer schlechteren Performance, sind also keine wirkliche Alternative.

Die Performance des Algorithmus mit statischen Mutations- und Crossoverwahrscheinlichkeiten hängt jedoch stark von den gewählten Werten ab, wobei die richtige Einstellung der Parameter vom Experimentator allerdings „*ad hoc*kery and black magic“ erfordert [48]:

> „Evolutionary Computation (EC) has many parameters to tune. These are usually fixed by hand, more by art than by science. A poor choice of any of these can result in unsatisfactory performance“

Es besteht in diesem Falle also die Gefahr, ungünstige Parameterwerte zu wählen und so schlechtere Ergebnisse zu erzielen. Mit einer nicht-statischen Parameterstrategie wird dieses Risiko reduziert, denn ungünstige Parametereinstellungen werden hierbei vermieden und die zu erwartende Performance des Algorithmus ist vergleichbar. Es ist also vorteilhaft, durch viele selbst-regulierende Parameter den Benutzer von der eventuell falschen Wahl der Werte zu befreien. Allerdings bedeutet die Einführung jeder zusätzlichen selbst-regulierenden Variable, die es im Verlaufe des evolutionären Prozesses zu optimieren gilt, auch eine zusätzliche Vergrößerung des Suchraumes, was wiederum direkten Einfluss auf die benötigte Rechenzeit zur Lösung des Problemes hat.

Ein Vergleich der hier präsentierten Ergebnisse mit den erwähnten Arbeiten auf dem Gebiet der GA zeigt, dass auch dort ein ausschließlich positiver Effekt der selbst-regulierenden Parametersteuerung nicht nachgewiesen werden konnte. Die Performance der angewendeten Strategien scheint stark problemabhängig zu sein, ebenso wie in den hier gezeigten Experimenten. Bei allen drei Testproblemen lassen sich Parameterkombinationen von c_0- und m_0-Werten identifizieren, die zu guten Resultaten führen (beim Ringe-Problem ist die Kombination $c_0 = 0.7$ und $m_0 = 0.8$ beispielsweise ca. 7% besser die statische Variante), wobei irritierenderweise diese Regionen unabhängig von der gewählten Variante der Parameteradaptation sind. Zusätzlich gilt, dass es keine Kombinationen gibt, die bei allen drei Problemstellungen gute Lösungen erzielen.

In [48] gibt P. Darwen eine mögliche Lösung dieses unbefriedigenden Problems:

> „[...] those efforts have focussed on one or another attribute, never all of them simultaneously. [...] such piecewise attempts are doomed to failure, because customizable features are so interdependent. To self-adapt one particular attribute (or even a few) while guessing the others still lets unlucky guesses wreak havoc.“

Dort wird aber auch das schon erwähnte Problem der mit jeder zusätzlichen Variable ansteigenden benötigten Rechenzeit als Haupthindernis für die Verwirklichung eines ganzheitlichen selbst-regulierenden Evolutionären Algorithmus angegeben.

Es ist bemerkenswert, dass keine der Varianten vom Typ A-D in allen Experimenten Überlegenheit zeigt. Die Tatsache, dass intuitiv vernünftige Strategien in fast allen Experimenten die gleiche Performance wie intuitiv schlechte Strategien zeigen (so zum Beispiel die Varianten vom Typ B und C in Abb. 7.7) kann als ein Hinweis dafür gesehen werden, dass Parameteranpassungen während der Evolution eine Leistungsverbesserung des Algorithmus nicht garantieren und sogar Verschlechterungen auftreten können. Aus diesem Grund werden die hier untersuchten Strategien zur Anpassung der Operatorwahrscheinlichkeiten in den folgenden Hauptexperimenten nicht weiter verfolgt werden.

7.4 Verwendung eines Modelles der Fitnessfunktion

Die in den vorherigen Abschnitten beschriebenen Methoden zur Reduzierung der Gesamtlaufzeit haben zu unterschiedlichen Ergebnissen geführt. Die größten Einsparungen werden durch die Einführung des vorzeitigen Abbruchs erreicht (Abschnitt 7.2), wohingegen die Parametervariation nicht die erhofften Resultate gezeigt hat (Abschnitt 7.3).

Die weiterhin unvermeidbare hohe Anzahl von Fitnessauswertungen bei EA ist allerdings oft verbunden mit zusätzlichen hohen Kosten und/oder hohem Zeitaufwand. Gerade deshalb ist eine rechenzeit-effektive Approximation der originalen Fitnessfunktion wünschenswert, d.h. die Bestimmung der Fitness eines Individuums mit einer Methode mit reduzierter Laufzeit. Dies ist besonders in den folgenden Fällen interessant: (i) wenn die Auswertung der Fitnessfunktion rechenzeitintensiv ist, (ii) wenn keine mathematische Fitnessfunktion definiert werden kann, oder (iii) wenn zusätzliche mechanische Apparate verwendet werden müssen.

Es existiert bereits eine Reihe von Ansätzen, die Anzahl der Fitnessauswertungen in EA zu reduzieren. Wenn die Fitnessfunktion mathematisch definiert ist, wird eine Näherung der Funktion durch Interpolation zwischen Stützstellen im Suchraum (bereits ausgewertete Individuen) definiert (siehe z.B. [57, 93]). In komplexeren Fällen wird die Fitness eines Individuums durch Rekombination der Fitnesswerte der Eltern definiert (z.B. in [147]). Ein weiterer Ansatz versucht die Anzahl der Auswertungen zu minimieren, indem Individuen in Untermengen des Suchraumes durch spezielle Individuen (sog. *Prototypen*) repräsentiert werden [100]. Nur für diese Individuen wird eine Auswertung durchgeführt. Einen umfassenden Überblick über Arbeiten zu diesem Thema gibt z.B. [92].

Im Bereich GP sind bislang keine Anstrengungen zur Modellierung von Fitnessfunktionen unternommen worden. Es existieren jedoch Arbeiten, die versuchen, mit anderen Methoden die Anzahl der Auswertungen in GP zu reduzieren. Mit Hilfe der Informationstheorie und statistischen Methoden wird beispielsweise in [68] die Anzahl der Auswertungen begrenzt.

In den folgenden Abschnitten wird gezeigt, dass es möglich ist, eine laufzeitintensive Fitnessfunktion durch ein Modell zu ersetzen. Modelliert die Fitnessfunktion an sich Gegebenheiten eines weiteren Systems (wie beispielsweise die Bestimmung der Geschwindigkeit eines Laufroboters in der Simulation, also einem Modell der Wirklichkeit), so wird auch von einem Meta-Modell der Fitnessfunktion gesprochen.

7.4.1 Turnierselektion mit Klassifizierern

Bei einem Turnier während der Evolution wird eine Menge T von Individuen (normalerweise nehmen vier Individuen teil, es existieren aber auch Varianten mit nur zwei oder auch mit mehr als vier Teilnehmern) aus der Gesamtpopulation selektiert und anschließend ausgewertet. Die Fitnesswerte der Individuen werden paarweise miteinander verglichen und die Menge T der Individuen des Turnieres wird in die gleich großen

Mengen T_L (die Individuen mit schlechterer Fitness des direkten Vergleiches) und T_W (bessere Fitness) mit $T_L \cup T_W = T$ aufgeteilt. Die Individuen in T_L werden im anschließenden Variationschritt durch mit dem Crossoveroperator aus Individuen aus T_W erzeugten und anschließend mutierten Individuen ersetzt. Diese neu entstandenen Individuen werden in die Population eingefügt.

Ob ein Individuum ersetzt wird oder unverändert in der Population verbleibt, kann rein formal als das Resultat einer Klassifikation C des Vergleichs $V = (i, j)$ zweier Individuen i, j mit $i, j \in T$ betrachtet werden:

$$C : (i, j) \mapsto \{c_0, c_1\}. \tag{7.13}$$

Die Vergleiche werden hierdurch in zwei Klassen c_0 und c_1 aufgeteilt. Anschließend kann durch eine einfache Zuordnung aus den Klassen c_0 und c_1 die Menge T_W bzw. T_L bestimmt werden:

$$\begin{aligned} C(i, j) \in c_0 &\quad \Leftrightarrow \quad i \in T_W, j \in T_L \\ C(i, j) \in c_1 &\quad \Leftrightarrow \quad i \in T_L, j \in T_W. \end{aligned} \tag{7.14}$$

Die Klasse c_0 enthält die Vergleiche, in denen i die bessere Fitness besitzt, Klasse c_1 enthält die Vergleiche, in denen j die bessere Fitness besitzt:

$$\begin{aligned} C(i, j) \in c_0 &\quad \Leftrightarrow \quad fitness(i) > fitness(j) \\ C(i, j) \in c_1 &\quad \Leftrightarrow \quad fitness(j) >= fitness(i). \end{aligned} \tag{7.15}$$

Wichtig hierbei ist, dass die Reihenfolge der Individuuen über das Ergebnis der Klassifikation entscheidet, denn es gilt

$$C(i, j) \in c_0 \Leftrightarrow C(j, i) \in c_1. \tag{7.16}$$

Die Fitness eines Individuums wird in der Regel durch eine Auswertung $A : i \longrightarrow I\!R$ bestimmt, d.h. einem Individuum wird ein reeller Wert zugeordnet. Dieser wird durch die Anwendung der Fitnessfunktion A des jeweiligen EA ermittelt.

In allen Standardimplementierungen, die diese Form der Selektion und Variation verwenden, wird für die Klassifikation C der Vergleich der Fitnesswerte (siehe Gleichung (7.15)) der Individuen herangezogen. Bei Experimenten, bei denen die Bestimmung der Fitnessfunktion zeitaufwändig ist, ist dieser Schritt der die Laufzeit bestimmende Faktor des Gesamtexperiments, da bei EA in der Regel viele hundert Auswertungen nötig sind. Bei sehr lang dauernden Auswertungen ist es wünschenswert, die Zahl der Auswertungen möglichst ohne Qualitätsverlust zu reduzieren, um so entweder schneller Ergebnisse zu bekommen oder mehrere Experimente in der gleichen Zeit durchführen zu können.

Ist für die Bestimmung der Fitness eines Individuums nicht nur eine Software, sondern zusätzlich auch Hardware notwendig, so kann durch Einsparungen bei der Zahl der tatsächlich durchzuführenden Auswertungen Arbeitszeit und Verschleiß der Hardware reduziert werden. Die Arbeitszeit der Experimentatoren und der Verschleiß bzw. die Reparatur defekter Teile der verwendeten Geräte und der damit verbundene Zeitverlust verursachen zum Teil erhebliche Kosten, die durch eine Reduzierung der Auswertungen vermieden werden können.

7.4.2 Ein Modell der Fitnessfunktion

Betrachtet man die Folge von Selektion und Variation in einem EA, so ist für einen reibungslosen Ablauf des Algorithmus eine beliebige gültige Klassifikation C' notwendig, die die Menge T in die Teilmengen T_L und T_W aufteilt[4]. Eine Klassifikation ist genau dann gültig, wenn die Menge T in gleich große Teilmengen T_L und T_W aufgeteilt wird. Kann nun ein Klassifizierer C' angegeben werden, der eine identische Klassifikation

[4] Bei einer rein zufälligen Klassifikation C' ist natürlich keinerlei Selektionsdruck vorhanden, da gleichwahrscheinlich bessere und schlechtere Individuen ersetzt werden. Die Population wird einer zufälligen Drift durch den Suchraum unterliegen.

wie der Klassifizierer C leistet, bei gleichzeitig reduzierter Laufzeit, so wird ein identischer Selektionsdruck aufgebaut und die benötigte Laufzeit reduziert. Der Klassifizierer C' stellt also ein Modell der bisherigen Fitnessfunktion dar, die die Grundlage für die Selektion ist.

Die entscheidende Frage an dieser Stelle ist die nach der Existenz von C'. Einfachstes Beispiel für C' ist eine laufzeitoptimierte Version von C, d.h. eine schnellere Ermittlung der Fitness der Individuen. Hierdurch zu erzielende Einsparungen sind allerdings gering, da immer noch die gleichen Funktionen und Abläufe für die Berechnung der Fitnesswerte ausgeführt werden müssen.

Kann nun die Klassifikation C der Individuen, für die bislang die durch die Auswertung A (zum Beispiel durch eine Dynamiksimulation eines Roboters) bestimmte Fitness die Grundlage war, aufgrund *anderer Kriterien* erfolgen, so wird der laufzeitbestimmende Schritt eingespart, d.h. die Bestimmung des exakten Fitnesswertes für jedes Individuum für jeden Vergleich.

Dies bedeutet, dass eine Abbildung C' gefunden werden muss, die die Klassifikation C modelliert:

$$C' : (i, j) \mapsto C(i, j), \tag{7.17}$$

mit anderen Worten eine Funktion, die das Ergebnis der Klassifikation C allein aus $V = (i, j)$ berechnet. C' verwendet dann für die Klassifikation ausschließlich genotypische Informationen (die jeweiligen Genome) aus i und j, die Klassifikation C hingegen basiert auf phänotypischen Informationen.

Da nicht davon ausgegangen werden kann, dass zu jeder Klassifikation C eine Modellfunktion C' mit identischer Klassifikationsleistung angegeben werden kann, muss die Güte einer Ersatzklassifikation definiert werden. Die Qualität einer Klassifikation C' kann sehr einfach als die Summe der Kosten für falsch klassifizierte Vergleiche einer Menge V berechnet werden:

$$quality(C') = \sum_{V} g(C(i, j), C'(i, j)), \tag{7.18}$$

wobei die Kosten einer Fehlklassifikation mit einer Funktion $g(C(i, j), C'(i, j))$ berechnet werden:

$$g(C(i, j), C'(i, j)) = \begin{cases} 0, & \text{wenn } C(i, j) = C'(i, j), \\ 1, & \text{sonst.} \end{cases} \tag{7.19}$$

$quality(C')$ ist damit gleichzeitig die Definition einer Fitnessfunktion für einen Algorithmus, der ein geeignetes Modell C' für C lernen soll. Es handelt sich hierbei um ein klassisches Beispiel für die Anwendung von überwachtem Lernen, da der Ausgang der zu lernenden Klassifikation C' zur Bestimmung der Qualität (Fitness) mit dem bekannten Ergebnis von C verglichen wird.

Dass es möglich ist, mit den Ergebnissen von C einen neuen Klassifizierer C' zu lernen, auch und vor allem bei komplexen Auswertungen der Individuen, wird in Abschnitt 10.3 gezeigt, bei dem mit Hilfe von Genetischer Programmierung ein Klassifizierer evolviert wird.

7.4.3 Das Konfidenzniveau

Wenn ein Klassifizierer das Ergebnis eines Evolutionsprozesses ist (siehe Abschnitt 10.3), so ist die Qualität der besten gefundenen Lösung nicht immer gleich der optimalen Lösung. Dies bedeutet im Falle des Klassifizierers, dass in bestimmten Fällen falsch klassifiziert wird. Die Anzahl der korrekten Klassifizierungen in Relation zur Gesamtzahl der Klassifikationen ergibt die *hit rate* des Klassifizierers. Eine weitere Größe, das Konfidenzniveau $k_V \in [0, 1]$ der Klassifikation, ist zusätzlich notwendig. Das Konfidenzniveau beschreibt die „Sicherheit“ der Klassifikation. Eine Klassifikation mit hohem Konfidenzniveau deutet eine zuverlässigeres Ergebnis an als eine Klassifikation mit niedrigerem Konfidenzniveau. Eine gewisse Fehlerwahrscheinlichkeit p_e für eine Fehlklassifikation bleibt allerdings dennoch. Die Fehlerwahrscheinlichkeit p_e

ist hierbei unabhängig von k_V. Die Ausgabe des erweiterten Klassifizierers C' ist also nach Gleichung (7.20) eine Klasse und das Konfidenzniveau k_V der Klassifikation

$$C' : V \mapsto < \{c_0, c_1\}, k_V > . \tag{7.20}$$

Hiermit wird für jede Klassifikation ein Konfidenzniveau angegeben, das es erlaubt, das Resultat der Klassifikation entweder zu akzeptieren oder abzulehnen. Wird das Ergebnis der Klassifikation akzeptiert, so kann die zeitaufwändige Bewertung der Individuen i, j entfallen. Eine Klassifikation kann beispielsweise dann akzeptiert werden, wenn k_V ein gegebenes Mindestkonfidenzniveau k überschreitet.

7.4.4 Einfluss auf die Performance

Es ist schwierig, allein aus der Anzahl oder dem Prozentsatz der Fehlklassifikationen, die darin resultieren, dass Individuen mit geringer Qualität gemäß der angegebenen Fitnessfunktion bessere Individuen ersetzen, einen ausschließlich negativen Einfluss auf die Performance des Gesamtalgorithmus zu berechnen. Obwohl als erste Konsequenz die durchschnittliche Fitness der Population durch Fehlklassifikationen sinkt, kann das genetische Material durch die Möglichkeit der Rekombination mit anderen Individuen oder durch Mutation durchaus positiven Einfluss auf die Fitnessentwicklung kommender Generationen haben.

EA basieren auf *beam search* und untersuchen Regionen des Suchraumes, die durch die Zusammensetzung der Population vorgegeben werden. Die beste Route zum globalen Optimum ist im vorhinein nicht bekannt und hängt von vielerlei Faktoren wie z.B. der Zusammensetzung der Startpopulation, der Anwendungsreihenfolge der Variationsoperatoren oder auch von Effekten der Pseudozufallszahlengeneratoren ab.

Der Einfluss des Konfidenzniveaus k, der Fehlerwahrscheinlichkeit p_e und der Laufzeitdifferenz zwischen Auswertung und Klassifikation wird im Folgenden anhand eines Beispieles verdeutlicht. Grundlage der Untersuchung ist ein GA (Parameter des GA sind in Tabelle A.3 aufgeführt) mit zwei verschiedenen Fitnessfunktionen. Als zu maximierende Fitnessfunktionen sind die Funktionen (7.21) und (7.22) definiert. Ein Individuum X des GA, mit $X = (x_0, \ldots, x_9)$ und $x_i \in \{0,1\}$, $i = 0, \ldots, 9$, wird durch eine 10-Bit-Binärzahl repräsentiert.

$$f_1(X) = \sum_{i=0}^{9} x_i, \tag{7.21}$$

$$f_2(X) = \begin{cases} 100 \cdot x_9 & \text{,wenn} \quad x_5 + x_6 \geq 1 \\ (x_0 + 10x_3)^2 - (x_1 + x_2 + x_4)^3 & \text{,sonst.} \end{cases} \tag{7.22}$$

Die Fitnesswerte werden für jedes Individuum berechnet. Nach einer Startphase, in der keine Klassifizierung stattfindet, ersetzen in den anschließenden Turnieren die Individuen mit der besseren Fitness die schlechteren Individuen mit der Wahrscheinlichkeit p_r (nach Gleichung (7.23)). Mit der Wahrscheinlichkeit p_p werden die besseren Individuen durch die schlechteren Individuen ersetzt (nach Gleichung (7.24)). Hiermit wird das Resultat einer Klassifikation C' mit einer Fehlerwahrscheinlichkeit p_e simuliert.

$$p_r = p_e(1-k) \tag{7.23}$$

$$p_p = (1-p_e)(1-k). \tag{7.24}$$

Es wird weiterhin vereinfachend angenommen, dass die Konfidenzniveau-Werte k_V von C' in [0,1] gleichverteilt sind. Dadurch ergibt sich für die Fehlerwahrscheinlichkeit $p_e = k_V$, d.h. p_e ist gleich der Wahrscheinlichkeit, dass k_V kleiner ist als k.

Um die Gesamtlaufzeit des Experimentes zu bestimmen, wird die Anzahl der Klassifikationen (n_K) und Auswertungen (n_A) während der Evolution summiert. Während der ersten Phase finden n_S Auswertungen statt und die entsprechende Laufzeit $(n_S \cdot t_A)$ wird zur Gesamtlaufzeit addiert. In der zweiten Phase beginnt die Klassifikation C' und die Laufzeit t_K wird für jeden Vergleich addiert. Ist hierbei $k_V < k$, d.h. die

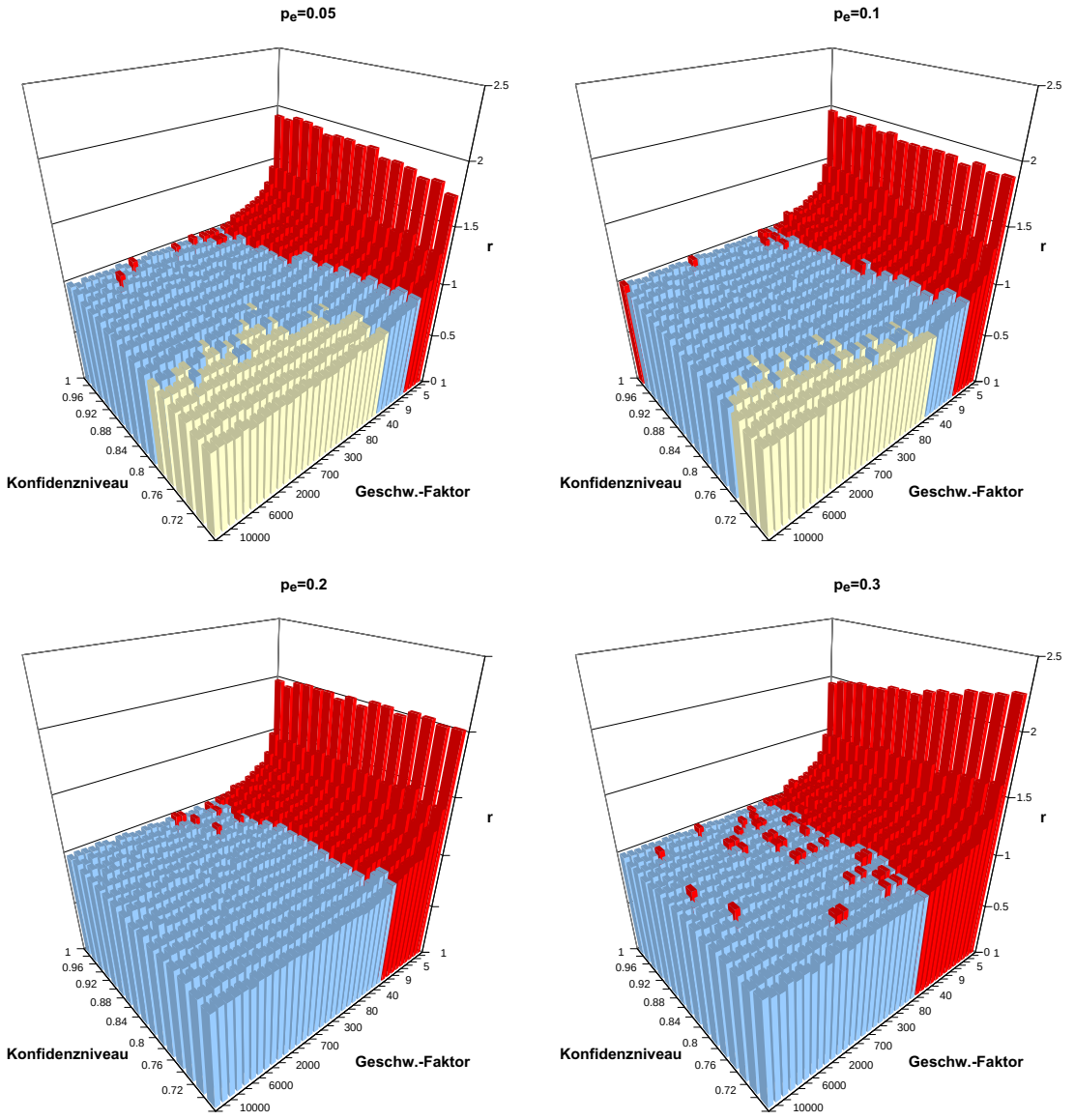


Abbildung 7.9: Vergleich des Einflusses von Geschwindigkeitsverhältnis und Konfidenzniveau auf die Gesamtlaufzeit. Dargestellt ist das Laufzeitverhältnis r (nach Gleichung 7.25) zwischen dem Standardalgorithmus und der Variante mit Klassifikation in Abhängigkeit vom Konfidenzniveau des Klassifizierers und dem Geschwindigkeitsunterschied zwischen Auswertung und Klassifikation. Vergleich nach Gleichung (7.21).

Klassifikation besitzt ein zu geringes Konfidenzniveau, wird die Fitness für jedes Individuum durch eine Auswertung bestimmt und die Laufzeit t_A dieser zwei Auswertungen hinzugezählt. Die Zeit für das Lernen von C' wird t_L genannt und als konstant angenommen. Verglichen wird die durchschnittliche Laufzeit von 100 Läufen die jeweils benötigt wird, bis die durchschnittliche Fitness der gesamten Population 80% der maximal möglichen (und a priori bekannten) Fitness erreicht. Dieser Wert wird dividiert durch die Standardlaufzeit (nach Gleichung (7.25)). Die Standardlaufzeit ist definiert als die in 100 Läufen durchschnittlich benötigte Zeit des unmodifizierten Algorithmus bis zum Erreichen des Fitnessniveaus. Das Laufzeitverhältnis r ist also

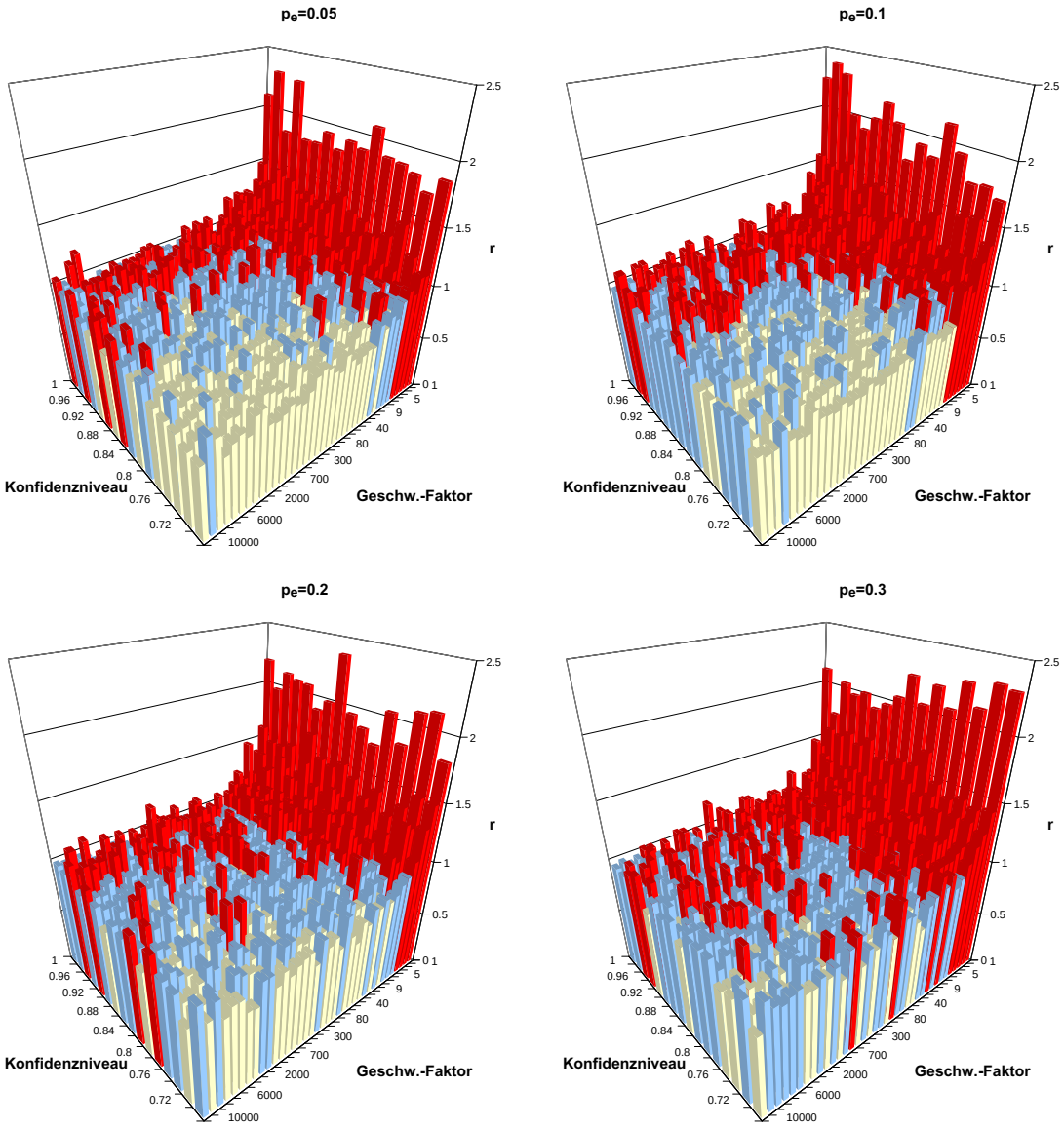


Abbildung 7.10: Vergleich des Einflusses von Geschwindigkeitsverhältnis und Konfidenzniveau auf die Gesamtlaufzeit. Dargestellt ist das Laufzeitverhältnis r (nach Gleichung (7.25)). Vergleich nach Gleichung (7.22).

$$r = \frac{1}{100} \sum_{i=0}^{100} \frac{(n_S + n_K^i) \cdot t_A + n_K^i \cdot t_K + t_L}{N \cdot t_A}, \tag{7.25}$$

wobei n_K^i, n_A^i die Anzahl der Klassifikationen bzw. Auswertungen im Lauf i beschreiben. Die Parameter für die Berechnung der tatsächlichen Laufzeiten sind in Tabelle A.4, die Parameter des GA in A.3 aufgeführt.

In Abbildung 7.9 sind die Vergleichsdaten dargestellt. Deutlich zu erkennen ist einerseits, dass sich mit kleinerem Konfidenzniveau k die Laufzeit des Algorithmus mit Klassifikation verringert, andererseits, dass mit sinkendem Geschwindigkeitsverhältnis von Auswertung und Klassifikation die Laufzeit des Algorithmus steigt. Die hell dargestellten Bereiche stellen Kombinationen von Geschwindigkeitsverhältnis und Konfi-

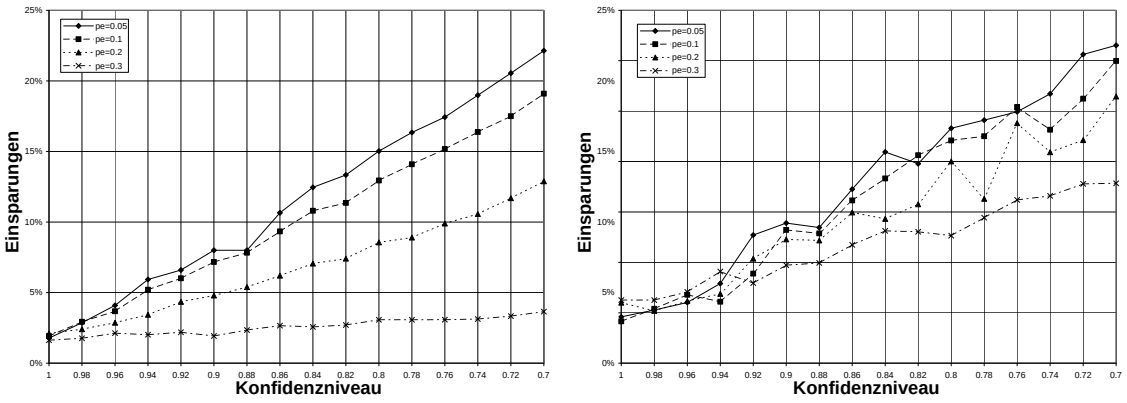


Abbildung 7.11: Durchschnittliche Einsparung tatsächlicher Auswertungen aller durchgeführten Experimente in %. **Rechts:** Einsparung an Auswertungen bei Fitnessfunktion (7.21), **Links:** Einsparung an Auswertungen bei Fitnessfunktion (7.22).

denzniveau dar, bei denen der Algorithmus mit Klassifikation absolut *weniger* Zeit bis zum Erreichen eines definierten Fitnessniveaus benötigt ($r < 1$). Unvorteilhafte Kombinationen führen zu vergleichsweise längeren Laufzeiten und sind dunkel dargestellt. Beträgt die Laufzeit des modifizierten Algorithmus weniger als 85% der Vergleichslaufzeit ($r < 0.85$), sind die entsprechenden Datenpunkte sehr hell dargestellt. In diesem Beispiel ist deutlich zu sehen, dass ab eines bestimmten Geschwindigkeitsverhältnisses zwischen Klassifikation und Auswertung keine Verbesserung der Laufzeit mehr erreicht wird, bzw. der Geschwindigkeitszuwachs nicht in der Lage ist, niedrigere Konfidenzniveaus der Klassifikation zu kompensieren. Umgekehrt ist bei Unterschreiten eines gewissen Geschwindigkeitsverhältnisses auch durch Erhöhen des Konfidenzniveaus der Klassifikation keine Verringerung der Laufzeit möglich. Steigt die Fehlerwahrscheinlichkeit p_e für eine Fehlklassifikation trotz Überschreiten des Konfidenzniveaus k an, so ist zu erkennen, dass die Laufzeitersparnis abnimmt. Viele Fehlklassifikationen führen also zu einer verlängerten Laufzeit. Es ist allerdings erstaunlich, dass eine Fehlerrate von 30% bei der Klassifikation immer noch Laufzeitverhätnisse $r \leq 1$ für viele Parameterkombinationen erlaubt.

Ist die Fitnessfunktion nicht so simpel wie in Gleichung (7.21), verändert sich das Bild. In Abbildung 7.10 ist der Verlauf der Laufzeiten mit der Fitnessfunktion (7.22) dargestellt. Deutlich erkennbar ist ein inhomogeneres Verhalten bei unterschiedlichen Parameterkombinationen. Die Tendenz zu geringeren Laufzeiten bei hohen Laufzeitdifferenzen zwischen Auswertung und Klassifikation und gleichzeitigem niedrigen Konfidenzniveau bleibt allerdings sichtbar. Auch hier ist die Laufzeitersparnis geringer mit steigender Fehlerwahrscheinlichkeit.

Betrachtet man anstelle der absoluten Laufzeiteinsparung die Anzahl der tatsächlich eingesparten Auswertungen, so ergibt sich ein etwas anderes Bild. Die Entwicklung der Anzahl der Auswertungen in Abhängigkeit von Konfidenzniveau und Fehlerwahrscheinlichkeit ist in Abbildung 7.11 dargestellt. Die Reduzierung der Anzahl der Auswertungen ist unabhängig von der Laufzeitdifferenz zwischen Klassifikation und Auswertung. Die dargestellten Kurven zeigen die durchschnittlich erreichte Einsparung bei 3700 Experimenten je Datenpunkt. Deutlich erkennbar ist ein Anstieg der eingesparten Auswertungen bei sinkendem Konfidenzniveau, eine direkte Folge aus der Tatsache, dass nur ein kleiner Prozentsatz der Klassifikationen mit sehr hohem Konfidenzniveau erfolgt, d.h. in der Mehrzahl aller Fälle ausgewertet werden muss. Mit steigender Fehlerwahrscheinlichkeit sinkt die prozentuale Ersparnis allerdings wieder. Diese Eigenschaft ist unabhängig von der gewählten Fitnessfunktion, bei Gleichung (7.21) ist sie allerdings stärker ausgeprägt als bei Gleichung (7.22).

Interessant ist, dass bei komplexerer Fitnessfunktion selbst hohe Fehlerwahrscheinlichkeiten eine Ersparnis an Auswertungen von bis zu 10% erlauben, und dass die prozentuale Ersparnis an Auswertungen bei hohen Fehlerwahrscheinlichkeiten höher ist als bei der simplen Fitnessfunktion. Eine Ursache dieses Phänomens kann an dieser Stelle wegen der begrenzten Aussagekraft der gewählten Beispiele nicht ausreichend begründet werden. In weiterführenden Arbeiten müssen daher Aspekte der Meta-Modellierung eingehend betrachtet werden. Es kann aber aufgrund der Eigenschaften der Beispiele vermutet werden, dass mit Hilfe der Meta-Modellierung sowohl die Gesamtlaufzeit als auch die Anzahl der durchzuführenden Auswertungen in einem evolutionären Experiment reduziert werden kann.

Anhand der Experimente mit realen vierbeinigen Laufrobotern in Kapitel 10 wird gezeigt, dass (i) auch für komplexe Genotyp-Phänotyp-Relationen ein Klassifizierer C' evolviert werden kann, der in der Lage ist, den Ausgang von Vergleichen auf Basis der Fitnesswerte von Individuen mit geringer Fehlerrate vorherzusagen, und (ii) dass mit der Methode der Meta-Evolution die Anzahl der für eine erfolgreiche Evolution notwendigen Auswertungen drastisch reduziert werden kann.

Kapitel 8

Experimente mit dem Simulationssystem SIGEL

Hauptaugenmerk der folgenden Experimente lag auf der Evolution von Laufprogrammen für variable Roboterstrukturen. Zuerst werden die verwendeten Fitnessfunktionen definiert, mit denen die daran anschließenden Experimente durchgeführt werden. Die in den einzelnen Experimenten erreichten Generationszahlen sind unterschiedlich, was darauf zurückzuführen ist, dass bei einer konstanten Zeit für die Evolution aufgrund von unterschiedlichen Komplexitätsgraden der simulierten Roboter und variabler Last und Zusammensetzung des Rechenclusters die Zahl der durchgeführten Turniere pro Zeiteinheit unterschiedlich ist. Für einen einzelnen Evolutionslauf werden typischerweise mehrere Tage benötigt.

8.1 Die verwendeten Fitnessfunktionen

Die Fitnessfunktion ist der entscheidende Teil bei der Anwendung eines EA. Mit ihr wird definiert, was mit dem Algorithmus überhaupt optimiert werden soll. In diesem Fall soll das zu Grunde liegende GP-System Programme evolvieren, die in der Lage sind, einen Laufroboter zu bewegen; mit anderen Worten: der Roboter soll Laufen lernen. Die Schwierigkeit besteht nun darin, eine Fitnessfunktion mathematisch so zu definieren, dass es möglich ist, bessere und schlechtere Laufprogramme zu unterscheiden.

Es existieren mathematische Modelle des Laufens für verschiedene biomechanische Konfigurationen. Die unterschiedlichen Gangarten von Zweibeinern (Menschen), Vierbeinern (Nagetiere, Hunde, Katzen, Pferde, Elefanten) und auch von Sechs- und Mehrbeinern (Insekten) sind in aufwändigen biomechanischen Messungen analysiert worden [175, 176]. Mit extrem schnellen Kameras können beispielsweise Gehbewegungen aufgenommen, extrahiert und mit Hilfe von zuvor ermittelten Größen (Längen, Massen, Winkel) des untersuchten Lebewesens zu einem kinematischen Modell zusammengefasst werden. Diese Gleichungen beschreiben dann mathematisch exakt den Bewegungsablauf des Individuums. Hauptnachteil dieser mathematischen Beschreibung, die natürlich die optimale Fitnessfunktion darstellt, ist, dass sie auf sehr aufwändigen Messungen beruht und zusätzlich nur für eine bestimmte geometrische Konfiguration korrekt ist. Kleine Unterschiede in den Abmessungen führen dazu, dass die Bewegungsgleichungen eines Vierbeiners für einen anderen Vierbeiner falsch sind. Gerade diese Abhängigkeit der Laufmustergenerierung von individuellen Merkmalen eines Individuums sollte mit dem hier verfolgten Ansatz aber vermieden werden. Hinzu kommt, dass der Aufwand zur Erstellung dieser präzisen Beschreibung eines Ganges und der Aufwand zur Implementierung eines Algorithmus für diesen speziellen Gang nicht unterschätzt werden darf.

Eine Fitnessfunktion aufzustellen, die das GP-System in die Lage versetzt, im Verlaufe der Evolution Individuen mit stetig steigender Qualität zu generieren, wobei die Qualität sowohl Qualität im Sinne der

Fitnessfunktion als auch Qualität im Sinne eines objektiven Betrachters des Gangmusters bedeutet, ist das Ziel der folgenden Abschnitte.

8.2 SimpleFitness

Die Funktion *simple fitness* ist eine vereinfachte Version der in Abschnitt 7 definierten Fitnessfunktion für das Verfahren eines einzelnen Roboterbeines (Gleichung (7.3)). Es wird hier jedoch, statt während der gesamten Simulationszeit die Position des Roboters in regelmäßigen Abständen zu messen und das Quadrat der Abweichung von der Sollposition zu berechnen, nur die Start- und Endposition zur Berechnung der Fitness herangezogen. Die Länge der direkten Verbindung zwischen Startpunkt $x(t_0)$ und Endpunkt $x(t_{end})$ wird als zurückgelegte Strecke s betrachtet, ungeachtet des eventuell vom Roboter tatsächlich gelaufenen Weges. Die Strecke s wird durch die dafür benötigte Zeit dividiert, um so die Geschwindigkeit des Roboters zu erhalten. Es wird keine Sollposition vorgegeben

$$fitness = \frac{s}{t_{end} - t_0}, \text{ mit } s = \mid x(t_{end}) - x(t_0) \mid . \tag{8.1}$$

Wie in den folgenden Experimenten zu sehen sein wird, erzeugt eine Evolution mit dieser Fitnessfunktion Individuen, die in der Lage sind, die Roboter tatsächlich fortzubewegen. Die erreichten Bewegungsabläufe sind allerdings ungewöhnlich und können nicht als „Gehen“ im herkömmlichen Sinne bezeichnet werden. Dies liegt vor allem daran, dass die Roboter nicht, wie erwartet, nur mit den Enden der Extremitäten den Boden berühren, gleichzeitig durch die Bewegung dieser Glieder den Vortrieb erzeugen und durch die Bewegung des Rumpfes das Gleichgewicht halten, sondern mit dem ganzen Körper auf dem Boden aufliegen und durch rhythmische Bewegungen in eine Richtung „robben“.

Um diese Art der Bewegung zu vermeiden und eine eher der intuitiven Vorstellung des Gehens entsprechende Art der Fortbewegung zu erzeugen, wurde eine erweiterte Fitnessfunktion implementiert, die auf der Funktion *simple fitness* aufbaut.

8.3 NiceWalkingFitness

Die Funktion *nice walking fitness* stellt eine Erweiterung der Funktion *simple fitness* dar, wobei die eigentliche Fitnessbewertung nach Gleichung (8.1) unverändert bleibt. Für die Dauer der Simulation wird allerdings ein Korridor festgelegt, innerhalb dessen sich ein ausgezeichnetes Roboterglied (in der Regel das Torsoelement) bewegen darf. Verlässt das Element diesen Korridor, wird die Fitness auf Null gesetzt und die Simulation kann vorzeitig abgebrochen werden (siehe Abschnitt 7.2). Eine kriechende oder robbende Bewegung wie sie mit der *simple fitness* erzielt wurde, kann dadurch vermieden werden, dass der Korridor für das Torsoelement in einer Mindesthöhe h_{min} oberhalb des Bodens definiert wird. Hierdurch wird ein Selektionsdruck zu Gunsten von Individuuen erzeugt, die eine aufrechte Bewegung des Roboters erzeugen. Die Funktion *nice walking fitness* ist formal definiert als

$$fitness = \begin{cases} \frac{(x(t_{end}) - x(t_0))}{t_{end} - t_0}, & \text{wenn gilt: } h_{min} \leq h(t) \leq h_{max} \text{ mit } t_0 \leq t < t_{end} \\ 0 & \text{sonst.} \end{cases} \tag{8.2}$$

Die evolvierten Bewegungsmuster unterscheiden sich erheblich von den mit der Funktion *simple fitness* erzeugten Programmen. Die Programme bewegen die Roboter in einer Weise, dass das Torsoelement den definierten Korridor nicht verlässt, d.h. einen Mindestabstand zum Boden einhält. Auf diese Weise können die erzeugten Bewegungsmuster tatsächlich als „gangähnlich“ bezeichnet werden.

In den folgenden Abschnitten sind Experimente aufgeführt, die zum Teil mit beiden Fitnessfunktionen durchgeführt worden sind. Ausführungen zu den visuellen Unterschieden befinden sich in den entsprechenden Abschnitten.

Unter der URL `http://ls11-www.cs.uni-dortmund.de/people/ziegler/robotics.html` sind im Internet Videos der für die einzelnen Roboter evolvierten Bewegungsabläufe inklusive der Experiment-Dateien verfügbar.

8.3.1 Evolution von GP-Programmen für beliebige Roboterarchitekturen

In den folgenden Abschnitten werden die verschiedenen Roboter beschrieben, die modelliert wurden, um die generelle Fähigkeit des Systems zu testen, Laufrobotersteuerungsprogramme für unterschiedliche Roboterstrukturen zu evolvieren. Dabei wurde auf ein möglichst vielfältiges Spektrum an Roboterarchitekturen Wert gelegt. Die Roboter sind aufsteigend nach Komplexität aufgeführt, d.h. mit wachsender Anzahl von Extremitäten, wobei die einfachsten Roboter überhaupt keine Beine besitzen. Manche Roboter sind bereits existierenden Roboterarchitekturen nachempfunden (z.B. der sechsbeinige Roboter in Abbildung 8.8), bei anderen Robotern gibt es keine real existierenden Vorbilder (siehe z.B. den Vierbeiner in Abbildung 8.7).

Ein minimaler Roboter

Abbildung 8.1 zeigt ein sehr einfaches Robotermodell. Dabei handelt es sich lediglich um einen Roboter mit zwei Gliedern, die mit einem Rotationsgelenk verbunden sind. In den zu diesem Experiment gehörigen Tabellen in Abschnitt B.1 sind die Parameter des Roboters, des GP-Systems, der Dynamiksimulation und der Umwelt aufgeführt.

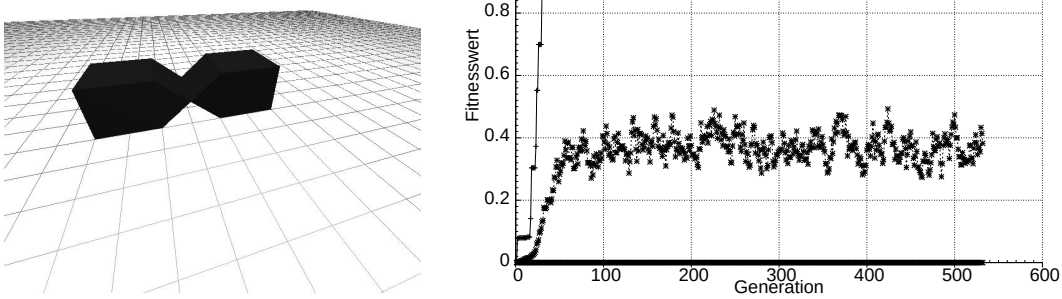


Abbildung 8.1: **Links:** Ein minimaler Roboter mit einem Freiheitsgrad. **Rechts:** Fitnessverlauf des Experiments mit dem minimalen Roboter und der Funktion *simple fitness*. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an.

Die Fortbewegung des Roboters wird durch eine zyklische Auf- und Abbewegung des Gelenkes erreicht, wodurch der zweigliedrige Roboter abwechselnd mit dem Vorder- und Hinterglied den Boden berührt. Durch ein schnelles Oszillieren des Gelenkes zu Beginn der Bewegung wird eine anschließende, gleichmäßige, hüpfende Bewegung ermöglicht. Der Roboter erreicht eine Geschwindigkeit von über $1.1\frac{m}{s}$, wenn das beste Individuum ausgeführt wird (siehe Abb. 8.1).

Ein Roboter mit linearer Struktur

Der Roboter in Abbildung 8.2 besteht aus einem würfelförmigen Torsoglied, drei Zwischengliedern und einem abgerundeten Endglied. Jedes Zwischenglied besteht aus zwei um 90 Grad gegeneinander versetzten

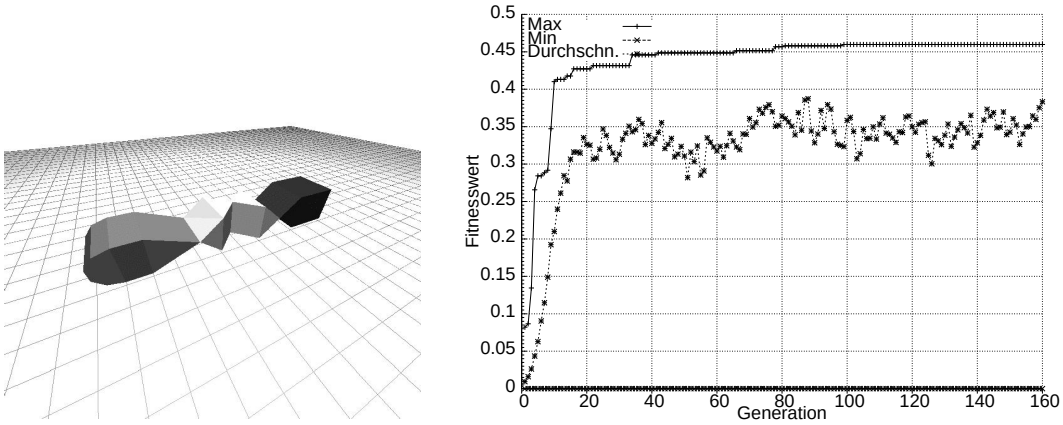


Abbildung 8.2: **Links:** Ein linearer Roboter mit drei DOF. **Rechts:** Fitnessverlauf des Experiments mit dem linearen Roboter und der Funktion *simple fitness*. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an.

Prismen. Alle Gelenke sind Rotationsgelenke. In den Tabellen in Abschnitt B.2 sind weitere Parameter des Roboters und des Experimentes aufgeführt.

Die Fortbewegung des Roboters entspricht der intuitiven Vorstellung: Die mittleren Glieder werden angehoben, wodurch der hintere Abschnitt des Roboters nach vorne bewegt wird[1]. Anschließend senkt sich der mittlere Abschnitt wieder und das Vorderteil bewegt sich nach vorne. Diese Bewegung wird zyklisch wiederholt, der Roboter bewegt sich wie eine Raupe stetig vorwärts. Interessant ist, dass das seitwärts gerichtete Gelenk in der Mitte des Roboters nicht verwendet, sondern aktiv versteift wird, um die lineare Vorwärtsbewegung, die mit einem maximalen Fitnesswert bewertet wird, nicht zu behindern.

Ein zweibeiniger Roboter

Der Zweibeiner besteht aus zwei Beinen mit jeweils drei Gliedern. Die obersten Glieder der Beine sind im Gegensatz zu der Anordnung bei humanoiden Robotern nicht um die Transversalachse sondern um die Sagittalachse drehbar. Das „Kniegelenk“ der Beine ist um 90 Grad gedreht (es dreht sich also um eine parallel zur Sagittalebene laufende Achse), bei humanoiden Beinen sind hingegen alle Gelenke (Hüfte, Knie, Fuß) um parallel zur Sagittalebene laufende Achsen drehbar. In den Tabellen in Abschnitt B.3 sind weitere Parameter des Roboters und der Experimente aufgeführt.

Die Bewegung, die der Roboter ausführt, wenn das beste Individuum der Evolution mit der Funktion *simple fitness* ausgeführt wird, lässt den Roboter zu Beginn aus der Startkonfiguration (siehe Abbildung 8.3) umfallen. Daran anschließend verwendet der Roboter eine besondere Fortbewegungsart, die auch in den späteren Experimenten das dominierende Bewegungsmuster war. Durch ruckartige Bewegungen werden die Glieder des Roboters angehoben und nach vorne beschleunigt, gefolgt von kurzen rückwärts gewandten Gegenbewegungen. Durch die Impulserhaltung der beschleunigten Glieder „rutscht“ hierdurch der auf dem Boden liegende Teil des Roboters nach und der Mittelpunkt des Roboters bewegt sich nach vorne. Nach einer kurzen Zeit wird diese Bewegung wiederholt. Auf diese Weise bewegt sich der Roboter ruckartig rutschend vorwärts.

Wenn als Zielfunktion die Funktion *nice walking fitness* gewählt wird, unterscheidet sich die evolvierte Bewegung grundsätzlich von der Bewegung, die mit der Funktion *simple fitness* evolviert wurde. Da die Funktion *nice walking fitness* Laufprogramme mit einer sehr schlechten Fitness bewertet, die das Zentralglied

[1] Dies wird durch unterschiedliche Materialien der einzelnen Glieder begünstigt. Die Materialien weisen dadurch verschiedene Dichten und Reibkoeffizienten auf.

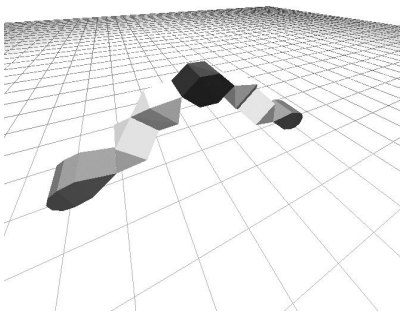


Abbildung 8.3: Ein zweibeiniger Roboter mit 6 DOF

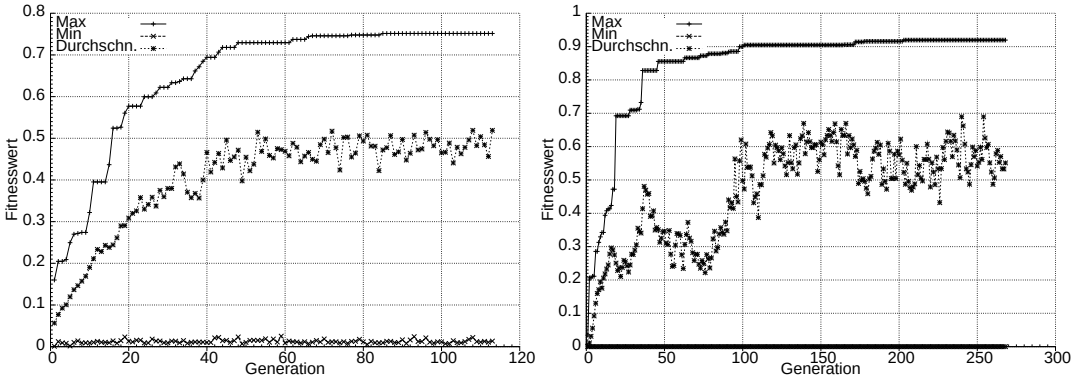


Abbildung 8.4: Fitnessverläufe von Experimenten mit dem zweibeinigen Roboter. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an. **Links:** Evolution mit der Funktion *simple fitness*. **Rechts:** Evolution mit der Funktion *nice walking fitness*.

des Roboters nicht in einer bestimmt Höhe halten, entstehen sehr schnell Programme, die tatsächlich eine aufrechte Fortbewegung ermöglichen. Durch die Startkonfiguration (der Roboter steht aufrecht, siehe Abbildung 8.3) muss zuerst eine Folge von Instruktionen evolviert werden, die ein Umkippen verhindert. Der Roboter beginnt dann mit einer Bewegung, die den gleichen Mechanismus verwendet, der auch schon bei der Bewegung mit der *simple fitness* beschrieben wurde: Ein Bein wird nach vorne beschleunigt und angehoben. Durch einen kurzen ruckartigen Impuls folgt der Rest des Roboters nach, wobei das zweite Bein eher rutschend nachgezogen wird. Anschließend versteift sich der Roboter wieder und das Vorderbein wird erneut angehoben und nach vorne beschleunigt. Die Kniegelenke der Beine, die parallel zur Sagittalebene (und damit senkrecht zur Bewegungsrichtung) stehen, werden bei dieser Bewegung aktiv versteift (d.h. durch entsprechende Motorkommandos in den Individuen), um ein Umfallen des Roboters auf die Seite zu verhindern. Wird mit der Funktion *simple fitness* evolviert, werden im Gegensatz dazu durchaus auch die Kniegelenke der Beine bewegt.

Bei diesen Experimenten ist ein weiteres, sehr interessantes Phänomen aufgetreten: Die Individuen wurden über eine Simulationszeit von 3 Minuten bewertet. In nachträglichen Simulationen zur Erstellung von Animationen wurde das beste Individuum länger simuliert. Interessanterweise begann der Roboter ab einer Simulationszeit von 3 Minuten umzufallen. Offensichtlich hatten die besten Individuen kein generell stabi-

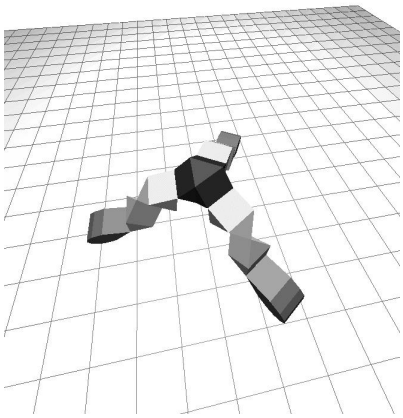


Abbildung 8.5: Ein dreibeiniger Roboter mit neun DOF.

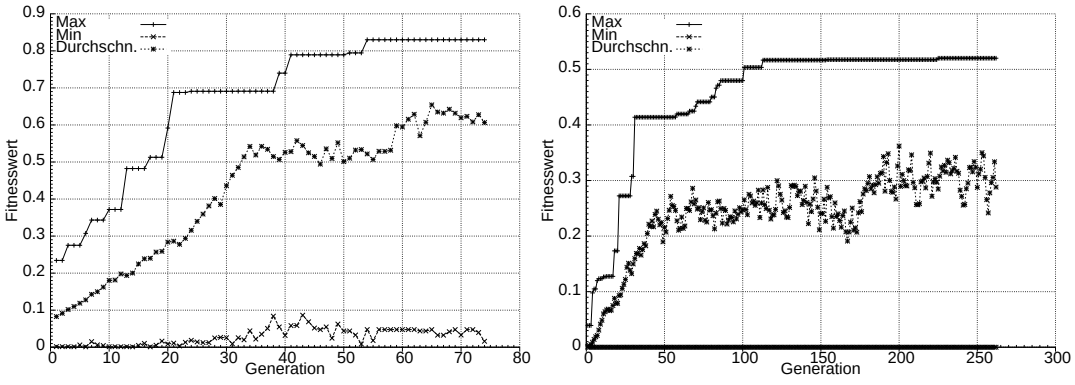


Abbildung 8.6: Fitnessverläufe von Experimenten mit dem dreibeinigen Roboter. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an. **Links:** Evolution mit der Funktion *simple fitness*. **Rechts:** Evolution mit der Funktion *nice walking fitness*.

les Laufen erzeugt, sondern ein Laufmuster, das nur im zur Fitnessbewertung beobachteten Zeitraum stabil war. Dieses Phänomen der Anpassung der Individuen an die Simulationszeit ist auch in anderen Experimenten beobachtet worden (siehe auch Abschnitt 9.2.3).

Ein dreibeiniger Roboter

Der dreibeinige Roboter besteht aus drei Beinen mit je drei Gliedern, die kreisförmig um einen Zentralkörper im Abstand von jeweils 120 Grad angeordnet sind. Die Beine sind kinematisch identisch mit den im vorherigen Abschnitt beschriebenen Beinen des zweibeinigen Roboters. In den Tabellen in Abschnitt B.4 sind weitere Parameter des Roboters und der Experimente aufgeführt.

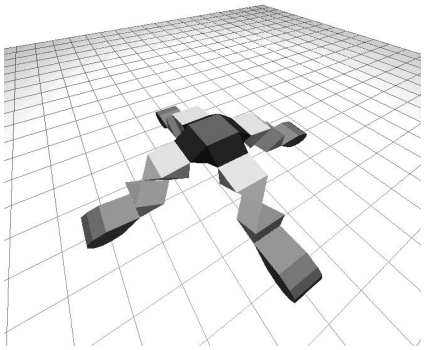
Die mit der Funktion *simple fitness* evolvierten Bewegungen ähneln der Bewegung, die der zweibeinige Roboter im vorherigen Abschnitt ausführt: Aus der Startkonfiguration (Abbildung 8.5) fällt der Roboter zu Beginn sofort zu Boden, wobei die drei Beine nebeneinander gelegt werden. Daran anschließend wird mit dem mittleren der drei Beine eine Schiebebewegung ausgeführt, die, in Verbindung mit kurzen, ruckartigen Abhebebewegungen des Torsos zu einer robbenden Vorwärtsbewegung führt. Die Bewegung ist erstaunlich

flüssig und verhältnismäßig schnell (ca. $1\frac{m}{s}$).

Wird für die Evolution die Funktion *nice walking fitness* verwendet, so ist die resultierende Bewegung völlig anders. Wiederum wird hier, wie beim Zweibeiner in Kombination mit der *nice walking fitness* der Torso von Anfang an auf Mindesthöhe gehalten, was in diesem Falle allerdings erheblich zu Lasten der maximal erreichten Geschwindigkeit geht (hier nur ca. 0.5 $\frac{m}{s}$). Der Bewegungsablauf ähnelt sehr stark der Bewegung des Zweibeiners mit *nice walking fitness*: Kurze, ruckartige Bewegungen eines Beines verursachen rutschende Bewegungen der anderen Glieder, die auf diese Weise nachgezogen werden, so dass eine deutliche Vorwärtsbewegung entsteht. Es wird scheinbar ein nicht unerheblicher Aufwand benötigt, um das Zentralglied des Roboters auf der geforderten Höhe zu halten, denn die Geschwindigkeit der Bewegung ist sehr niedrig. Die Evolution benötigt außerdem erheblich länger, um gute Bewegungen zu generieren.

Ein vierbeiniger Roboter

Der vierbeinige Roboter in diesem Experiment besteht aus vier im Winkel von 90 Grad zueinander um ein Zentralglied angeordneten Beinen. Diese Konfiguration (siehe Abbildung 8.7) ist nicht die aus der Natur bekannte Anordnung der Beine bei Vierbeinigen Lebewesen. In diesem Modell ist im Gegensatz dazu eine Punktsymmetrie der Beine zum Torsomittelpunkt vorhanden. Aus diesem Grund existiert auch keine Vorzugsbewegungsrichtung für den Roboter. In den Tabellen in Abschnitt B.5 sind weitere Parameter des Roboters aus Abbildung 8.7 und der Experimente aufgeführt.



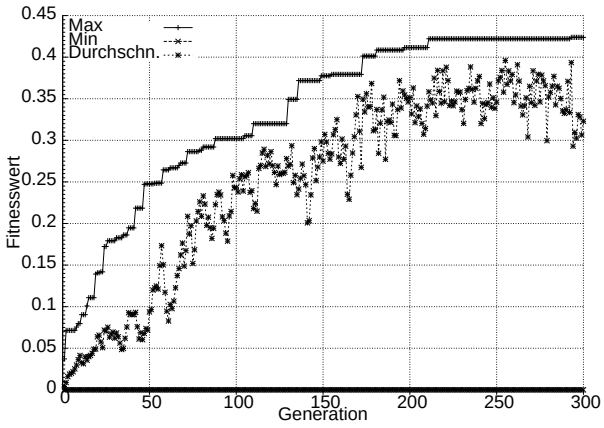


Abbildung 8.7: **Links:** Ein vierbeiner Roboter mit 12 DOF. Die Beine sind punktsymmetrisch angeordnet und kinematisch mit den Beinen des drei- und des zweibeinigen Roboters identisch. **Rechts:** Fitnessverlauf des Experiments mit dem vierbeinigen Roboter und der Funktion *nice walking fitness*. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an.

Der Roboter bewegt sich mit einer Variante des vom Dreibeiner und Zweibeiner bekannten Ganges fort. Wie dort wird auch in diesem Fall ein Bein ruckartig nach vorne bewegt, gefolgt von einer rutschenden Nachzieh-Bewegung der anderen drei Beine. Auf diese Weise wird eine stabile Vorwärtsbewegung erzielt.

Mit dem vierbeinigen Roboter aus Abbildung 8.7 sind Experimente zu verschiedenen Fragestellungen durchgeführt worden. Besonders der Einfluss der Operatorwahrscheinlichkeiten sowie die Robustheit des GP-Systems bezüglich des notwendigen Befehlsumfanges wird untersucht (Abschnitt 8.3.2). Der Vierbeiner ist für diese Untersuchungen wegen seiner komplexen Struktur ausgewählt worden.

Sechsbeiner

Der Sechsbeiner ist ein SIGEL-Modell, das eine gängige Roboterarchitektur widerspiegelt. Hierbei sind sechs Beine zu je drei an den Seiten des Roboters angeordnet, eine bei sechsbeinigen Insekten und bei biologisch inspirierten Robotern häufig anzutreffende Architektur, die eine hohe statische Stabilität im Grundzustand (Stehen) mit einer hohen Robustheit während der Fortbewegung kombiniert (es ist zu jedem Zeitpunkt möglich, drei Beine auf dem Boden zu haben). In den Tabellen in Abschnitt B.6 sind weitere Parameter des Roboters aus Abbildung 8.8 und der Experimente aufgeführt.

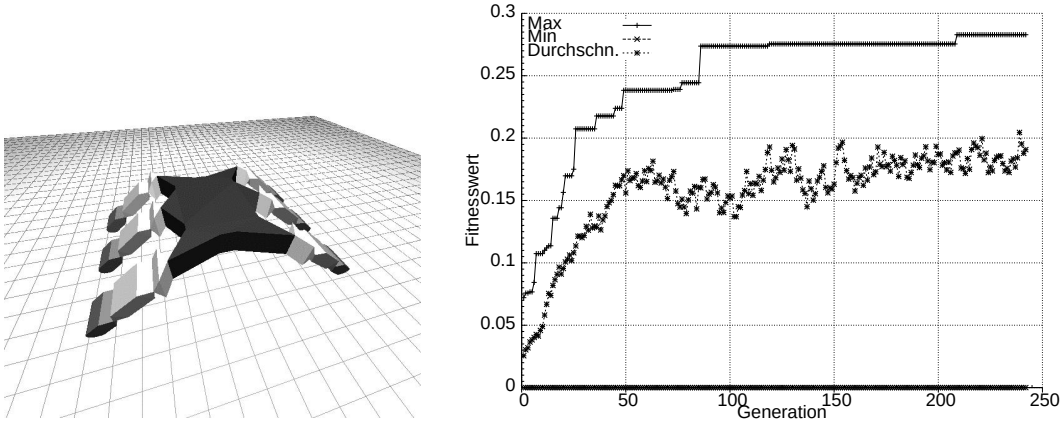


Abbildung 8.8: **Links:** Ein sechsbeiniger Roboter mit 18 DOF. **Rechts:** Fitnessverlauf des Experiments mit dem sechsbeinigen Roboter und der Funktion *nice walking fitness*. Die Fitness gibt die Geschwindigkeit des Roboters in $\frac{m}{s}$ an.

Der Bewegungsablauf dieses Roboters ist ein Extremfall der schon öfter beschriebenen ruckartigen Fortbewegung. Der Roboter versteift alle Beine um die Startkonfiguration zu erhalten, denn es wurde mit der Funktion *nice walking fitness* evolviert. Die Fortbewegung des Roboters wird durch ruckartiges Auf- und Abbewegen eines einzelnen Beines erzeugt. Durch den Impuls dieses Beines und der Impulserhaltung im Gesamtsystem wird der Roboter insgesamt in Richtung des Beines bewegt.

8.3.2 Einfluss der GP-Parameter auf die Evolution

Im Kapitel 7 wurde bereits der Einfluss der Operatorwahrscheinlichkeiten auf die Evolution untersucht und festgestellt, dass ein eindeutig positiver Einfluss selbst-regulierender Operatorwahrscheinlichkeiten nicht eindeutig nachweisbar ist. Hier wird nun anhand des vierbeinigen Roboters die Leistung des GP-Systems mit zwei unterschiedlichen statischen Operatorwahrscheinlichkeiten analysiert. Sie markieren jeweils extreme Einstellungen, die bei EA verwendet werden. Jedes Experiment ist jeweils 10 Mal mit unterschiedlichen Startwerten des Zufallszahlengenerators wiederholt worden, um zufällige Streuungen der Fitnessentwicklungen zu reduzieren. Die Einstellungen der Simulation, der Umgebung und des Roboters sind identisch mit den im vorherigen Abschnitt beschriebenen Experimenten mit dem vierbeinigen Roboter (bis auf die Operatorwahrscheinlichkeiten) und in den Tabellen in Abschnitt B.7, Abschnitt B.8 und Abschnitt B.9 aufgeführt. Als Fitnessfunktion wurde jedesmal die Funktion *nice walking fitness* gewählt.

Evolution mit hoher Mutationswahrscheinlichkeit

Bei diesen Experimenten wird die Mutationswahrscheinlichkeit auf konstant hohem Niveau gehalten, die Crossoverwahrscheinlichkeit ist dagegen gering. Die Parameter des GP-Systems für das Experiment sind in

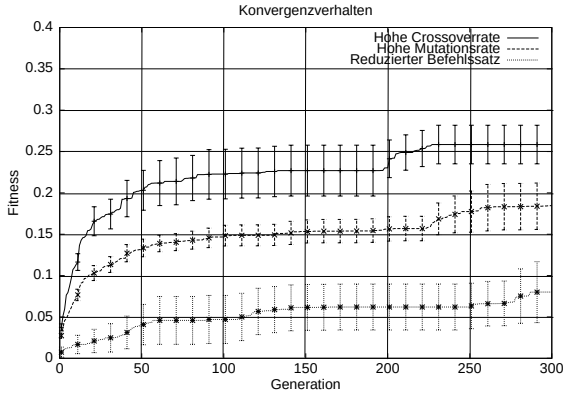


Abbildung 8.9: Durchschnittliche Fitness und Standardfehler der Experimente mit hoher Crossoverrate (oben), hoher Mutationsrate (mitte) und reduziertem Befehlssatz (unten). Deutlich zu erkennen ist die geringe Fitness der Serie mit reduziertem Befehlssatz.

Abschnitt B.7 aufgeführt. In Abbildung 8.9 ist der durchschnittliche Fitnesswert und der Standardfehler aufgeführt. Die Serie mit hoher Mutationswahrscheinlichkeit zeichnet sich durch eine geringere Performance gegenüber dem Experiment mit hoher Crossoverrate aus, bei gleichzeitig geringerem Standardfehler.

Evolution mit hoher Crossoverwahrscheinlichkeit

In dieser Serie von Experimenten wird die Crossoverwahrscheinlichkeit auf konstant hohem Niveau gehalten, die Mutationswahrscheinlichkeit dagegen ist klein. Die Parameter des GP-Systems sind in Abschnitt B.8 aufgeführt. In Abbildung 8.9 ist der durchschnittliche Fitnesswert und der Standardfehler aufgeführt. Bei diesen Experimenten ist die durchschnittliche Fitness höher als bei den anderen Serien, der Standardfehler ist allerdings auch größer, d.h. die Experimente zeigen sehr unterschiedliche Leistungen.

Evolution mit reduziertem Instruktionssatz

Bei dieser Serie von Experimenten steht die Frage nach der Robustheit des GP-Systems im Vordergrund. Der in den vorherigen Experimenten verwendete Befehlssatz ist umfangreich und umfasst neben den grundlegenden arithmetischen Funktionen noch zusätzliche Befehle (wie z.B. MIN, MAX, o.ä). Diese Befehle sind jedoch nur aufgrund von Vermutungen über ihre Nützlichkeit in den Standardbefehlssatz eingeflossen. Es ist nun interessant, ob dieser Standardsatz von Befehlen minimal ist oder ob ein GP-Experiment mit reduziertem Befehlssatz auch in der Lage ist, Laufrobotersteuerungen zu evolvieren. Hierfür wurde die Menge der erlaubten Befehle auf die Registermanipulationsbefehle ADD, COPY und LOAD, die Roboterbefehle MOVE und SENSE und auf eine einzige mathematische Fuktion, ADD, reduziert. Die Einstellungen der Simulation, der Umgebung und des Roboters sind ansonsten identisch mit den Einstellungen der vorherigen Experimente. In Abbildung 8.9 sind die Ergebnisse der Experimente dargestellt. Die erreichbare Fitness ist in diesem Fall sehr viel kleiner als bei den anderen Serien, allerdings ist auch mit diesem minimalen Befehlssatz erkennbar, dass der Roboter in der Lage ist, sich fortzubewegen. GP benötigt also im Prinzip nicht mehr Befehle als in Tabelle B.25 in Abschnitt B.9 aufgeführt sind, um Laufrobotersteuerungen zu evolvieren. Die erreichbare Geschwindigkeit mit erweiterten Instruktionssätzen ist allerdings sehr viel höher. Eine mögliche Begründung hierfür ist, dass die Registerinhalte, die die Stellwinkel bzw. die Stellmomente der Motoren angeben, durch Multiplikationen sehr viel schneller verändert werden können, als es mit einer einfachen Addition möglich ist. Für starke Änderungen der Gelenkwinkel müssen im Falle der Experimente mit reduziertem Befehlssatz mehrere Instruktionen verwendet werden. Die hierfür erforderliche Zeit ist

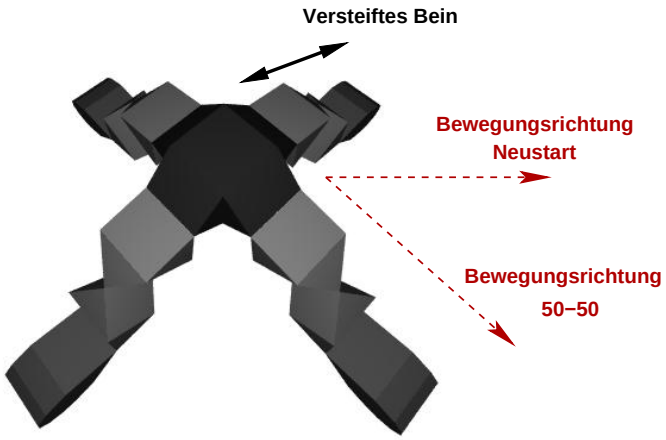


Abbildung 8.10: Der Vierbeiner mit versteiftem Bein. Die Vorzugsbewegungsrichtungen bei der Evolution mit neuer, zufälliger Startpopulation (Neustart) sowie bei Teilinitialisierung mit evolvierten Individuuen (50-50) sind unterschiedlich.

jedoch nur eingeschränkt vorhanden, da der Roboter in der aktuellen Position gehalten werden muss und deshalb auch für die anderen Gelenke die Stellwinkel/-momente berechnet werden müssen.

8.3.3 Untersuchung der Robustheit des Evolutionsprozesses

Das System SIGEL bietet die Möglichkeit, zu jedem Zeitpunkt die Variablen des Evolutionsprozesses zu verändern. Hinter dieser Eigenschaft steht die Motivation, dass ein versierter Experimentator geeignet in die Evolution eingreifen kann, um den Fortschritt des Experiments zu beschleunigen. Eine Problematik, die an dieser Stelle besonders untersucht wird, ist die Frage nach der Wiederverwendbarkeit bereits evolvierter Individuen. Ein Individuum (oder eine Menge davon) kann dann als wiederverwendbar bezeichnet werden, wenn der Evolutionsprozess durch die Aufnahme der Individuen beschleunigt wird oder die Qualität der besten Individuen signifikant besser ist. Die Frage nach der Wiederverwendbarkeit ist besonders wichtig, wenn versucht werden soll, die Anzahl der Auswertungen bzw. die Gesamtlaufzeit der Evolution zu minimieren.

Diese Fragestellung ist von besonderem Interesse, denn wenn ein Roboter marginal verändert wird, eine bereits erfolgreich evolvierte Population aber wieder verwendet werden kann (ggf. mit einer anschließenden Kalibrierungsphase), dann ist die zu erwartende Zeitspanne bis zum Erreichen eines ähnlichen Fitnessniveaus möglicherweise geringer. Um diese Hypothese zu testen, wurde der vierbeinige Roboter für eine weitere Serie von Experimenten verwendet.

Ein Bein des Roboters wurde versteift, d.h. die Gelenke dieses Beines sind durch das Steuerungsprogramm nicht länger zu bewegen (siehe Abbildung 8.10). Kinematische Details sind in Tabelle B.26 in Abschnitt B.11 aufgeführt). Diese Modifikation des Roboters hat einen realen Hintergrund, denn bei Laufrobotern werden die Gelenkmotoren stark belastet. Fällt ein Motor aus, bedeutet dies meist, dass das betreffende Gelenk nicht mehr frei bewegt werden kann, sondern bei einer zufälligen Auslenkung feststeht. Ein unvermeidlicher Totalausfall des Roboters kann dann vermieden werden, wenn es möglich ist, mit einer zum Teil aus erfolgreichen Individuen für die ursprüngliche Architektur bestehenden Population schnell neue, für die modifizierte Version des Roboters besser geeignete (wahrscheinlich aber langsamere) Laufprogramme zu evolvieren.

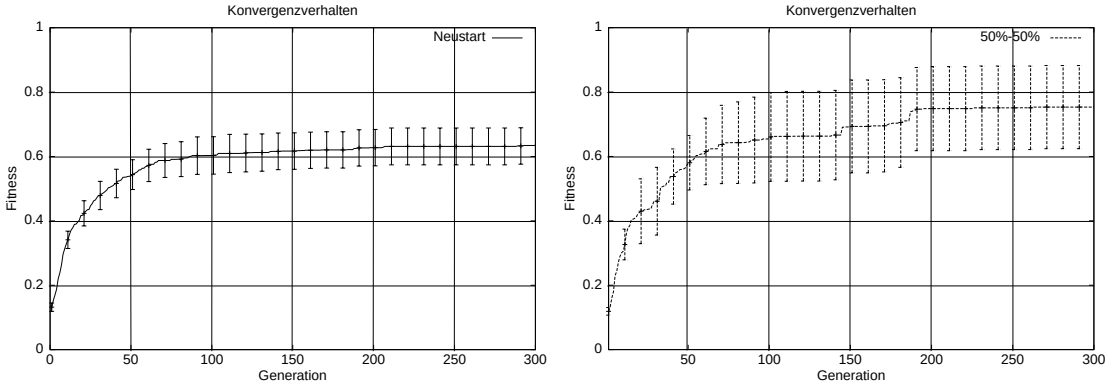


Abbildung 8.11: **Links:** Durchschnittliche Fitness und Standardfehler bei der Neuevolution mit dem modifizierten Vierbeiner. **Rechts:** Durchschnittliche Fitness und Standardfehler bei der Evolution mit dem modifizierten Vierbeiner und 50% zufälligen, 50% vorevolvierten Individuen.

Evolution mit zufälliger Startpopulation

In der ersten Serie dieses Versuches wurde ein komplett neues Evolutionsexperiment mit einer zufällig initialisierten Startpopulation durchgeführt (Parameter siehe Tabelle B.29). In Abbildung 8.11 links ist der Verlauf der Fitnesswerte der Neuevolution zu sehen. Zu beobachten ist, dass sich die Bewegungsrichtung des Roboters unterscheidet, wenn Laufprogramme dieses Experimentes oder Laufprogramme des Experimentes mit bereits evolvierten Programmen den Roboter steuern.

Evolution mit bereits evolvierten Individuen

Interessant ist der Vergleich des Fitnessverlaufes mit den Ergebnissen der zweiten Serie, bei der 50% der Individuen der Startpopulation bereits erfolgreich evolviert wurden (Abbildung 8.11). Parameter der Evolution sind in Tabelle B.27 aufgeführt. Die durchschnittliche Fitness ist nur geringfügig höher, der Standardfehler ist allerdings deutlich größer. Das beste Individuum dieser Serie war z.B. um 30% schneller als das beste Individuum bei der Neuevolution.

Bei der visuellen Darstellung der Bewegungen lässt sich allerdings sofort erkennen, dass die Evolutionen zu sehr unterschiedlichen Resultaten geführt haben. Die Neuevolution führte zu einer deutlich anderen Fortbewegungsart, deren Rhythmus an Bewegungen von vierbeinigen Lebewesen erinnert. Auch die Bewegungsrichtung unterscheidet sich von der Bewegungsrichtung des anderen Experimentes. Offensichtlich hat die veränderte Struktur des Roboters zu einem anderen Verlauf der Evolution geführt.

Die Individuen aus früheren Experimenten hingegen haben merklichen Einfluss auf die resultierenden Bewegungsabläufe: im zweiten Experiment ist die ruckartige, rutschende Bewegung früherer Experimente zu erkennen und auch die Bewegungsrichtung des Roboter bleibt unverändert.

Zusammenfassung

Die Ergebnisse der Experimente in diesem Kapitel haben deutlich gezeigt, dass GP in der Lage ist, für beliebige Roboterarchitekturen Laufprogramme zu evolvieren, die es Robotern in der Simulation ermöglichen, sich gehend fortzubewegen. Die Architektur der Roboter, d.h. die Anzahl der Freiheitsgrade und die kinematische Struktur, können dem System ohne Vorverarbeitungsschritt zur Extraktion von Modellwissen, wie es in traditionellen Methoden üblich ist, zur Verfügung gestellt werden. Essenziell für das Resultat ist eine

geeignete Formulierung der Fitnessfunktion: die Experimente haben deutlich gemacht, dass es notwendig ist, entsprechende mathematische Kriterien zu formulieren, um einen aufrechten Gang zu evolvieren.

Interessant ist, dass die evolvierten Programme nicht nur an die Architektur des jeweiligen Roboters angepasst werden, sondern auch die in der Dynamiksimulation herrschenden Bedingungen weitgehend ausnutzen. Die ruckartige, rutschende Fortbewegung, die von vielen Laufprogrammen erzeugt wird, ist nur aufgrund der jeweiligen Reibungs- und Materialkonstanten möglich. Werden die Konstanten verändert, versagen die meisten Programme sofort. Es ist weiterhin beobachtet worden, dass auch die Simulationszeit Einfluss auf die evolvierten Programme hat, indem Programme in manchen Fällen nur für den Zeitraum, der für die Ermittlung der Fitness berücksichtigt wird, eine stabile Fortbewegung erzeugen. Kurz nach dem Ende dieser Zeitspanne bricht das bis dahin stabile Laufmuster zusammen. In den meisten Fällen ist allerdings das Laufmuster ohne Zeitbeschränkung stabil.

Kapitel 9

Hybride Evolution in Simulation und Realität

Am Lehrstuhl für Systemanalyse wurde in Zusammenarbeit mit der Chalmers Universität in Göteborg, Schweden, ein Prototyp eines humanoiden Miniaturroboters entwickelt [13]. Zielsetzung des Projektes war es, einen Roboter zu konstruieren, der aus Standardbauteilen zusammengesetzt werden sollte, d.h. aus preiswerten und frei verfügbaren Komponenten. Gleichzeitig sollte die Gesamtgröße des Roboters 40 cm nicht überschreiten, damit für Experimente mit dem Roboter keine zusätzliche Experimentierumgebung (Labor) nötig ist. Weiterhin sollten die Dimensionen der einzelnen Elemente des Roboters so gewählt werden, dass der Roboter ein skaliertes Abbild der menschlichen Proportionen darstellt. Diese Vorgaben wurden umgesetzt und es entstanden die identischen Prototypen ZORC und ELVINA (Abbildung 9.1). Eine detaillierte Beschreibung der Konstruktion des Roboters ist in [13, 177] zu finden. Die Freiheitsgrade des menschlichen Vorbildes konnten allerdings wegen des beschränkten Bauraumes, vor allem im Hüftbereich des Roboters, nicht erreicht werden, so dass die Beweglichkeit des Roboters verglichen mit der des Menschen eingeschränkt ist (siehe Tabelle 9.1).

	Mensch	**Zorc**
Kopfgelenk	3	-
Schultergelenk	3	2
Torso	3	1
Hüftgelenk	3	2
Kniegelenk	1	1
Fußgelenk	2(3)	1
Gesamt	**24(26)**	**13**

Tabelle 9.1: Vergleich der Freiheitsgrade Mensch - ZORC. Nicht gezählt sind Gelenke des Kiefers sowie der Finger und Zehen, da sie nicht wesentlich für die Fortbewegung sind.

9.1 Der humanoide Miniaturroboter ZORC

Durch die festen Vorgaben (preiswerte Bauteile, einfache Montage, geringe Höhe und geringes Gewicht) entstanden allerdings auch Nachteile. Ein gravierender Nachteil der Konstruktion ist die geringe Präzision

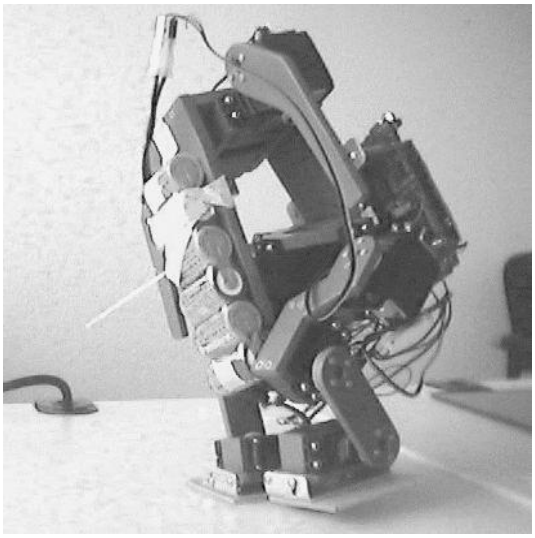


Abbildung 9.1: Die erste Version des humanoiden Miniaturroboters ZORC.

der Bauelemente (tragende Teile, Motoren) und die dadurch verursachte hohe Elastizität in der Gesamtkonstruktion. Elastizitäten stellen ein großes Problem für regelungstechnische Anwendungen dar, und zwar um so mehr, wenn nicht genormte Teile mit dementsprechend nicht bekanntem Biegungsverhalten verwendet werden. Das in den Motoren und Getrieben existierende Gelenkspiel führt zu weiteren Elastizitäten. Dies stellt eine weitere Erhöhung der Komplexität des Gesamtproblems dar, Laufprogramme für einen humanoiden Roboter mit GP zu evolvieren[1].

Trotz der Einschränkungen konnte anhand von manuell implementierten Laufprogrammen gezeigt werden, dass ELVINA und ZORC in der Lage sind, vollständig autonom zu laufen. Hierbei wurde von Anfang an darauf Wert gelegt, dass ZORC *immer* ohne externe Stromversorgung und *immer* mit onboard ausgeführtem Kontrollprogramm laufen sollte. Experimente mit externer Stromquelle haben gezeigt, dass das Verhalten der Stromquelle (ein Labornetzgerät) nicht dem Verhalten von Batterien gleichkam. So ist bei extremer Beanspruchung der Motoren, wie es zum Beispiel durch das Balancieren auf einem Bein verursacht wird, die Stromaufnahme des Roboters sehr hoch. Ein Labornetzgerät begrenzt den maximalen Strom jedoch, wohingegen die Batterien nahezu unbegrenzt Strom liefern. Andererseits ist der Roboter mit Batterien immer auch vom aktuellen Ladestand der Akkus abhängig, der sehr großen Einfluss auf die maximalen Haltekräfte bzw. Stellmomente hat. Die Motoren selbst haben unterschiedliche Kennlinien für Stellmoment und Winkelgeschwindigkeit bei unterschiedlichen Temperaturen, so dass direkt nach dem Einschalten des Roboters mit kalten Motoren andere Bewegungen erreicht werden, als nach Erreichen der Betriebstemperatur nach einigen Minuten. Werden diese Eigenschaften des Roboters bei der Programmierung nicht berücksichtigt, ist das Verhalten des Roboters nicht immer gleich[2].

Bald nach der Fertigstellung der Prototypen wurde deutlich, dass die Roboter in der vorliegenden Form nur bedingt für die gewünschten Aufgabenstellungen (Evolution von Laufalgorithmen) brauchbar waren. Zwar konnte mit einigem Aufwand ein simples Parameteroptimierungsexperiment durchgeführt werden [177], bei dem das Feedback über die Bewegung des Roboters durch eine Kamera errechnet wurde, für komplexe-

[1] Ein Wettlauf über 100 cm zwischen ELVINA und ZORC mit manuell erstellten Laufprogrammen fand im Februar 2001 im Rahmen der Forschungstage der Universität Dortmund statt. Sieger war ELVINA, ZORC musste wegen eines Motordefektes im Kniegelenk nach 20 cm aufgeben. Die manuelle Implementierung der Laufprogramme dauerte jeweils mehrere Wochen und zeigte deutlich den Bedarf nach einer automatischen Generierung von Laufalgorithmen.

[2] Unter der URL `http://ls11-www.cs.uni-dortmund.de/people/ziegler/robotics.html` ist ein Video der manuell erzeugten Laufbewegung zu finden.

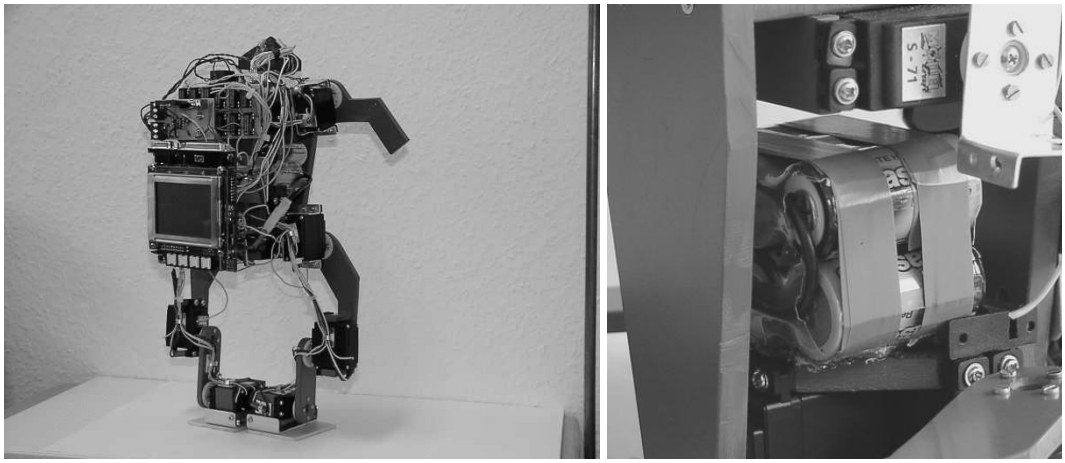


Abbildung 9.2: Der humanoide Miniaturroboter ZORC nach der Überarbeitung. Deutlich zu erkennen sind die verlängerten Beine und die Batterieaufbewahrung im Inneren des Torsos. Die Elektronik wurde für die Sensordatenerfassung erheblich erweitert.

re Experimente waren die Roboter zu einfach konstruiert und nicht mit der hierfür erforderlichen Sensorik ausgestattet. Die Geometrie der Beine und Arme, die nicht vorhandene Sensorik und das ungelöste Problem, die relativ schweren Batterien (die Batterien machen ca. 20% des Gesamtgewichtes aus) vorteilhaft zu positionieren, machten ein Redesign der Geometrie notwendig. Zahlreiche Sensoren wurden eingebaut, die das essenzielle Feedback für das GP-System liefern. Es wurden Drucktaster in den Füßen montiert, die Rückschlüsse über den Bodenkontakt der Fußsohlen zulassen. Eine Abfrage des aktuellen Gelenkwinkels jedes Servomotors ist jederzeit durch hinzugefügte Gelenksensoren möglich. Zusätzlich ist noch ein Kippsensor, der Neigebewegungen des Torsos misst, und ein Sensor für den Ladestand der Batterien vorhanden, so dass mit der erweiterten Sensorik dem GP-System nun ausreichende Informationen über den aktuellen Zustand des Roboters zur Verfügung gestellt werden können. Die zweite, verbesserte Version von ZORC ist in Abbildung 9.2 zu sehen.

Dieser Roboter wurde für die im Folgenden beschriebenen Experimente verwendet. Er hat ein Gewicht von ca. 2000 Gramm in voll funktionsfähigem Zustand, d.h. mit Batterie und Prozessor, und eine Höhe von ca. 40 cm. Der autonome Betrieb von ZORC wird durch das MK-4 Mikrocontrollerboard ermöglicht[3]. Es handelt sich dabei um ein speziell für den Einsatz in mobilen Robotern entworfenes Mikrocontroller-Board mit einem einfachen Multitasking-Betriebssystem im ROM und zahlreichen digitalen und analogen Schnittstellen. Es sind zusätzlich spezielle Anschlüsse für Gleichstrom- und Servomotoren vorhanden. Über eine serielle Schnittstelle können Programme auf den MK-4 überspielt und dort ausgeführt werden. Die Programme für den Mikrocontroller werden mit einem GNU-Crosscompiler auf einem LINUX-Rechner erzeugt.

Wichtig für das Funktionieren der späteren Kontrollprogramme ist eine Low-Level-Routine, die die Bewegungen der Servomotoren verlangsamt. Servomotoren aus dem Modellbau sind dafür ausgelegt, extern angegebene Stellwinkel schnell anzufahren und mit maximaler Kraft zu halten. Stellwinkeländerungen, die ein Kontrollprogramm generiert, führen zu ruckartigen Bewegungen, die den Roboter instabil werden lassen können, da die Motoren immer mit maximaler Winkelgeschwindigkeit den neuen Winkel anfahren. Die Geschwindigkeit der Motoren wird verlangsamt, indem der neue Stellwinkel durch Interpolation in Teilschritten angefahren wird. Die resultierende Bewegung ist dann weicher, wenn für die Gesamtbewegung eine ausreichende Zeitspanne zur Verfügung gestellt wird.

[3] Die Online-Dokumentation des MK-4 ist im Internet unter der URL `http://www.ee.uwa.edu.au/~braunl/eyebot/` verfügbar.

Abbildung 9.3: Der MK-4 Mikrocontroller von ZORC. Der MK-4 ist auf der Rückseite von ZORC montiert und erlaubt einen autonomen Betrieb des Roboters. Die verwendeten Batterien erlauben eine durchschnittliche Operationszeit von ca. 1h.

Betriebssystem	**Plattform/Taktfrequenz**	**Dauer**
Linux 2.4	AMD Athlon 850 MHz	11min 15s
Linux 2.4	AMD Athlon 1550 MHz	6min 15s
Solaris 5.7	Sun UltraSparc 170 MHz	74min 15s

Tabelle 9.2: Vergleich der Auswertungszeit eines 10s laufenden Kontrollprogrammes mit dem SIGEL-Modell von ZORC auf unterschiedlichen Plattformen.

Die Gelenkwinkel der einzelnen Servomotoren werden über eine eigens hierfür entwickelte Elektronik abgefragt und an den analogen Ports zur Verfügung gestellt. Es ist für ein Kontrollprogramm wichtig, Informationen über die aktuell anliegenden Gelenkwinkel zu bekommen, da aus der Differenz zwischen Soll- und Istwert (verursacht zum Beispiel durch zu hohe Belastungen oder durch Hindernisse) neue Regelgrößen berechnet werden können (und müssen).

9.2 Simulation von ZORC in SIGEL

Die Erfahrungen bei der Konstruktion und der Erweiterung von ZORC haben gezeigt, dass die Hardware sehr empfindlich und für eine Dauerbelastung, wie sie ein evolutionäres Experiment mit seinen vielen Evaluationen darstellt, nicht ausgelegt ist. Zum einen werden die Servomotoren bei der Evolution von Laufprogrammen sehr stark belastet, was die Lebensdauer der Motoren stark reduziert. Zum anderen ist der apparative und personelle Aufwand, der entsteht, wenn ein Experimentator oder ein automatisches System während des kompletten Experimentes für jede Evaluation und nach jedem unvorhergesehenen Zwischenfall ZORC in eine vordefinierte Startposition setzen muss, sehr hoch. Diese vornehmlich auf Kostenvermeidung beruhenden Gründe für eine Simulation werden zusätzlich untermauert durch den hohen Zeitbedarf, den eine einzelne Bewertung eines Laufprogrammes mit einem realen Roboter bedeutet. Trotz der durch die hohe Komplexität des Robotermodells verursachten langen Laufzeit einer Auswertung in der Simulation ist durch die parallele Auswertung vieler Laufprogramme gleichzeitig auf einem Rechencluster ein Laufzeitvorteil gegenüber dem Experiment mit dem realen Roboter gegeben. In Tabelle 9.2 ist z.B. ein Vergleich der Dauer einer Auswertung eines 10 Sekunden laufenden Laufprogrammes mit dem in den folgenden Abschnitten detailliert beschriebenen Modell von ZORC aufgeführt.

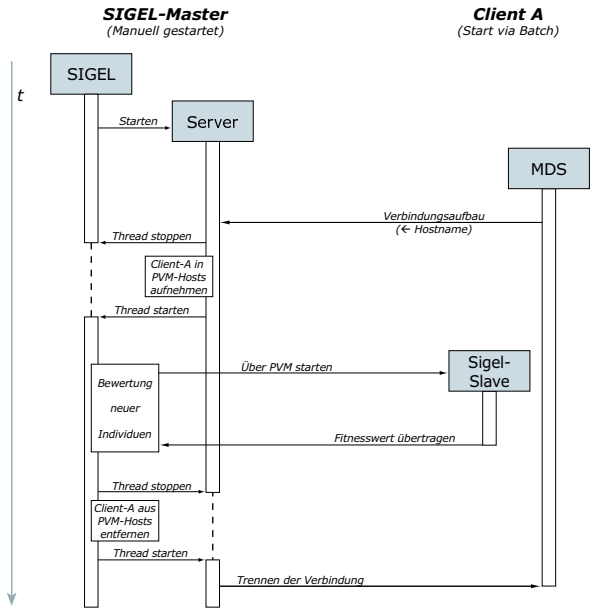


Abbildung 9.4: Parallelisierung der Evolution von Laufroboterprogrammen mit SIGEL und dem am Lehrstuhl für Systemanalyse installierten Batch-System bzw. PVM.

9.2.1 Parallelisierung

Die Tatsache, dass ein Experimentator nur eine begrenzte Zeit am Tag hindurch Auswertungen vornehmen kann, bedeutet, dass nur ca. 1000 Auswertungen pro Tag durchgeführt werden (bei einer angenommenen Arbeitszeit von 7h pro Tag). Bereits 5 PCs (AMD Athlon 1,6 GHz) sind in der Lage mehr Auswertungen in der gleichen Zeit durchzuführen. Am Lehrstuhl für Systemanalyse sind ca. 30 PCs dieser Leistungsklasse vorhanden, so dass die gleiche Anzahl Auswertungen bereits nach 4 Stunden erreicht wird, bzw. dass pro Tag ca. 7000 Auswertungen durchgeführt werden können. In Abbildung 9.4 ist schematisch die Parallelisierung der Auswertungen dargestellt.

Das SIGEL-System (Master) fordert Rechner für die Ausführung einer Simulation an. Die Verwaltung der zur Verfügung stehenden Rechner wird durch einen eigenen Thread (Server) realisiert. Stehen Rechner (Clients) bereit, werden sie in eine dynamische Host-Liste eingetragen. Über PVM werden dann auf diesen reservierten Rechnern die Simulationen gestartet. Nach Beendigung der Simulation wird die Fitness an den Master übetragen, der Prozess beendet und der Host aus der dynamischen Liste gelöscht. Die Anzahl der zur Verfügung stehenden Rechner für die Berechnung der Fitnesswerte kann dynamisch sein und ist abhängig von der aktuellen Last des Gesamtsystems.

9.2.2 Das Zwei-Phasen-Modell der Evolution

Durch die zur Verfügung stehenden Methoden zur Parallelisierung der Evolution und durch den bereits erwähnten zu erwartenden hohen Verschleiß der Roboter begründet, wurde ein Computermodell von ZORC für das System SIGEL entwickelt und die Evolution in einem ersten Schritt in der Simulation gestartet. In einer anschließenden Phase sollten dann das beste oder die besten Individuen auf dem realen Roboter ausgeführt werden. In Abbildung 9.5 ist der Ablauf der Evolution schematisch dargestellt.

Um die Ergebnisse der Evolution mit einem simulierten Modell des Roboters überhaupt übertragbar zu machen, musste bei der Modellierung von ZORC in SIGEL besonders auf eine genaue Übereinstimmung der Geometrie, der Massen und der Massenverteilung geachtet werden. Nicht modellierbare Eigenschaften

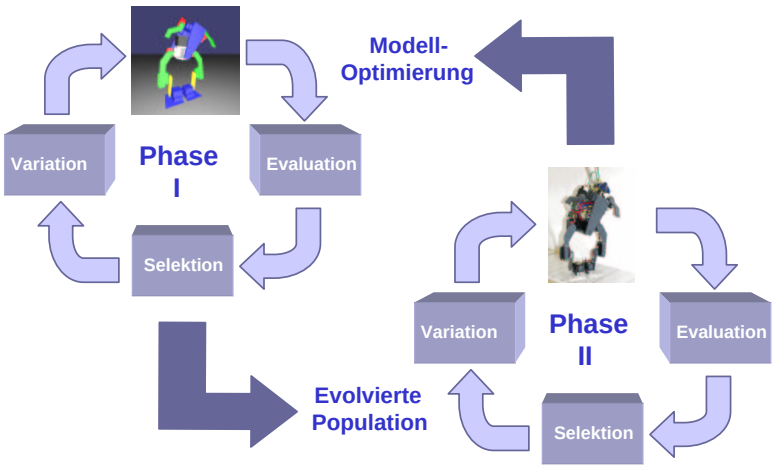


Abbildung 9.5: Schematische Darstellung der zwei Phasen der Evolution. In der Simulation evolvierte Programme werden als Startpopulation für die zweite Phase verwendet. Ergebnisse der zweiten Phase können als Kalibrierung für das Modell von Phase I verwendet werden.

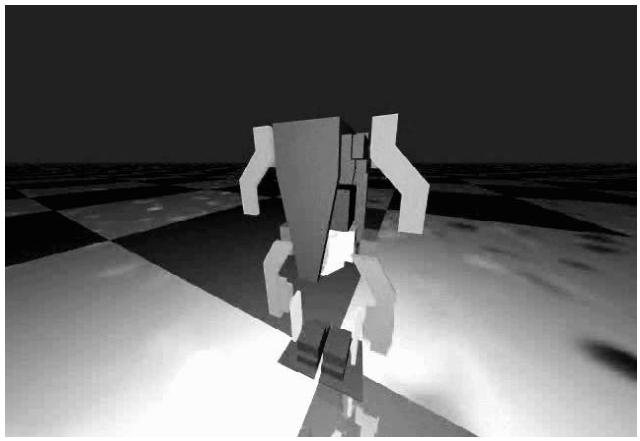


Abbildung 9.6: Der humanoide Miniaturroboter ZORC im Robotersimulationssystem SIGEL. Der Roboter wurde aus einfachen geometrischen Objekten konstruiert. Die Masse und Dichten der Glieder und die Kinematik entsprechen denen des realen Roboters.

des realen Roboters wie zum Beispiel die tatsächliche Gelenkreibung, die tatsächlichen Reibkonstanten der verwendeten Materialien oder die Elastizitäten der Konstruktion führen dazu, dass der Roboter in der Simulation ein anderes Verhalten aufweist als in der Realität. Der Modellierung sind allerdings immer Grenzen gesetzt, die prinzipiell verhindern, ein exaktes Abbild zu erstellen, so dass allein der Grad der Exaktheit entscheidet, inwieweit die Ergebnisse übertragen werden können. Der generelle Aufbau der zwei-phasigen Evolution erlaubt es, Unzulänglichkeiten, die durch Fehler in der Modellierung oder der numerischen Berechnung auftreten, zu überwinden. In der ersten Phase werden Experimente in der Simulation durchgeführt. Anschließend wird aus einer Menge von erfolgreich evolvierten Individuen, gegebenenfalls erweitert um zufällig erzeugte neue Individuen, eine Startpopulation für die zweite Stufe der Evolution gebildet. Bei

diesen Experimenten wird dann der reale Roboter für die Bewertung der Individuen verwendet.

Die zweite Phase der Evolution kann zusätzlich zur Evolution von Laufmustern mit dem realen Roboter verwendet werden, um das Modell des Roboters in der ersten Phase zu verbessern, so dass auf diese Weise in einer weiteren Iteration qualitativ hochwertigere Resultate erzielt werden können. In Abschnitt 9.3 wird die Notwendigkeit dieser Rückkopplung der Ergebnisse an einem Beispiel beschrieben.

Die Ergebnisse der Simulationen der ersten Phase bilden den Startpunkt einer weiteren Evolutionsschleife, die umso früher terminiert, je besser das Modell die wirklichen Gegebenheiten abgebildet hat. Nach der Kalibrierungsphase, während der die Ergebnisse der zweiten Phase vornehmlich zu Verbesserungen des Modells geführt haben, können die in der kombinierten Evolution generierten Kontrollprogramme erfolgreich, d.h. ohne Korrekturen, auf dem realen Roboter ausgeführt werden.

In Abschnitt 9.3 wird auf die generelle Problematik des Transfers von Kontrollprogrammen aus der Simulation auf einen realen Roboter im Detail eingegangen.

9.2.3 Die verwendeten Fitnessfunktionen

Im Gegensatz zu den in Kapitel 8 aufgeführten Experimenten, die den zurückgelegten Weg pro Zeiteinheit (*simple fitness*, Gleichung (8.1)) bzw. die Geschwindigkeit des Roboters unter Beibehaltung einer Mindest- bzw. Maximalhöhe (*nice walking fitness*, Gleichung (8.2)) als Fitnesswerte verwenden, ist für die Experimente mit ZORC eine komplexere Fitnessfunktion definiert:

$$fitness(i) = w_s \cdot \frac{x^i(t_{end}) - x^i(t_0)}{t_{end} - t_0} + w_h \cdot \frac{h^i_{av}}{h_{soll}}. \tag{9.1}$$

Auch Gleichung (9.1) verwendet die durchschnittliche Geschwindigkeit v des Roboters ($v = \frac{x^i(t_{end}) - x^i(t_0)}{t_{end} - t_0}$) während der Simulation bzw. die durchschnittliche Höhe eines ausgezeichneten Elementes des Roboters (h^i_{av}) relativ zu einer gegebenen Sollhöhe (h_{soll}) zur Berechnung der Fitness eines Laufprogrammes (Individuums).

Es hat sich allerdings gezeigt, dass nur durch die Auswahl einer geeigneten Kombination von Gewichtungsfaktoren w_s, w_h Bewegungsabläufe evolviert werden konnten, die dem humanoiden Laufen ähnlich sind. Wird durch einen großen Wert von w_s der Einfluss der Geschwindigkeit auf die Gesamtfitness erhöht, so sind die daraus resultierenden Bewegungsabläufe in der Regel Fall- oder Springbewegungen[4] mit hoher Geschwindigkeit. Wird im Gegensatz dazu das Einhalten einer definierten Höhe stark gewichtet (durch einen hohen Faktor w_h), so resultiert die Ausführung eines erfolgreich evolvierten Programmes darin, dass der Roboter in aufrechter Position stehenbleibt und sich nicht bewegt.

Wie in Abschnitt 8.3.3 bereits beobachtet, wurde auch hier bei manchen Experimenten eine Anpassung an die Simulationszeit registriert: Einzelne Individuen hielten den Roboter aufrecht, bis kurz vor Beendigung der Simulation eine Fall- oder Springbewegung eingeleitet wurde. Hierdurch wurde eine Vorwärtsbewegung erzielt, die nicht als Fallen bewertet wurde, da die Simulationszeit abgelaufen war, bevor das Individuum die Mindesthöhe h_{soll} unterschreiten konnte. Besonders interessant war das exakte Timing dieser Bewegung, die genau auf die identischen Startbedingungen der Dynamiksimulation sowie die immer gleiche Simulationszeit abgestimmt war. In Abschnitt 9.3 wird ebenfalls auf diese Problematik eingegangen.

In vielen Vorexperimenten musste also eine geeignete Kombination von w_s und w_h gefunden werden, die es erlaubt, Bewegungsprogramme zu evolvieren, die den Roboter in einer dem Menschen ähnlichen Weise bewegen.

[4] Unter der URL `http://ls11-www.cs.uni-dortmund.de/people/ziegler/robotics.html` sind Animationen dieser Fall- bzw- Springbewegungen zu finden.

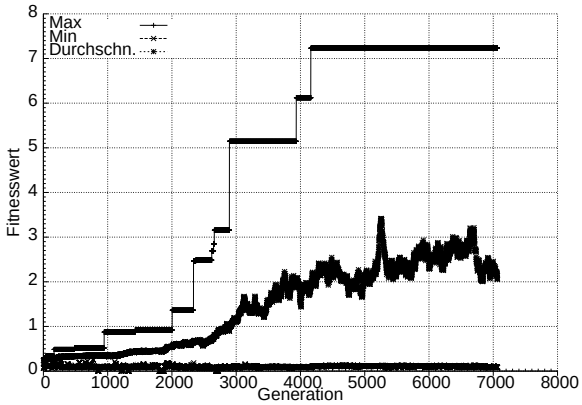


Abbildung 9.7: Verlauf der durchschnittlichen, minimalen und maximalen Fitnesswerte während des Experimentes. Verwendet wurde die Fitnessfunktion (9.2). Details zum Experiment sind in den Tabellen C.1 - C.5 in Anhang C aufgelistet.

9.2.4 Experimente

Als eine geeignete Fitnessfunktion mit guten Gewichtungsfaktoren w_s und w_h ist schließlich die Funktion (9.2) gewählt worden:

$$fitness(i) = w_s \cdot \frac{x^i(t_{end}) - x^i(t_0)}{t_{end} - t_0} + w_h \cdot \frac{\frac{1}{N+1}\sum_{t=0}^{N \cdot h} h^i(t)}{h_{soll}}, \quad N = 200.000, w_s = 100, w_h = 1, h = 5 \cdot 10^{-5} \quad (9.2)$$

Durch die Simulationsschrittweite von $h = 5 \cdot 10^{-5}$ und die Simulationszeit von $10s$ wird die Höhe des Roboters 200.000 Mal gemessen (N=200.000). Diese Werte sind für alle Experimente konstant. In den Tabellen Abschnitt C.1 sind weitere Parameter des Experimentes aufgeführt.

Unter der URL `http://ls11-www.cs.uni-dortmund.de/people/ziegler/robotics.html` sind im Internet Videos der evolvierten Bewegungsabläufe des simulierten ZORC verfügbar.

Bei den Experimenten wird ebenfalls die Methode des vorzeitigen Abbruches der Simulation verwendet (siehe Abschnitt 7.2). Unterschreitet die Höhe des Roboters einen definierten Wert (hier: 66% der Sollhöhe h_{soll}), so wird die Simulation abgebrochen und die Fitness des Individuums mit einer angenommenen Höhe $h^i(t) = 0$ für alle noch folgenden Zeitschritte bis $t_{end} = 10s$ berechnet. Auf diese Weise werden schlechte Individuen mit einem schlechten Fitnesswert „bestraft“ und gleichzeitig durch den vorzeitigen Abbruch wertvolle Rechenzeit gespart. Der Gesamtalgorithmus konnte durch den vorzeitigen Abbruch dieser Individuen um bis zu 900% beschleunigt werden. In Abbildung 9.7 ist der Verlauf der Fitnesswerte eines Experimentes mit der Fitnessfunktion aus Gleichung (9.2) abgebildet.

Die Bewegung des Roboters wird durch schnelle, kleine Schritte erzeugt, die den Roboter mit einer Geschwindigkeit von $7\frac{cm}{s}$ laufen lassen. Interessanterweise ist das beste Programm (nach Gleichung (9.2)) nur 145 Zeilen lang. Die schnelle Bewegung ist möglich, da die Instruktionen eine simulierte Ausführungsdauer von nur $0.001s$ besitzen. Hierdurch sind schnelle Wechsel der Sollwinkel für die Motoren möglich. Im Abschnitt 9.3 wird deutlich, dass gerade die Ausführungsdauer der Instruktionen große Wirkung auf das Verhalten des Roboters hat und genau eingestellt werden muss, um die Übertragbarkeit von Programmen auf den realen Roboter zu gewährleisten.

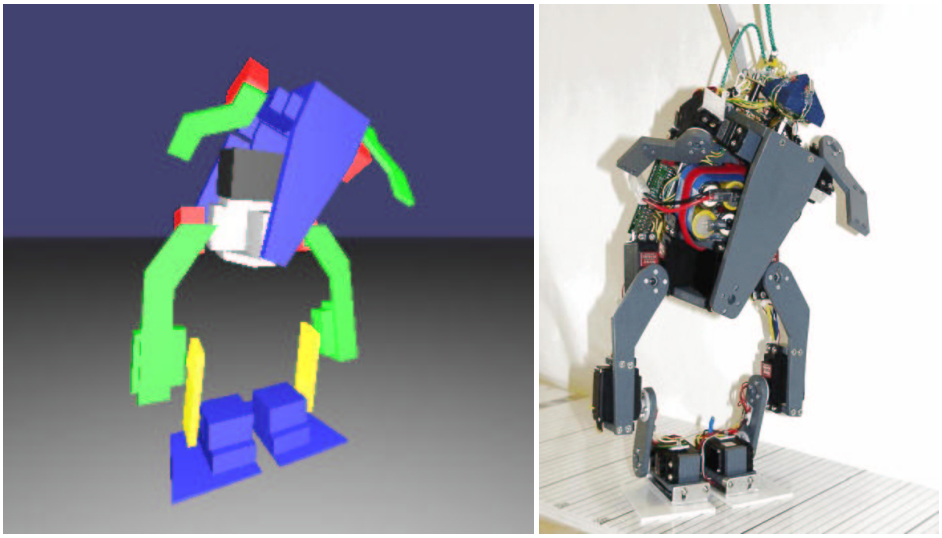


Abbildung 9.8: Der humanoide Miniaturroboter ZORC im Robotersimulationssystem SIGEL und in der Realität. Es wird auf beiden Robotern das *identische* Kontrollprogramm ausgeführt.

9.3 Transfer der Ergebnisse auf den realen Roboter

Im Folgenden wird ein Experiment beschrieben, das erfolgreich die vorevolvierten Individuen der ersten Phase der Evolution an die Gegebenheiten des realen Roboters anpasst.

Als Startpopulation für die zweite Phase der Evolution wird eine Mischung aus den besten Individuen aus den Simulationsexperimenten zusammengestellt, zusätzlich werden noch Individuen in die Startpopulation eingefügt, die ein manuell erstelltes Programm enthalten. Dieses Programm bewegt die Motoren in einer Weise, dass ZORC aufrecht stehen bleibt, ohne sich zu bewegen. In Abbildung 9.8 ist die Position zu sehen, in der ZORC verharrt, nachdem das Programm ausgeführt wurde. Deutlich sichtbar ist, dass sowohl der in SIGEL simulierte als auch der reale Roboter die gleiche Position einnehmen. Es wurde in beiden Fällen exakt das gleiche Programm ohne jegliche Änderung ausgeführt.

Auch für Phase zwei wurde wiederum SIGEL verwendet. Es wurde eine spezielle Fitnessfunktion entwickelt, die es erlaubt, aus SIGEL heraus einen externen Roboter zu steuern. Hierfür wird das Laufprogramm des zu bewertenden Individuums über die serielle Schnittstelle auf ZORC bzw. den MK-4-Mikrocontroller übertragen. Dort wird das Programm ausgeführt. Hierfür ist ein spezieller Interpreter implementiert worden, der Programme, die mit SIGEL evolviert wurden, ausführt und den Roboter steuert.

Die Programme werden manuell bewertet. Die Bewertung erfolgt hierbei anhand folgender Kriterien:

- Ein Programm wird dann als gut bewertet, wenn es den Roboter aufrecht halten kann. Zu Anfang der Evolution wird aufrechtes Stehen stärker positiv bewertet als gegen Ende der Evolution, um einen entsprechenden Selektionsdruck zu erzeugen.
- Fällt ein Roboter, so wird eine Fallbewegung nach vorne besser bewertet als eine Fallbewegung nach hinten. Hierdurch wird eine Vorwärtsbewegung durch die Selektion bevorzugt.
- Führt ein Roboter erratische Bewegungen aus, so ist die Fitness Null. Hierdurch werden potenziell schädliche Bewegungen aus der Population entfernt.
- Gleichmäßige, oszillierende Bewegungen werden besser bewertet als nicht-periodische Bewegungen. Hierdurch wird dem Umstand Rechnung getragen, dass Laufen eine periodische Bewegung ist.

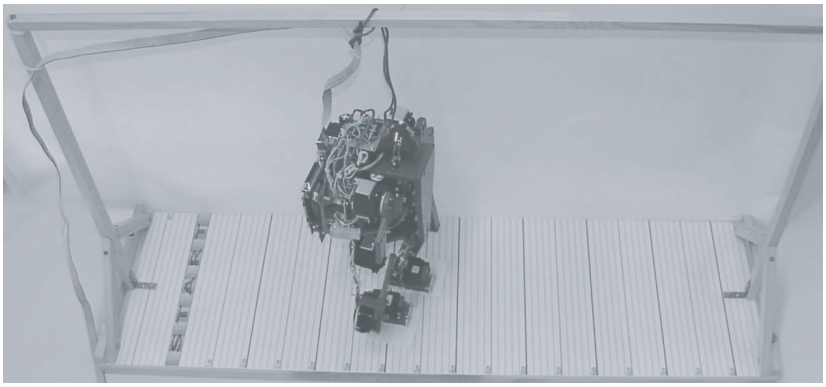


Abbildung 9.9: Der „Laufstall“. Um ein Umfallen des Roboters zu vermeiden, ist er an einem Querbalken ohne Einschränkungen der Bewegungsfreiheit befestigt. Die zurückgelegte Strecke kann am bemaßten Untergrund abgelesen werden.

- Ästhetische Eindrücke als Bewertungsgrundlage sind ausdrücklich erwünscht. „Schöne“ Bewegungen oder Bewegungsansätze werden besser bewertet als andere Bewegungen.

Um eine gewisse Objektivität der Bewertungen zu gewährleisten, wurden die Bewegungen auf einem bemaßten Untergrund durchgeführt, so dass der während der Ausführung des Laufprogrammes zurückgelegte Weg gemessen werden konnte. Insbesondere in der Anfangsphase der Evolution kommt einer differenzierten Bewertung der Individuen durch einen menschlichen Betrachter besondere Bedeutung zu, denn hier werden rein subjektive Eindrücke der Bewegung in numerische Werte umgesetzt. Hierzu zählen unter anderem die Symmetrie der Bewegung, die „Weichheit“, etwaige erratische Zwischenbewegungen etc. Durch die Bewertung eines Individuums werden solche in der Struktur der Individuen enthaltene Bewegungsmuster gefördert oder unterdrückt, so dass die Suchrichtung der Evolution hiermit entscheidend geprägt wird. Dies ist gleichzeitig aber auch ein bedeutender Nachteil, denn durch eine „falsche“ Bewertung von Individuen, die großes Potenzial im Sinne der Fitnessfunktion besitzen, kann der evolutionäre Prozess deutlich verlangsamt werden. Gerade in der Anfangsphase, in der zufällig erzeugte Programme bewertet werden müssen, sind oft nur graduelle Unterschiede zwischen rein zufälligen Programmen und Programmen mit ersten strukturierten Bewegungen erkennbar, so dass es häufig zu Fehlern in der Bewertung kommen kann.

Dass dennoch erfolgreich Laufprogramme evolviert werden, ist der generellen Robustheit der EA (siehe auch Abschnitt 7.4 zum Einfluss von Fehlklassifikationen) zu verdanken. In Abbildung 9.9 ist der Aufbau für die Experimente mit ZORC abgebildet. Die Bewertung eines Individuums erfolgt manuell durch die Eingabe eines Wertes als Fitness für das ausgeführte Laufprogramm.

Die Verwaltung der Population, die Variation der Individuen etc. wird weiterhin vom SIGEL-GP-System vorgenommen. In den Tabellen in Abschnitt C.2 sind die Parameter des Experimentes aufgeführt. In Abbildung 9.10 ist die Entwicklung der Fitnesswerte des manuell durchgeführten Experimentes dargestellt. Hierbei sind nicht die tatsächlichen Werte von Interesse, da sie nur die subjektiven Einschätzungen der jeweiligen Laufmuster repräsentieren, sondern die Entwicklung der Werte.

Ein Vergleich der besten Programme der Experimente mit simuliertem und realem Roboter zeigt deutliche Unterschiede in der Struktur auf. In Abbildung 9.11 sind Histogramme der Häufigkeiten der Instruktionen in beiden Programmen dargestellt. Interessant ist, dass das beste Laufprogramm der Evolution mit dem realen Roboter vollständig auf SENSE-Anweisungen verzichtet. Dies ist deshalb umso bemerkenswerter, als erheblicher technischer Aufwand (beim realen Roboter) notwendig war, um die SENSE-Anweisungen überhaupt ermöglichen zu können. Es ist aber anscheinend für ein erfolgreiches Laufprogramm nicht notwendig, Informationen über die aktuellen Gelenkwinkel des realen Roboters zu erhalten. Ein Grund hierfür

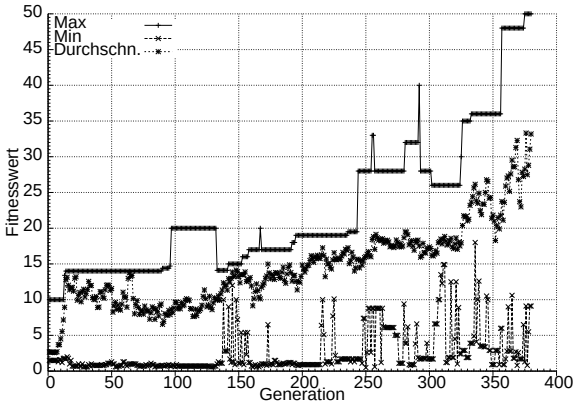


Abbildung 9.10: Verlauf der minimalen, maximalen und durchschnittlichen Fitnesswerte des Experimentes mit dem realen Roboter und manueller Bewertung der Fitness.

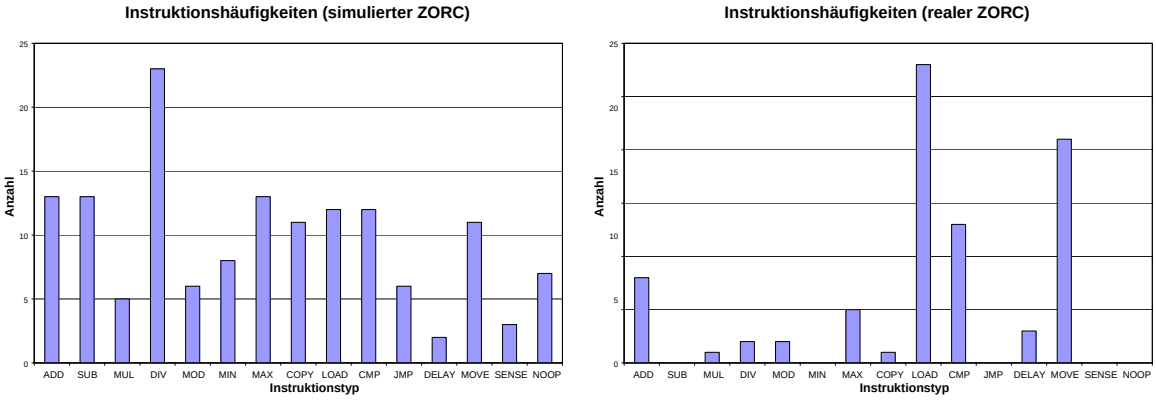


Abbildung 9.11: Vergleich der Häufigkeiten von Instruktionen im jeweils besten Individuum der Experimente mit SIGEL (links) und mit dem realen ZORC (rechts).

könnte auch darin liegen, dass zwischen der Soll- und der Istposition der Servomotoren kein Unterschied besteht, so dass auf eine zeitaufwändige Messung der Daten verzichtet wird und die Sollposition als erreicht angenommen wird. Demzufolge kann nicht von einer kontrollierten Bewegung des Roboters ausgegangen werden, da keine Informationen über den aktuellen Zustand des Roboters für die Berechnung der folgenden Bewegungen berücksichtigt werden. Der Roboter wird „blind“ gesteuert.

Andererseits ist in diesem Programm die MOVE-Anweisung überproportional häufig vertreten. Eine Erklärung für diese Anhäufung könnte darin liegen, dass der Roboter zu Beginn der Ausführung des Programmes erst einmal in der Startposition gehalten werden muss. Hierfür müssen für alle Motoren Stellwinkel berechnet und an die Motoren gesendet werden, da sonst der Roboter in sich zusammenfallen würde. Erst danach kann mit der eigentlichen Bewegung begonnen werden. Ermöglicht wird eine „blinde“ Steuerung des Roboters ohne Feedback der Sensoren zusätzlich dadurch, dass für alle Programme identische Anfangsbedingungen herrschen. Durch Variation der Anfangsbedingungen (beispielsweise eine randomisierte Initialkonfiguration) oder durch bewusste Zufallselemente während der Bewegung bzw. der Simulation kann eventuell erzwungen werden, dass zur Laufgenerierung die internen Variablen des Roboters verwendet werden. Entsprechende Untersuchungen konnten wegen den extrem zeitaufwändigen Experimenten in Simulation und Realität und dem hohen Verschleiß des Roboters nicht durchgeführt werden.

Kapitel 10

Evolution von Laufmustern mit realen Robotern

Nachdem in Kapitel 8 die Evolution von Laufrobotersteuerungen mit simulierten Robotern und in Kapitel 9 die hybride Evolution von Programmen in der Simulation mit anschließender Kalibrierungsphase präsentiert worden sind, wird in diesem Abschnitt die Evolution von Laufmustern ausschließlich mit realen Robotern ohne Simulation präsentiert.

Die verwendeten Roboter sind Sonys ERS-2100 Roboter (Abbildung 10.1), die als kommerzielles Produkt unter dem Namen Aibo[1] vermarktet werden. Diese Roboter werden als Fußballroboter in der *Sony Legged League* eingesetzt. Als Teil des *GermanTeam* ist die Universität Dortmund unter Federführung des Lehrstuhls für Systemanalyse am Fachbereich Informatik mit einem eigenen Team[2] in dieser Liga vertreten. Die Aibo-Roboter werden völlig frei programmierbar ausgeliefert, es ist keine vorgefertigte Software vorhanden. Alle Teilnehmer an der Sony Legged League dürfen nur mit unmodifizierten Robotern teilnehmen, so dass bezüglich der Hardware alle Teams gleich ausgestattet sind. Dies hat den Vorteil, dass ausschlaggebend für Sieg oder Niederlage allein die implementierten Algorithmen sind. Diese Tatsache ist in anderen Ligen der RoboCup-Federation[3] nicht immer gegeben, im Gegenteil, oft sind hohe Ausgaben für hochwertige Roboterteile direkt verantwortlich für ein gutes Abschneiden der Teams.

Für den Erfolg eines Teams von autonomen Laufrobotern im Roboterfußball sind viele Faktoren wichtig. Hierzu zählen natürlich effiziente und präzise Lokalisierungsmechanismen, die robuste Kommunikation zwischen Teammitgliedern und natürlich eine schnelle und sichere Fortbewegung. Gerade hier hat in der Vergangenheit der Schlüssel zum Erfolg gelegen. So hat z.B. der mehrmalige Weltmeister, das Team der University of New South Wales (UNSW) aus Australien, hervorragende Laufalgorithmen verwendet, die ein schnelles und genaues Ansteuern von Positionen auf dem Spielfeld erlaubten.

10.1 Evolution von Laufmustern für Sony AIBO Roboter

Im GermanTeam kommt ein Algorithmus zum Einsatz, der aus dem aktuellem Zustand der Beine und vorgegebener Bewegungsrichtung und -geschwindigkeit zentral die Sollpositionen und -orientierungen der

[1] Näheres über die Aibo Roboter ist im Internet unter der URL `http://www.aibo.com` zu finden.

[2] Die RUHRPOTT HELLHOUNDS (erreichbar im Internet unter der URL `www.fussballhun.de`) sind ein von einer Projektgruppe des Fachbereiches Informatik im Jahre 2001 gegründetes Roboterteam.

[3] Es existieren zwei verschiedene Organisationen, die Roboterfußball-Wettkämpfe organisieren. Zum einen die RoboCup-Federation (`www.robocup.org`), zu der auch die Sony Legged League gehört, und die Federation of International Robot-soccer Association (FiRa, `www.fira.net`)

Abbildung 10.1: Aibo, vierbeiniger Laufroboter der Firma Sony.

Füße und darauf aufbauend die Sollwinkel aller Beingelenke mit einer inversen Kinematik (der InvKinWalkingEngine, siehe [143]) des Aibo berechnet. Die Aibo-Roboter des GermanTeams (Abbildung 10.1) sind vollständig autonome Roboter, denn sie besitzen einen Sony MIPS2000 Prozessor onboard, eine Stromversorgung über Batterien und zahlreiche Sensoren. In Tabelle D.1 im Anhang D sind einige technische Details der Aibo-Roboter aufgeführt. Über Mikroschalter kann für jeden Fuß das Aufliegen auf dem Boden erkannt werden, Beschleunigungssensoren für jede Achse können zur Beschreibung der Orientierung im Raum verwendet werden. Zusätzlich sind Drucktaster auf Rücken und Kopf vorhanden, die z.B. zu Beginn eines Roboterfußballspieles zur Initialisierung gedrückt werden können. Wichtigster Sensor ist eine Farb-CCD-Kamera im Kopf des Roboters, die zur visuellen Orientierung dient.

Der Aibo-Roboter verfügt über 20 Freiheitsgrade (20 DOF), von denen 12 der Steuerung der Beine dienen. Die übrigen Motoren steuern Kopf, Schwanz, Ohren und Mund des Roboters und sind für die Laufbewegungen nicht relevant.

Im GermanTeam wird ein parametrisierbares Laufmuster auf Basis einer inversen Kinematik zur Generierung von Laufmustern verwendet [142]. Eine Gehbewegung wird realisiert, indem die Fußspitzen der vier Beine des Roboters entlang einer für jedes Bein vordefinierten Trajektorie geführt werden. Werden die Fußspitzen aller Beine gleichzeitig koordiniert entlang dieser Bahnen geführt, entsteht eine Laufbewegung des Aibos. Die Beinbewegungen werden in zwei Phasen geteilt: In der Schwingphase werden die Fußspitzen in Kreisbahnen in der Luft bewegt, während der Stemmphase werden sie linear am Boden geführt. Ausgehend von der vorgegebenen Sollposition der Fußspitzen werden die Gelenkwinkel der Beine des Hundes bestimmt, indem eine Konfiguration des Beines berechnet wird (also eine Kombination von Stellwinkeln der Gelenke im Bein), die ein Erreichen der Sollposition sicherstellt. Auf welche Art und Weise die Trajektorien der Fußpositionen berechnet werden, wird durch die Parameter der InvKinWalkingEngine definiert. In Tabelle D.2 in Anhang D sind diese Parameter der InvKinWalkingEngine vollständig aufgeführt. Die Bewegungen, die aus dieser Transformation resultieren, unterscheiden sich in für die Robustheit und Geschwindigkeit der Bewegungen relevanten Details.

Über die verschiedensten Wertekombinationen dieser Parameter können unterschiedliche Laufmuster erzeugt werden. Die im GermanTeam vor 2001 verwendete Kombination von Parametern war das Resultat von manuellen Experimenten kombiniert mit der Erfahrung des Entwicklers der InvKinWalkingEngine. Um diese nicht befriedigende Einstellung zu verbessern, wurde mit Hilfe eines GA eine automatisierte Optimierung dieser Parameter implementiert.

Um die Fitness eines Individuums während der Evolution zu bestimmen, wird die inverse Kinematik mit

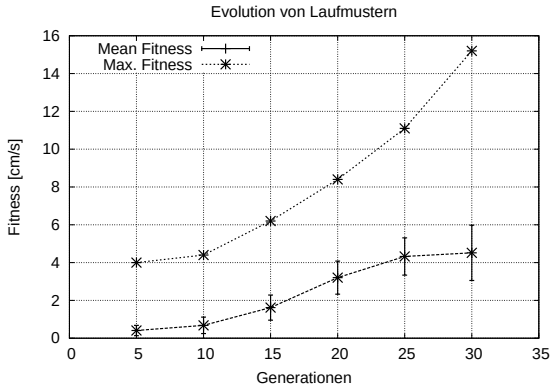


Abbildung 10.2: Verlauf der Fitnesswerte (maximal und durchschnittlich mit Varianz) während der Evolution. Die gesamte Population ist hierbei alle 5 Generationen einmal komplett getestet worden.

den Parametern eines Individuums berechnet (die Zuordnung der einzelnen Werte zu den Parametern der inversen Kinematik ist in Tabelle D.3 aufgelistet) und das resultierende Laufmuster anschließend vom Roboter ausgeführt. Eine automatische Berechnung der Fitness mit einem Bilderfassungssystem ist im Prinzip möglich, technisch jedoch sehr aufwändig und wurde deshalb nicht realisiert. So ist zum Beispiel für jeden Roboter eine definierte Startposition notwendig, auch müssen eventuelle Stürze erkannt werden. Zum automatischen Aufrichten des Roboters ist eine zusätzliche Mechanik notwendig, die den technischen Aufwand des Experimentes sehr erhöht. Die Individuuen wurden stattdessen paarweise manuell bewertet. Erfahrungen mit den Experimenten mit ZORC in Kapitel 9 haben für diese Experimente zur Wahl der Turnierselektion geführt, die allein auf der Basis von Vergleichen funktioniert. Für die Turnierselektion ist eine exakte Angabe der Fitness als numerischer Wert nicht notwendig, es reicht eine Klassifikation nach Tabelle 10.1. Gleichzeitig wird dadurch das Problem umgangen, in der ersten Phase der Evolution eigentlich ununterscheidbaren Individuen numerische Fitnesswerte zuweisen zu müssen.

10.2 Interaktive Evolution von Laufmustern

Die Parameter der InvKinWalkingEngine stellen das Genom eines Individuums des GA dar. Die Zuordnung der Elemente des Genoms eines Individuums zu den Parametern der inversen Kinematik ist in den Tabellen D.2 und D.3 aufgeführt. Während der durchgeführten Experimente sind einige Parameter mit Standardwerten belegt worden (zum Beispiel die Parameter zur Korrektur von geneigtem Laufen, oder GT2002-spezifische Parameter, die die minimale Zeitdauer zwischen zwei unterschiedlich parametrisierten Laufmustern festlegen), so dass für die tatsächliche Parametrisierung der inversen Kinematik die Zahl der Parameter von 27 auf 16 reduziert werden konnte. Ein Individuum des GA besteht also aus einem Vektor von 16 ganzzahligen Werten für die inverse Kinematik.

Die Population wurde mit dem Parametervektor initialisiert, der im GermanTeam für die Erzeugung des Standard-Laufmusters verwendet wird. Die Parameter wurden für die Initialisierung durch Addition einer standard-normalverteilten Zufallsvariable verrauscht. Die Turniere während der Evolution wurden manuell bewertet, wobei kein Individuum an mehr als einem Turnier einer Generation teilnehmen kann. Nach je fünf Generationen wurde die Geschwindigkeit aller Individuen in der Population manuell gemessen, um einen objektiven Eindruck vom Fortschritt der Fitnesswerte zu bekommen. In Abbildung 10.2 ist ein deutlicher Anstieg der maximalen und der durchschnittlichen Fitness der Population im Verlauf der Evolution zu sehen. Die Parameter des GA für diesen Lauf sind in Tabelle D.4 aufgeführt. Interessant ist, dass schon nach wenigen Generationen die Geschwindigkeit des im GermanTeam verwendeten Standard-Laufmusters von

Klasse	Bedeutung
c_0	i_2 besser als i_1
c_1	i_1 besser als i_2
c_2	i_1 und i_2 nicht unterscheidbar

Tabelle 10.1: Mögliche Ausgabeklassen des Klassifizierers C'.

ca. $0.14\frac{m}{s}$ übertroffen wird. Für das beschriebene Experiment sind 30 Generationen ausgewertet worden, was bei einer Populationsgröße von 26 Individuen 780 Auswertungen bedeutet. Jede dieser Auswertungen wurde manuell mit einem realen Roboter durchgeführt. In den folgenden Abschnitten wird nun die in Abschnitt 7.4 erläuterte Methode der Evolution mit einem Meta-Modell der Fitnessfunktion verwendet, um möglichst wenige Auswertungen für die Evolution von guten Parameterkombinationen zu benötigen.

10.3 Ableiten eines Meta-Modelles der Fitnessfunktion

In diesem Experiment wird die Evolution von Klassifizierern anhand der Evolution von Laufmustern für den Aibo-Roboter demonstriert. Während des Experimentes im vorherigen Abschnitt sind Daten über die Individuen und die Turniervergleiche aufgezeichnet worden. Die Daten wurden in der Form

$$V' = < i_1, i_2, c > \tag{10.1}$$

gespeichert, wobei c den Ausgang des jeweiligen Vergleiches in drei Klassen unterteilt und die Individuen i_1, i_2 jeweils 16-Dimensionale Vektoren reeller Zahlen darstellen, deren Bedeutung für die zu Grunde liegende inverse Kinematik nach [142] in Tabelle D.3 aufgeführt sind. Die drei Klassen sind in Tabelle 10.1 aufgelistet.

Anhand dieser Daten wird im Folgenden die Evolution von Klassifizierern mit GP für die beschleunigte Evolution von Laufmustern präsentiert.

Für die Evolution eines Klassifizierers wurde die frei erhältliche Academic-Version von DISCIPULUS[4] mit den standardmäßigen Einstellungen für Klassifikationsprobleme ohne weitere Änderungen verwendet [87]. Discipulus ist ein Programm zur Lösung von Regressions- und Klassifikationsproblemen mit Genetischer Programmierung. Es ist unter anderem interessant, weil es eine sehr schnelle Maschinensprachen-Repräsentation von GP-Individuen verwendet. In Discipulus werden die meisten Einstellungen für die Evolution (wie z.B. die Populationsgröße, Operatorwahrscheinlichkeiten, Programmlängen etc.) automatisch gesetzt. Da das System sehr schnell ist, werden viele Läufe für ein Problem durchgeführt, wobei die Parameter des Systems automatisch variiert werden.

Für die Verwendung in Discipulus wurden die Daten des Experimentes in einem Vorverarbeitungsschritt nochmal überarbeitet. So wurden die Resultate der Turniere aufgrund von Gleichung (7.16) jeweils doppelt aufgeführt, wobei die Reihenfolge von i_1 und i_2 vertauscht wurde. Die Fitness der evolvierten Klassifizierer für die Klassen c_0, c_1 und c_2 wird nach (7.18) durch die hit rate definiert, d.h. durch den Prozentsatz der korrekt klassifizierten Datensätze.

Für jede Klasse $\{c_0, c_1, c_2\}$ wurde ein eigener Klassifizierer evolviert, so dass das Gesamtergebnis, der Klassifizierer C' nach (10.2), die verrechnete Ausgabe aller drei Einzelklassifikationen ist, und zwar diejenige Klasse mit dem höchsten Konfidenzniveau k_V:

$$C'(V') \quad = \quad < c, k_V > \text{ mit } c = c_i \mid k_V = max\{k_V(c_i) \mid i = 0,1,2\}. \tag{10.2}$$

[4] DISCIPULUS ist ein Produkt der Firma AIM Learning Inc.

		Alle Daten	letzte 5 Generationen	letzte 2 Generationen
(10.2)	über Konf.-Niveau	63,8%	70,0%	60,0%
	Varianz	0.02	0.02	0.02
	Fehlklassifikationen	51,7%	39,0%	32,7%
	Varianz	0.02	0.01	0.03
(10.3)	über Konf.-Niveau	90,3%	69,0%	78,5%
	Varianz	0.01	0.03	0.03
	Fehlklassifikationen	54,1%	35,0%	34,7%
	Varianz	0.02	0.02	0.02

Tabelle 10.2: Vergleich der Klassifikationsleistung beider Ansätze. Konfidenzniveau 65%. Deutlich zu erkennen ist die schlechte Klassifikationsleistung wenn alle Daten verwendet werden. Die Varianten, die nur Daten der jüngeren Vergangenheit verwenden, zeigen niedrigere durchschnittliche Fehlerraten.

Alternativ wurde für das Ergebnis der Gesamtklassifikation in Betracht gezogen, dass nicht nur positive Klassifikationen mit hohem Konfidenzniveau, sondern auch negative Klassifikationen brauchbare Informationen darstellen. Ein Klassifizierer, der etwa für die Klassifikation in Klasse c_0 ein Konfidenzniveau von k_V angibt, zeigt gleichzeitig mit $1-k_V$ an, dass der Vergleich entweder zur Klasse c_1 oder Klasse c_2 gehört. Das Ergebnis der Gesamtklassifikation ist nun nach Gleichung (10.3) definiert als:

$$C'(V') = < c, k'_V > \text{ mit } c = c_i \mid k'_V = max\left\{k'_V(c_i) \mid i = 0,1,2\right\} \tag{10.3}$$

$$k'_V(c_i) = max\left\{ k_V(c_i), \frac{\sum\limits_{c_j \neq c_i} (1 - k_V(c_j))}{2} \mid i, j = 0,1,2 \right\}. \tag{10.4}$$

Hierbei wird berücksichtigt, dass die Wahrscheinlichkeit zu einer bestimmten Klasse zu gehören einerseits vom Konfidenzniveau des entsprechenden Einzelklassifizierers und andererseits vom Durchschnitt der komplementären Konfidenzniveaus der beiden anderen Einzelklassifizierer abhängt. Da nach (10.4) die Bedingung $0.5 < k'_V(c_i) < 1.0$ gilt, muss das gegebene Konfidenzniveau k (siehe Abschnitt 7.4.3) durch

$$k' = 0.5 + 0.5 \cdot k \tag{10.5}$$

angepasst werden, um nach (10.3) das Gesamtklassifikationsergebnis bestimmen zu können.

10.4 Einfluss der Trainingsdaten

Um das GP-System mit Trainingsdaten zu versorgen, wurden zwei verschiedene Strategien untersucht. Hierbei wurden einerseits alle in der Vergangenheit aufgezeichneten Turniere für die Klassifikation der aktuellen Generation verwendet, andererseits nur die Daten der letzten Generationen für die Evolution des Klassifizierers hinzugezogen (siehe Abbildung 10.3). Die Daten wurden dabei zu je 50% auf die Trainings- und Validierungsmenge aufgeteilt. Beide Varianten wurden zur Generierung der Trainings- und Validierungsmenge für die Evolution von Klassifizierern mit Discipulus verwendet und anschließend analysiert. Der evolvierte Klassifizierer wurde dann auf die Turniere der jeweils nächsten Generation angewendet. Für die Experimente mit reduzierter Trainingsmenge wurde einmal ein Zeitfenster von 5 Generationen, beim zweiten Lauf ein Zeitfenster von 2 Generationen gewählt, wobei die Daten der vorletzten Generation die Validierungsmenge und die übrigen Daten die Trainingsmenge bilden. In Tabelle 10.2 ist das Ergebnis des Vergleiches dargestellt.

Alle im Verlaufe der Evolution entstehenden Daten für die Evolution eines Klassifizierers zu verwenden wirkt sich ungünstig aus, wie die Performance der entsprechenden Klassifizierer zeigt. Deutlich bessere

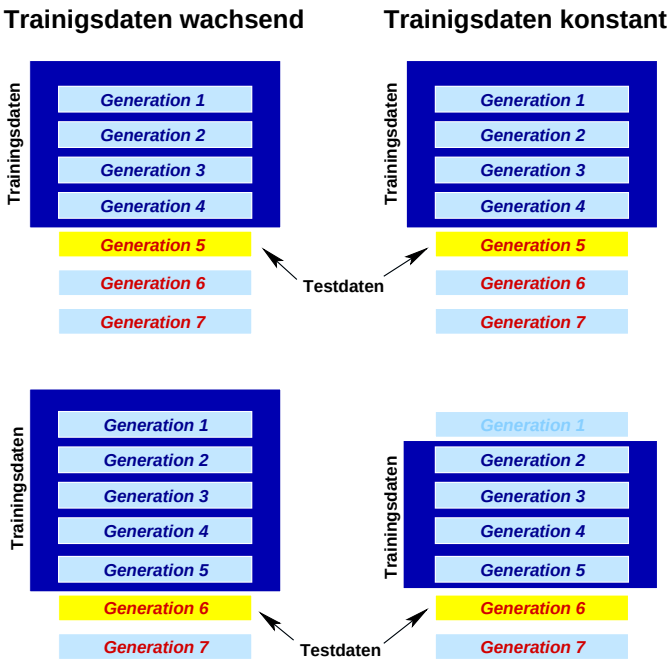


Abbildung 10.3: Zwei Ansätze zur Generierung von Trainingsdaten. **Links:** Alle Datensätze werden verwendet. **Rechts:** Datensätze, die älter als eine bestimmte Grenze sind (hier vier Generationen), werden nicht mehr für die Evolution verwendet.

Resultate (vor allem im Bereich der Fehlklassifikationen) erreichen die Klassifizierer, die nur begrenzt Datensätze aus der Vergangenheit für die Evolution erhalten.

Keine der beiden alternativen Ansätze zur Bestimmung des Gesamtklassifikationsergebnisses (nach Gleichung (10.2) und (10.3)) ist der jeweils anderen überlegen, wie die im Wesentlichen gleichen Fehlerraten andeuten. Wird das Ergebnis nach Gleichung (10.3) gebildet, so ist allerdings im Mittel die Anzahl der eingesparten Auswertungen (also der Anteil der Klassifikationen über dem geforderten Mindestkonfidenzniveau) höher, zumindest bei den hier durchgeführten drei Experimenten.

Die Tatsache, dass bessere Ergebnisse erzielt werden, wenn nur Datensätze der jüngeren Vergangenheit verwendet werden, um Klassifizierer zu evolvieren, deutet darauf hin, dass sich die Entscheidungskriterien für den Klassifizierer im Verlaufe der Evolution verschieben. Müssen frühere Datensätze berücksichtigt werden, ist keine präzise Klassifikation der aktuellen Generation mehr möglich, denn die Fitness des Klassifizierer beruht auf einer gleichmäßigen Leistung über die gesamte Trainingsmenge. So wird durch die Auswahl der Trainingsdaten Wert auf für die aktuelle Population irrelevante Turniervergleiche gelegt, die die Performance des Klassifizierers deutlich schwächen.

Eine Analyse der Klassifizierer zeigt, dass die Auswahl der Trainingsdaten deutlichen Einfluss auf die Wichtigkeit einzelner Parameter hat (siehe Abbildung 10.4). Die Parameter der Klassifikation sind hierbei die zwei reellwertigen 16-Tupel der Individuen des Vergleichs. Für die Untersuchung der Wichtigkeit wurden die Klassifikationen durchgeführt, wobei jeweils ein Parameter nicht für die Klassifikation verwendet werden konnte. Der Einfluss des Parameters kann dann anhand der Änderung der Fitness des Klassifizierers gegenüber der Fitness des unveränderten Klassifizierers bewertet werden. In Abbildung 10.4 wird die Verteilung der Einflüsse der Parameter von beiden Individuen am Beispiel der Generation 22 dargestellt.

Es zeigt sich dabei, dass die Einzelklassifizierer, die nur jüngere Datensätze verwenden, auf unterschiedlichen Parametern arbeiten, bzw. dass die Parameter unterschiedlichen Einfluss auf die Performance der Klassifizierer besitzen (Abbildung 10.4, rechts.). Der Einfluss der Parameter ist auf unterschiedliche Art

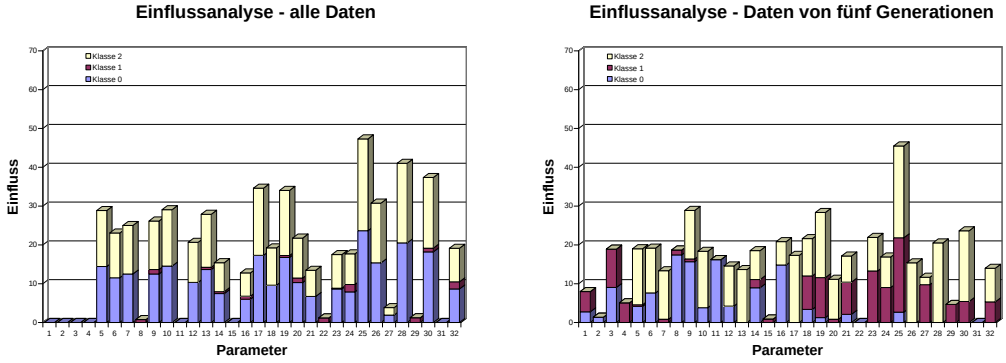


Abbildung 10.4: Darstellung der Einflussanalyse der einzelnen Parameter (Parameter 1 – 16: Individuum 1, Parameter 2 – 32 Individuum 2) für die Klassifikation am Beispiel der Generation 22. **Links:** Verteilung der Einflüsse bei Einbeziehung aller Datensätze **Rechts:** Einfluss der Parameter wenn die letzten fünf Generationen berücksichtigt werden.

und Weise auf die 32 Werte (16 Werte des ersten und 16 Werte des zweiten Individuums) verteilt, je nach Zusammensetzung der Trainingsmenge. Während der Klassifizierer der Klasse c_0 (Individuum 0 gewinnt den Vergleich) hauptsächlich auf Parametern von Individuum 0 arbeitet, arbeitet der Klassifizierer für die Klasse c_1 (Individuum 1 gewinnt den Vergleich) auf Parametern von Individuum 1. Der Klassifizierer, der für die Klassifizierung eines Unentschiedens zuständig ist (Klasse 2), verwendet allerdings alle Parameter nahezu gleichmäßig. Dies ist auffällig, da gute Klassifikationsleistungen erzielt werden (siehe Tabelle 10.2), obwohl für die Einzelklassifizierer das jeweils andere Individuum nahezu bedeutungslos ist. Die Klassifikation wird sozusagen „blind“ durchgeführt.

Im Gegensatz dazu stehen die Klassifizierer, die alle Datensätze berücksichtigen. Die Spezialisierung der Klassifizierer ist einer undifferenzierten Abhängigkeit gewichen. So sind die Einzelklassifizierer der Klassen c_0 und c_2 von der Mehrzahl der Parameter sehr stark abhängig, von anderen Parametern hingegen überhaupt nicht, wie beispielsweise der Klassifizierer der Klasse c_1, der nahezu unabhängig von allen Parametern ist.

Die Analyse der Daten der Evolution von Laufmustern für Aibo-Roboter zeigt deutlich das Potenzial der Evolution mit einem Meta-Modell. Dieser Ansatz wird im nächsten Abschnitt konsequent verfolgt, wobei hier *online* während der Evolution die Klassfizierer für die jeweils nächste Generation mit GP evolviert wird. Das Gesamtklassifikationsergebnis wird dabei nach Gleichung (10.3) bestimmt, da bei gleicher Fehlerrate mehr Auswertungen eingespart werden können als nach (10.2).

10.5 Interaktive Evolution mit einem Meta-Modell

Um die Möglichkeiten zur Einsparung von kostenintensiven Auswertungen während der Evolution zu verdeutlichen, ist der in Abschnitt 10.2 präsentierte Evolutionslauf mit identischen Parametereinstellungen (aufgeführt in den Tabellen D.4, D.2 und D.3) wiederholt worden. Die ersten fünf Generationen beider Läufe sind identisch und bilden zu Beginn der Meta-Evolution die Trainings- bzw. Validierungsmenge für die Generierung von Klassifizierern mit GP. Die in Abschnitt 10.3 verwendeten Parameter wurden für die Evolution von Klassifizierern mit Discipulus unverändert übernommen. Die Resultate der manuell ausgewerteten Turniere der letzten fünf Generationen bilden die Trainings- und Validierungsmenge im Verhältnis 75% : 25%, da aufgrund der relativ kleinen Population die Trainingsmenge bei einer Aufteilung von 50% : 50% zu klein werden würde. Die Daten der letzten fünf Generationen wurden verwandt, da bei weniger Generationen die Anzahl der Datensätze insgesamt zu klein wird.

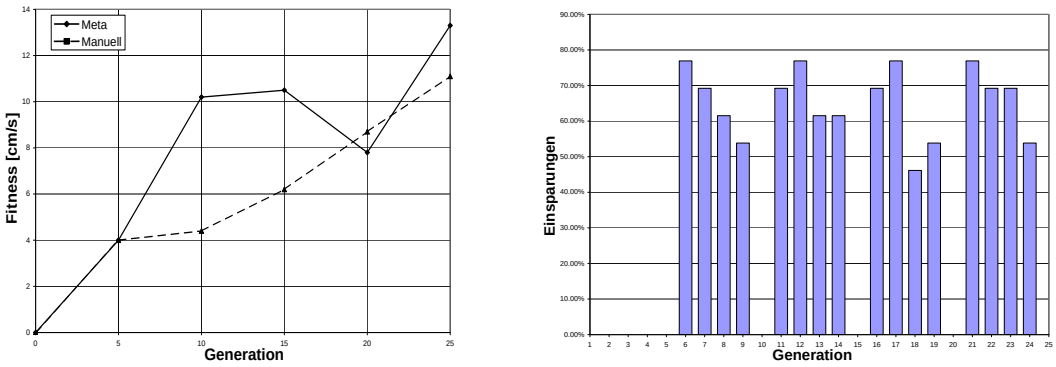


Abbildung 10.5: **Links:** Die Fitness des besten Individuums der Population bei manueller und Meta-Modell-Evolution. **Rechts:** Prozentsatz der Klassifizierungen pro Generation. Die Evolution mit Meta-Modell erreicht ähnliche Fitnesswerte mit nur ca. 53% der Auswertungen, die die manuelle Evolution benötigt. Die gesamte Population wird alle fünf Generationen auswertet um exakte Fitnesswerte zu erhalten (deshalb $p_{1,2,3,4,5,10,15,20,25} = 0$).

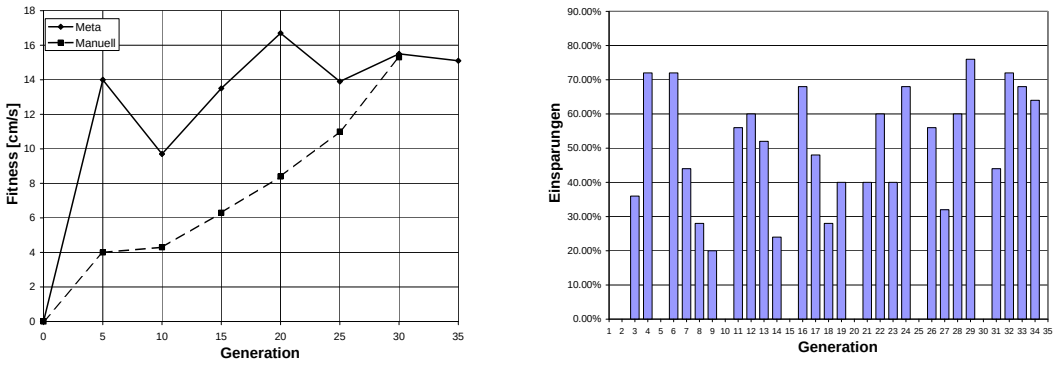


Abbildung 10.6: **Links:** Die Fitness des besten Individuums der Population bei manueller und Meta-Modell-Evolution. **Rechts:** Prozentsatz der Klassifizierungen pro Generation. Die Evolution mit Meta-Modell erreicht schnell bessere Fitnesswerte als die manuelle Evolution ohne Meta-Modell.

In Abbildung 10.5 ist der Vergleich der ersten 25 Generationen beider Experimente dargestellt. Die manuelle Evolution benötigte für 25 Generationen 650 Auswertungen mit dem realen Roboter, wobei das beste Individuum des Experimentes eine Fitness von 11 $\frac{cm}{s}$ erreicht. Die Evolution mit Klassifikation erreicht ähnliche Fitnesswerte (13 $\frac{cm}{s}$), benötigt hierfür jedoch nur 305 anstatt 650 Auswertungen, d.h. nur ca. 53% der Anzahl, die für die manuelle Evolution notwendig waren. Pro Generation konnten hierbei bis zu 80% der Auswertungen eingespart werden. Durch die Verwendung von - fehlerbehafteten - Klassifizieren kann das beste Individuum während der Evolution verlorengehen, wie der Einbruch der Fitnesskurve in Abbildung 10.5 verdeutlicht.

In einem weiteren Experiment wurden die Ergebnisse des oben dargestellten Laufes bestätigt (siehe Abbildung 10.6). Die Parameter des GA für dieses Experiment sind in Tabelle D.5 in Abschnitt D.2 aufgeführt. Das Experiment zeigt, dass auch hier die Einsparung an benötigten Auswertungen mit dem realen Roboter groß ist. Pro Generation konnten bis zu 70% der Auswertungen eingespart werden. Von den 1750 Auswertungen, die insgesamt für die Evolution von 35 Generationen benötigt werden, konnten 664 Auswertungen mit Hilfe der Klassifikation durchgeführt werden, d.h. es wurden ca. 38% aller Auswertungen eingespart.

Die Durchführung des manuellen Experimentes ohne Meta-Evolution benötigte ca. 8 Stunden. Das erste Experiment mit Meta-Modell benötigte ungefähr die doppelte Zeit, das zweite Experiment sogar mehrere Tage. Dies wurde durch die zeitaufwändige Evolution von Klassifizierern für die einzelnen Klassen (siehe 10.2) in jeder Generation verursacht. Wenn bei der Evolution von Klassifizierern nicht immer von einer rein zufällig initialisierten Startpopulation ausgegangen wird, sondern ein bereits evolvierter Klassifizierer durch Hinzufügen neuer und Entfernen alter Datenpunkte angepasst wird, kann die Zeit für die Evolution verkürzt werden.

Ziel der Meta-Evolution ist es, die Anzahl der notwendigen Auswertungen während der Evolution so weit wie möglich zu reduzieren. Mit den beiden Experimenten konnte gezeigt werden, dass Einsparungen von bis zu 50% erreicht werden können, ohne Einbußen bei der Qualität hinnehmen zu müssen. Auf diese Weise können viele kostenverursachende Auswertungen vermieden werden, was die Lebensdauer der verwendeten Roboter deutlich erhöht. Es hat sich jedoch gezeigt, dass es nicht ausreicht, die Geschwindigkeit der Bewegungen als alleiniges Kriterium für die Selektion zu verwenden. Die evolvierten Laufmuster sind schnell, jedoch nicht ohne Einschränkungen geeignet, im Roboterfußball eingesetzt zu werden, da sie nicht die geforderte Robustheit aufweisen, wobei Robustheit im Sinne des Roboterfußballs eine versatzfreie Geradeausbewegung und außerdem eine einfache Parametrisierbarkeit für variable Geschwindigkeiten und für Kurvenlauf bedeutet. Um auch diesen Ansprüchen zu genügen, müssen diese Kriterien zusätzlich zur Geschwindigkeit in das Urteil des Experimentators einfließen. Dies kann zum Beispiel dadurch geschehen, dass die Kriterien für einen Vergleich zweier Individuuen wechseln und durch geschickte Kombination der Häufigkeiten so eine multikriterielle Optimierung ermöglicht wird.

Zusammenfassung und Ausblick

Im Mittelpunkt dieser Arbeit steht die automatische Generierung von Kontrollalgorithmen für beliebige Laufroboter. Hierfür wird die Genetische Programmierung verwendet, eine besondere Ausprägung der Evolutionären Algorithmen, bei der die Problemvariablen eines Optimierungsproblems in Form von interpretierbaren Rechenvorschriften repräsentiert werden, d.h. als Algorithmus oder als Computerprogramm. Ein Kontrollprogramm für einen Laufroboter zu generieren wird hier als ein Optimierungsproblem interpretiert, bei dem die Güte der Lösungen (d.h. die Qualität der Laufprogramme) berechnet wird, indem die entsprechenden Programme einen Roboter steuern. Über den Grad der Erfüllung bestimmter Kriterien, die zusammengenommen die Fitnessfunktion bilden, wird die Güte bestimmt. Durch wiederholte Anwendung der Variationsoperatoren und anschließende Auslese der besten Laufprogramme wird die Qualität der Programme sukzessiv gesteigert.

Laufrobotersteuerungen mit Hilfe der Genetischen Programmierung zu evolvieren bietet viele Vorteile, die nicht auf den ersten Blick ersichtlich sind. Hier ist vor allem die geringe Menge an spezifischem Problemwissen (in diesem Fall über den Roboter, genauer gesagt über die Anzahl der Beine, der Freiheitsgrade, der herrschenden Symmetrien etc.) zu nennen, das für die Anwendung eines evolutionären Optimierverfahrens vorausgesetzt wird. Ein Programm zur Steuerung der Beine eines Laufroboters manuell zu implementieren bedeutet hingegen, dass der Entwickler über fundierte Kenntnisse beispielsweise auf dem Gebiet der Mechanik, der Regelungstechnik und der Mathematik der Differentialgleichungen verfügen muss und darüber hinaus sich auch auf den Gebieten der numerischen Mathematik und der Computersprachen auskennt. Es ist leicht zu erkennen, dass die Gruppe der potenziell hierzu fähigen Personen relativ klein ist.

Demgegenüber steht die Evolution von Programmen, die im einfachsten Fall auf einer Halbordnung der Laufmuster beruht, d.h. auf einer einfachen *Programm-x-ist-besser-als-Programm-y*-Relation, nach der die Population geordnet werden kann. Auf dieser Halbordnung baut der Selektionsmechanismus der Evolutionären Algorithmen auf, der für den Fortschritt der Population in Richtung Optimum verantwortlich ist. Die Kriterien, anhand derer diese Halbordnung aufgebaut wird, können mathematisch formulierte Funktionen sein, es ist allerdings auch möglich und in manchen Fällen sogar ausdrücklich erwünscht, subjektive Eindrücke von menschlichen Betrachtern für die Vergleiche von Laufprogrammen heranzuziehen.

Die Evolution von Laufrobotersteuerungen benötigt - wie jedes evolutionäre Experiment - viele Auswertungen bis zum Erreichen einer bestimmten Güte. Dies ist bedingt durch die populationsbasierte Suche, die durch die Eigenschaften der Variations- und Selektionsmechanismen auch Verschlechterungen einzelner Programme während der Suche zulässt. Die Anzahl der Auswertungen, aber auch die Dauer jeder einzelnen Auswertung, ist bestimmend für die Laufzeit eines evolutionären Experimentes. Wenn über viele Generationen hinweg evolviert wird, muss mehrere hundert oder tausend Mal ein Laufprogramm einen Roboter über eine bestimmte Zeit steuern. Werden reale Roboter hierfür verwendet, bedeutet dies einen hohen Verschleiß, der mit entsprechenden Kosten verbunden ist. Wird der Roboter simuliert, ist ein entsprechender Aufwand für die Erstellung von Computermodellen nötig und die Dynamiksimulation des Roboters ist, abhängig von der Komplexität des Roboters, ebenfalls zeitaufwändig. Die Reduktion der Anzahl von Auswertungen während der Evolution einerseits und die Verkürzung der Dauer einer Auswertung andererseits ist Gegenstand der Untersuchungen dieser Arbeit. In Kapitel 7 wird gezeigt, dass mit Hilfe des vorzeiti-

gen Abbruchs von Auswertungen die Geschwindigkeit des Algorithmus bedeutend gesteigert werden kann. Wertvolle Rechenzeit wird nicht mehr für die Bewertung von wahrscheinlich minderwertigen Laufprogrammen verbraucht, sondern kann für potenziell gute Laufprogramme verwendet werden. Die Qualität eines Laufprogramms wird während der Ausführung kontinuierlich überprüft. Unterschreitet die Güte einen vorher definierten Wert, so wird die Ausführung des Laufprogramms beendet und es wird mit einer schlechten Fitness bewertet.

Nicht nur die Dauer sondern auch die Anzahl der Auswertungen kann reduziert werden, wie das Beispiel eines Meta-Modells der Fitnessfunktion demonstriert. Die Vergleiche, die für das Erstellen einer Halbordnung der Population notwendig sind und die auf den Fitnesswerten der Laufprogramme basieren, werden durch Klassifikationen ersetzt. Die Grundlage dieser Klassifikationen ist allein die Struktur der Individuen, so dass die zeitaufwändige Bestimmung der Fitnesswerte eines Laufprogrammes entfallen kann. Die Entscheidungen der Klassifizierer, die ebenfalls in einem evolutionären Prozess erzeugt werden, basieren auf Resultaten früherer Auswertungen, die als Trainingsmenge verwendet werden. Der Klassifizierer wird an die sich ändernde Struktur der Population angepasst, indem die Trainingsmenge durch Auswertungen aktualisiert wird. Evolutionsläufe, die diese Methode der Modellierung der Fitnessfunktion verwenden, haben bei deutlich geringerer Anzahl von Auswertungen Resultate mit gleichbleibender Qualität gezeigt.

Es spielt für die Evolution von Laufprogrammen keine Rolle, ob der verwendete Roboter real existiert oder nur als Computermodell vorhanden ist. Wichtig für eine erfolgreiche Evolution ist nur, dass die Größen, die zur Bestimmung der Fitness der auf den Robotern ausgeführten Laufprogramme verwendet werden, über geeignete Messverfahren aufgenommen werden können. Hierbei sind bestimmte Werte wie zum Beispiel die während der Ausführung des Laufprogramms vom Roboter zurückgelegte Wegstrecke in Simulation und Realität bestimmbar, andere Größen hingegen, wie beispielsweise die subjektive Bewertung der Laufbewegungen, sind nur bei der Evolution mit realen Robotern möglich. In Kapitel 9 wird gezeigt, dass die Ergebnisse der Simulation durchaus auch auf die Realität übertragbar sind. Voraussetzung hierfür ist ein existierendes Computermodell eines realen Roboters und eine gemeinsame Schnittstelle über die das Laufprogramm Sensordaten des Roboters erfassen und die Aktoren steuern kann. Verhält sich der reale Roboter nicht so, wie in der Simulation vorhergesagt, so kann durch entsprechende Anpassungen des Modells der Aufwand für die notwendige Kalibrierung der Laufprogramme an den realen Roboter reduziert werden. Diese hybride Evolution, d.h. die Evolution sowohl in der Simulation als auch mit einem realen Roboter, hat sich als sehr gut geeignet erwiesen, Laufprogramme für den humanoiden Miniaturroboter ZORC zu evolvieren.

Ein vielversprechender Ansatz für zukünftige Arbeiten auf diesem Gebiet ist es, die Übertragbarkeit von Laufprogrammen aus der Simulation auf reale Roboter ebenfalls als ein Fitnesskriterium zu betrachten. In einem - dann aufwändigeren - kombinierten Prozess, bei dem die Fitness sowohl durch Kriterien zur Bewertung von Laufmustern als auch zur Bewertung der Übertragbarkeit definiert wird, könnte die manuelle Anpassung des Modelles an den realen Roboter ersetzt werden durch einen automatisierten Prozess.

Die Struktur der Roboter, auf denen die Programme ausgeführt werden, hat auf ambivalente Weise Einfluss auf die Evolution: Sie spielt auf der einen Seite eine besondere Rolle, da ein im Sinne der Fitnessfunktion gutes oder schlechtes Programm diese Qualität nur für einen bestimmten Roboter aufweist. Auf der anderen Seite ist die Roboterarchitektur von untergeordneter Bedeutung, denn die Kriterien für gutes oder schlechtes Laufen werden unabhängig von der jeweiligen Ausprägung eines Roboters definiert. Wird ein Laufprogramm für einen Roboter evolviert, so lautet die Zielsetzung des evolutionären Prozesses: (i) Die Programme sollen möglichst gut im Sinne der Kriterien der Fitnessfunktion werden, und (ii) sie sollen diese Güte mit dem bereitgestellten Roboter erreichen. Diese Vorgehensweise erlaubt es, den Roboter als einen Parameter der Evolution zu betrachten, der beliebig variierbar ist. Die in Teil III der Arbeit aufgeführten Experimente mit einer Vielzahl unterschiedlicher Roboter (Computermodelle in Kapitel 8, reale Roboter in Kapitel 9) zeigen deutlich, dass mit identischen Anfangsbedingungen des GP-Systems erfolgreich Laufprogramme für eine Vielzahl unterschiedlicher Roboter evolvierbar sind. Es spielt hierbei keine Rolle, ob die Roboter sechs, vier, zwei oder sogar gar keine Beine besitzen.

Eine der wichtigsten Fragen dieser Arbeit ist die nach den essenziellen Kriterien, die ein erfolgreiches Programm erfüllen muss, um überhaupt ein Laufmuster zu erzeugen. Die Komplexität des Laufvorganges in der Biologie und die ausgefeilte mechanische Struktur existierender Laufroboter lassen die Vermutung zu, dass nur durch einen ebenso komplexen Kriterienkatalog, der die Fitnessfunktion der Evolution von Laufprogrammen bildet, eine kontrollierte, laufähnliche Fortbewegung erzielt werden kann. In den aufgeführten Experimenten mit simulierten und realen Robotern (Kapitel 8 und 9) wird dagegen deutlich, dass schon durch wenige und sehr primitive Kriterien erfolgreich Laufprogramme evolviert werden können. Das allereinfachste Kriterium, dass zur Evolution von Laufmustern geführt hat, ist die direkte Korrelation zwischen Bewegungsgeschwindigkeit des Roboters und der Fitness der Programme, wie sie z.B. beim linearen Roboter in Kapitel 8 angewendet wird. Dieses Kriterium hat bei allen untersuchten Roboterarchitekturen zu einer kontrollierten Bewegung geführt. Die resultierenden Bewegungsmuster stimmen bei den Robotern mit mehreren Beinen jedoch nicht mit der intuitiven Vorstellung von natürlichem Gehen überein. Durch Hinzufügen weiterer Kriterien (wie beispielsweise der positiven Bewertung einer aufrechten Haltung) ist es möglich, die evolvierten Laufmuster zu verändern. Die bei laufenden Lebewesen vorherrschende Auswahl des Gangmusters nach Gesichtspunkten der Energieffizienz ist ebenfalls ein möglicher Ansatzpunkt zukünftiger Experimente.

Gerade bei simulierten Robotern hat die visuelle Überprüfung der evolvierten Laufrobotersteuerungen häufig dazu geführt, dass die Kriterien der Fitnessfunktion bzw. deren Gewichtung überarbeitet wurden. Dieser Prozess, also die Variation der Fitnessfunktion oder bestimmter Teile der Fitnessfunktion aufgrund von visuellen Eindrücken der evolvierten Bewegungsmuster, kann mit Recht selbst ein evolutionärer Prozess genannt werden, bei dem die Kriterien für die Evolution von Laufmustern und ihre Gewichtung aufgestellt werden.

Wenn die Komplexität einer natürlichen Bewegung (wobei „natürlich“ an dieser Stelle ein subjektiver Eindruck der Fortbewegungsmuster künstlicher Apparate ist) nur durch einen Katalog unterschiedlich gewichteter Kriterien zu beschreiben ist, so wird dadurch eine potenzielle Methode für zukünftige Untersuchungen aufgezeigt: Methoden der multikriteriellen Optimierung, angewendet auf die Evolution von Laufrobotersteuerungen, stellen eine Möglichkeit dar, komplexere Fortbewegungsmuster zu evolvieren. Ein erster Ansatz für die Umsetzung dieser Methode kann die Anwendung abwechselnder Entscheidungskriterien bei den Vergleichen der Turnierselektion sein. Eine komplexe Fitnessfunktion kann so durch unterschiedlich häufige Verwendung der verschiedenen Kriterien realisiert werden. Die optimale Zusammensetzung der Fitnessfunktion kann ebenfalls durch einen weiteren evolutionären Prozess bestimmt werden, wie es in Ansätzen bereits in Kapitel 9.2.3 praktiziert wurde.

Ob durch immer komplexere Fitnessfunktionen oder über eine entsprechende Zusammensetzung und ausgewogene Gewichtung des Kriterienkataloges auch immer komplexere Laufmuster generiert werden können ist hierbei ein ungelöstes Problem. Der in dieser Arbeit verfolgte Ansatz der „ganzheitlichen“ Evolution von Laufroboterprogrammen, d.h. der Evolution des kompletten Programms zur koordinierten Bewegung der Beine eines Laufroboters in einem Evolutionslauf, bedeutet zwangsläufig einen Verzicht auf vorhandenes Problemwissen. Einerseits ist dies beabsichtigt, um die Evolution möglichst wenig einzuschränken und einem breiteren Anwenderkreis die Benutzung des Systems zu erlauben, auf der anderen Seite hat dies zur Folge, dass möglicherweise wertvolle Information nicht verwendet wird. Das Wissen über die meist modulare Struktur von Laufrobotern, deren Beine in der Regel kinematisch identisch sind, oder über die symmetrische Struktur von stabilen Laufbewegungen kann zu einer beschleunigten Evolution oder zu verbesserten Laufbewegungen führen, wenn es in einer geeigneten Weise in den Evolutionären Algorithmus integriert werden kann.

Durch eine Strukturierung des Problems, beispielsweise eine Unterteilung des Laufens in die Bewegungen einzelner Beine, kann auch die Evolution von Laufprogrammen strukturiert werden. Programme, die ein einzelnes Bein kontrolliert bewegen, können anschließend in einer höheren Ebene, die für die Laufmustergenerierung verantwortlich ist, als parametrisierbare Prozeduren verwendet werden. Die Programmstrukturen können dann in getrennten Evolutionen erzeugt werden. Auf diese Weise könnte auch die Steuerung

des Roboters auf höherer Ebene realisiert werden, also die kontrollierte Bewegung des Roboters im Raum. Hierfür notwendig sind parametrisierbare Laufmuster, die es dem Roboter erlauben, beliebige Bahnen im Raum zu laufen.

Diese Methode ist natürlich nur anwendbar, wenn eine Strukturierung der Steuerungsebenen möglich ist. Besitzt ein Roboter keine Beine, wie beispielsweise die zu Beginn von Kapitel 8 dargestellten Roboter, oder sind die Beine des Roboters kinematisch voneinander verschieden, so lässt sich eine solche Strukturierung nicht mehr einfach angeben. Für die Integration von Problemwissen in Evolutionäre Algorithmen existiert noch dazu kein einheitliches Vorgehensmodell, so dass die in dieser Arbeit verwendete Methode der ganzheitlichen Evolution kompletter Laufrobotersteuerungen den weitesten Anwendungsbereich aufweist.

Jens Ziegler, Februar 2003

Über den Autor

Von 1991 bis 1997 Studium der Angewandten Informatik mit dem Schwerpunkt Ingenieurwissenschaften (Maschinenbau) an der Universität Dortmund, Abschluss Diplom-Informatiker. Seit 1997 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Dr. Wolfgang Banzhaf am Lehrstuhl für Systemanalyse der Universität Dortmund. Schwerpunkte der wissenschaftlichen Tätigkeit lagen auf den Gebieten der Künstlichen Chemie, der Genetischen Programmierung und der Robotik, speziell den autonomen Laufrobotern, sowie auf dem Gebiet des Roboterfußballs mit Laufrobotern.

Publikationen

Artikel in Zeitschriften/Buchkapitel

[1] P. Dittrich, J. Ziegler, and W. Banzhaf. Artificial Chemistries - A Review. *Artificial Life*, 7:225 – 275, 2001.

[2] A. Skusa, W. Banzhaf, J. Busch, P. Dittrich, and J. Ziegler. Künstliche Chemie. *Künstliche Intelligenz*, pages 12 – 19, 2000.

[3] J. Ziegler and W. Banzhaf. Evolving Control Metabolisms for a Robot. *Artificial Life*, 7:171 – 190, 2001.

[4] J. Ziegler, P. Dittrich, and W. Banzhaf. Towards a metabolic robot control system. In Mike Holcombe and Ray Paton, editors, *Information Processing in Cells and Tissues*, pages 305–318, New York, 1998. Plenum Press.

Konferenzbeiträge

[5] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, and Wolfgang Banzhaf. Automatic generation of control programs for walking robots using genetic programming. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Proceedings of the 5th European Conference on Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 258–268. Springer, New York, 2002.

[6] I. Dahm and J. Ziegler. Adaptive methods to improve self-localization in robot soccer. In *Proceedings of the 6th international RoboCup Symposium*, Lecture Notes in Artificial Intelligence, pages 276–283. Springer New York, Heidelberg, Berlin, 2002.

[7] I. Dahm and J. Ziegler. Using artificial neural networks to construct a meta-model for the evolution of gait patterns of four-legged walking robots. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the 5th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 825–832. Professional Engineering Publishing, 2002.

[8] P. Dittrich, J. Ziegler, and W. Banzhaf. Mesoscopic analysis of self-evolution in an artificial chemistry. In C. Adami, R. Belew, H. Kitano, and C. Taylor, editors, *Artificial Life VI*, pages 95–103, Cambridge, MA, 1998. MIT Press.

[9] J. Ziegler and W. Banzhaf. Evolution of robot leg movements in a physical simulation. In K. Berns and R. Dillmann, editors, *Proceedings of the Fourth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 395–402, Bury St Edmunds, London, UK, 2001. Professional Engineering Publishing.

[10] J. Ziegler and W. Banzhaf. Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Proceedings of the 6th European Conference on Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 264–275. Springer, New York, 2003.

[11] J. Ziegler and W. Banzhaf. You can judge a book by its cover - evolution of gait controllers based on program code analysis. In *Proceedings of the Sixth International Conference on Climbing and Walking Robots (CLAWAR)*, Bury St Edmunds, London, UK, 2003. Professional Engineering Publishing. in print.

[12] J. Ziegler, J. Barnholt, J. Busch, and W. Banzhaf. Automatic evolution of control programs for a small humanoid walking robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the 5th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 109–116. Professional Engineering Publishing, 2002.

[13] J. Ziegler, K. Wolff, P. Nordin, and W. Banzhaf. Constructing a small humanoid walking robot as a platform for the genetic evolution of walking. In U. Rückert, J. Sitte, and U. Witkowski, editors, *Proceedings of the 1st International Workshop on Autonomous Minirobots for Reserch and Edutainmant (AMiRe)*, pages 51–59. HNI Verlagsschriftenreihe, 2001.

Sonstige Veröffentlichungen

[14] C. Aue, A. Benkacem, M. Gregorius, A. Ross, S. R. Abdallah, D. Sawitzki, V. Strunk, H. Türk, M. C. Varcol, J. Busch, and J. Ziegler. Simulator für GP-evolvierte Laufrobotersteuerungsprogramme - PG 368. Technical report, Department of Computer Science, University of Dortmund, Germany, 2001.

[15] P. Speroni di Fenizio, P. Dittrich, W. Banzhaf, and J. Ziegler. Towards a theory of organizations. In *Proceedings of the Fourth German Workshop on Artificial Life*, 2000.

[16] P. Dittrich, W. Banzhaf, H. Rauhe, and J. Ziegler. Macroscopic and microscopic computation in an artificial chemistry. In *Proceedings of the Second German Workshop on Artificial Life*, 1997.

[17] P. Dittrich, J. Ziegler, and W. Banzhaf. Mesoscopic analysis of an artificial chemistry with self-organizing topology. In C. Wilke, S. Altmeyer, and T. Martinetz, editors, *Proceedings of the Third German Workshop on Artificial Life*. Verlag Harry Deutsch, 1998.

[18] J. Ziegler. On the influence of adaptive operator probabilities in genetic programming. Technical report, Technical Report Sys-03/02, Chair of Systems Analysis, University of Dortmund, 2002. ISSN 0941-4568.

[19] J. Ziegler and W. Banzhaf. Evolving a ”nose” for a robot. In *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*. Morgan Kaufmann, 2000.

Literaturverzeichnis

[20] A. Agapie. Theoretical analysis of mutation-adaptive evolutionary algorithms. *Evolutionary Computation*, 9(2):127–146, 2001.

[21] D. Andre and J. R. Koza. A parallel implementation of genetic programming that achieves superlinear performance. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume III, pages 1163–1174. CSREA Press, 1996.

[22] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In M. Palaniswami, Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, Piscataway, NJ, 1995.

[23] P. J. Angeline. Two self-adaptive crossover operators for genetic programming. In P. J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 5, pages 89–110. MIT Press, Cambridge, MA, USA, 1996.

[24] P. J. Angeline and J. B. Pollack. Coevolving high-level representations. In C. G. Langton, editor, *Proceedings of the 3rd Artificial Life Workshop*, pages 55–71. Addison-Wesley, Reading, MA, 1994, 1994.

[25] T. Bäck. Optimal mutation rates in genetic search. In S. Forrest, editor, *Proceedings of the Fifth Internationl Conference on Genetic Algorithms ICGA*, pages 2–8. Morgan Kaufmann, San Francisco, CA, 1993.

[26] T. Bäck. Self-adaptation in genetic algorithms. In F. J. Varela and P. Bourgine, editors, *Towards a Pratice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press, Cambridge, MA, 1992.

[27] T. Bäck. An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundamenta Informaticae*, 34:1–15, 1998.

[28] T. Bäck, A. E. Eiben, and N. A. L. van der Vaart. An empirical study on GAs "without parameters". In H.-P. Schwefel M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, pages 315–324. Springer, Paris, 2000.

[29] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):1–15, 1998.

[30] W. Banzhaf, P. Nordin, R Keller, and F. Francone. *Genetic Programming — An Introduction*. dpunkt/Morgan Kaufmann, Heidelberg/San Francisco, 1998.

[31] D. Baraff. *Physically Based Modeling: Principles and Practice*. Pixar Animation Studios, Richmond, CA. WWW Document.

[32] D. P. Barnes and J. B. Wardle. Robust gait generation for hexapodal robot locomotion. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, pages 382–383. ACM Press, 1999.

[33] A. Bastian. Adaptive genetic programming for system identification. In H.-J. Zimmermann, editor, *Eufit '97 – 5th European Congress on Intelligent Techniques and Soft Computing*, pages 759–763. ELITE – European Laboratory for Intelligent Techniques Engineering, Wissenschaftsverlag, Mainz, 1997.

[34] R. Beer and J. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1:91–112, 1992.

[35] V. Braitenberg. *Vehikel. Experimente mit kybernetischen Wesen*. Rowohlt Verlag, Hamburg, 1993.

[36] M. Brameier and W. Banzhaf. Evolving teams of predictors with genetic programming. *Genetic Programming and Evolvable Machines*, 2:381–407, 2001.

[37] M. Brameier, W. Kantschik, P. Dittrich, and W. Banzhaf. SYSGP, a C++ library of different GP variants. *Internal Report of SFB 531, Univ. of Dortmund, ISSN 1433-3325*, 1998.

[38] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik*. Harri Deutsch Verlag, Frankfurt/Main, 1989.

[39] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 1:253–262, 1988.

[40] R. A. Brooks. New approaches to robotics. *Science*, 253:1227 – 1232, 1991.

[41] R. A. Brooks. Coherent behavior from many adaptive processes. In D. Cliff, P. Husbands, J.A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 22–29. MIT Press, 1994.

[42] R. A. Brooks. From robot dreams to reality. *Nature*, 406:946–947, 2000.

[43] L. Bull, T. C. Fogarty, and M. Snaith. Evolution in multi-agent systems: Evolving communicating classifier systems for gait in a quadrupedal robot. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 382–388. Morgan Kaufmann, San Mateo, CA, 1995.

[44] A. Buschmann, M. Frik, M. Guddat, M. Karatas, and D. C. Losch. Modular generation and optimization of gait patterns for walking machines. In *Proc. Autonomous Walking 98: Theory and practical Realisation of Walking Machines*, pages 7–14. Fraunhofer IFF, 1998.

[45] D. Cliff, P. Husbands, and I. Harvey. Evolving visually guided robots. In H.R.J. Meyer and S. Wilson, editors, *Proceedings of the Second International Conference on the Simulation of Adaptive Behavior*, pages 374–383. MIT Press/Bradford Books, 1993.

[46] M. Colombetti and M. Dorigo. Learning to control an autonomous robot by distributed genetic algorithms. In J.-A. Meyer, H. Roitblatt, and S. Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB)*, pages 305–312. MIT Press/Bradford Books, 1992.

[47] H. Cruse. Coordination of leg movement in walking animals. In J.-A. Meyer and S.W. Wilson, editors, *From animals to animats. International Conference on Simulation of Adaptive Behavior (SAB)*, pages 105–119. MIT Press, Cambridge, MA, 1991.

[48] P. J. Darwen. Black magic: Interdependence prevents principled parameter setting, self-adapting costs too much computation. In S. Halloy and T. Williams, editors, *Applied Complexity: From Neural Nets to Managed Landscapes*, pages 227–237. New Zealand Institute for Food and Crop Research, Christchurch, New Zealand, 2000.

[49] A. Dasgupta and Y. Nakamura. Making feasible walking motion of humanoid robots from human motion capture data. In *International Conference on Robotics and Automation (ICRA)*, pages 1044–1049. IEEE Press, 1999.

[50] L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, San Francisco, CA, 1989.

[51] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[52] F. Delcomyn. Walking robots and the central and peripheral control of locomotion in insects. *Autonomous Robots*, 7:259–270, 1999.

[53] P. Dittrich, A. Bürgel, and W. Banzhaf. Learning to control a robot with random morphology. In P. Husbands and J.-A. Meyer, editors, *Proceedings First European Workshop on Evolutionary Robotics*, pages 165–178. Springer, Berlin, 1998.

[54] M. Dorigo and U. Schnepf. Genetics-based machine learning and behavior based robotics: A new synthesis. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(1):141–154, 1993.

[55] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robots*. Cambridge University Press, Cambridge, UK, 2000.

[56] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in Evolutionary Algorithms. *IEEE-Evolutionary Computation*, 3(2):124, 1999.

[57] M. Emmerich, A. Giotis, M. Özdenir, T. Bäck, and K. Giannakoglou. Metamodel-assisted evolution strategies. In *Parallel Problem Solving from Nature*. Springer, Berlin, 2002.

[58] K. Endo, T. Maeno, and H. Kitano. Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation - from simulation to the real robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 783–790. Professional Engineering Publishing, 2002.

[59] L. J. Eshelman, K. E. Mathias, and J. D. Schaffer. Crossover operator biases: Exploiting the population distribution. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 354–361. Morgan Kaufmann, San Francisco, 1997.

[60] K. S. Espenschied, Roger D. Quinn, Hillel J. Chiel, and Randall D. Beer. Leg coordination mechanisms in the stick insect applied to hexapod robot locomotion. *Adaptive Behavior*, 1(4):455–468, 1993.

[61] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston, MA, 1987.

[62] T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)*, pages 104–109. Morgan Kaufmann, San Francisco, CA, 1989.

[63] D. Fogel. *Evolutionary Computation*. IEEE Press, NY, 1995.

[64] M. Frik, M. Guddat, M. Karatas, and D. C. Losch. A novel approach to autonomous control of walking machines. In *Proceedings of the 2nd International Conference on Climbing and Walking Robots (CLAWAR)*, pages 333–342. Professional Engineering Publishing Limited, Bury St. Edmunds, 1999.

[65] A. Fujii and A. Ishiguro. Evolving a CPG controller for a biped robot with neuromodulation. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 17–24. Professional Engineering Publishing, 2002.

[66] Y. Gawdiak, J. M. Bradshaw, B. Williams, and H. Thomas. R2d2 in a softball: The personal satellite assistant. In H. Lieberman, editor, *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI)*, pages 125–128. ACM Press, New York, 2000.

[67] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, 1993.

[68] M. Giacobini, M. Tomassini, and L. Vanneschi. Limiting the number fitness cases in genetic programming using statistics. In J.J. Merelo et al., editor, *Proccedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN)*, Lecture Notes in Computer Science, LNCS 2439, pages 371–380. Springer Verlag, 2002.

[69] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, reading, Mass., 1989.

[70] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–128, 1986.

[71] J. J. Grefenstette and A. C. Schultz. An evolutionary approach to learning in robots. In *Machine Learning Workshop on Robot Learning*, New Brunswick, NJ, 1994.

[72] L. Gritz and J. K. Hahn. Genetic programming evolution of controllers for 3-D character animation. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 139–146. Morgan Kaufmann, 1997.

[73] F. Gruau. Cellular encoding for interactive evolutionary robotics. In R.A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Conference on Artificial Life*, pages 368–377. MIT Press/Bradford Books, 1994.

[74] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 258–265. Morgan Kaufmann, San Francisco, CA, 1999, 13-17.

[75] A. T. Hayes and P. Dormiani-Tabatabaei. Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3900–3905. IEEE, 2002.

[76] F. Herrera and M. Lozano. Adaptation of genetic algorithms parameters based on fuzzy logic. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 95–125. Physica Verlag (Springer), Heidelberg, New York, 1996.

[77] R. Hinterding, Z. Michalewicz, and A. E. Eiben. Adaptation in evolutionary computation: A survey. In *Proceedings of The Conference on Evolutionary Computation, World Congress on Computational Intelligence*, pages 65–69. IEEE Press, 1997.

[78] R. Hinterding, Z. Michalewicz, and T. C. Peachey. Self–adaptive genetic algorithm for numeric functions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 420–429. Springer, Berlin, 1996.

[79] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[80] J. H. Holland. Genetic algorithms and classifier systems: Foundations and future directions. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Prceedings of the Second International Conference on Genetic Algorithms*, pages 82–89. Lawrence Erlbaum Associates, 1987.

[81] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA., 1979.

[82] G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. Autonomous evolution of gaits with the sony quadruped robots. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1297–1304. Morgan Kaufmann, 1999.

[83] C. R. Houck and M. G. Kay. Adapting the order and frequency of genetic operators through coevolving populations. Technical Report NCSU-IE TR 96-03, North Carolina State University, 1996.

[84] P. Husbands and I. Harvey. Evolution versus design: Controlling autonomous robots. In *Integrating Perception, Planning and Action: Proceedings of the Third Annual Conference on Artificial Intelligence, Simulation and Planning*, pages 139–146. IEEE Press, 1992.

[85] P. Husbands, I. Harvey, D. Cliff, and G. Miller. Artificial evolution: A new path for artificial intelligence? *Brain and Cognition*, 34:130–159, 1997.

[86] W. Ilg, K. Berns, R. Dillmann, M.S. Fischer, H. Witte, J. Biltzinger, R. Lehmann, and N. Schilling. From quadrupedal animals to quadrupod animates. Biologically inspired construction and control architecture for a quadruped walking machine. In B. Edmonds and K. Dautenhahn, editors, *Proceedings of the Fifth International Conference of the Society for Adaptive Behavior (SAB)*, pages 400–408. MIT Press, Cambridge, MA, 1998.

[87] AIM Learning Inc. *Discipulus Owner's Manual*. http://www.aimlearning.com/Technology Overview.htm. WWW-Document.

[88] International Society for Adaptive Behavior. *From Animals to Animats: International Conference on the Simulation of Adaptive Behavior (SAB)*. MIT Press, 1990–2002.

[89] P. M. Todd J.-A. Meyer, editor. *Adaptive Behavior*. MIT Press/Sage Publications, 1990–.

[90] N. Jakobi. *Minimal Simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, 1998.

[91] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, and J. Merelo, editors, *Proceedings of the 3rd European Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 704–720. Springer New York, Berlin, Heidelberg, 1995.

[92] Y. Jin. *Fitness Approximation in Evolutionary Computation - Bibliography*. http://www.soft-computing.de/amec.html. WWW-Document.

[93] Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.

[94] G. Joblove, W. Latham, K. Sims, S. Todd, and M. Tolson. The applications of evolutionary and biological processes to computer art and animation. In L. Valastyan and L. Walsh, editors, *Proceedings of the Annual Conference on Computer Graphics*, pages 389–390, New York, NY, USA, August 1993. ACM Press.

[95] B. A. Julstrom. Adaptive operator probabilities in a genetic algorithm that applies three operators. In B. Bryant, J. Carroll, D. Oppenheim, J. Hightower, and K.M. George, editors, *Selected Areas in Cryptography*, pages 233–238. ACM Press, New York, 1997.

[96] B. A. Julstrom. What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 81–87. Morgan Kaufmann, San Francisco, CA, 1995.

[97] W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In J. F. Miller, M. Tomassini, P. Luca Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Proceedings of the fourth European Conference on Genetic Programming (EuroGP)*, volume 2038 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

[98] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf. Empirical analysis of different levels of meta-evolution. In P. J. Angeline and V. W. Porto, editors, *1999 Congress on Evolutionary Computation (CEC'99)*, volume 3, pages 2086–2093. IEEE Press, Piscataway NJ, 1999.

[99] O. Khatib, 2002. Personal communication.

[100] H.-S. Kim and S.-B. Cho. An efficient genetic algorithms with less fitness evaluation by clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 887–894. IEEE Press, 2001.

[101] J. Kim, W. K. Chung, Y. Youm, and B. H. Lee. Real-time ZMP compensation method using null motion for mobile manipulators. In *International Conference on Robotics and Automation (ICRA)*, pages 1967–1972. IEEE Press, 2002.

[102] S. Kirkpatrik, G. Gellatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[103] K. Kleiner. Look to the insect. *New Scientist, No. 1951, 12 Nov. 1994*, 144:27–29, 1994.

[104] M. Komosinski. The world of framsticks: Simulation, evolution, interaction. In J.-C. Heudin, editor, *Proceedings of the Second International Conference on Virtual Worlds*, volume 1834 of *Lecture Notes in Artificial Intelligence*, pages 214–224. Springer Verlag, Berlin, Heidelberg, 2000.

[105] J. Koza. *Genetic Programming II*. MIT Press, Cambridge, MA, 1994.

[106] J. Koza, D. Andre, F. Bennett, and A. Keane. *Genetic Programming III*. Morgan Kaufmann, San Francisco, CA, 1999.

[107] J. Koza and J. Rice. Automatic programming of robots using genetic programming. In *Proceedings of AAAI-92*, pages 194–201. MIT Press, Cambridge, MA, 1992.

[108] J. R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[109] W.-P. Lee, H. Hallam, and H. H. Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of the IEEE 4th International Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1997.

[110] M. Lewis, A. Fagg, and G. Bekey. Genetic algorithms for gait synthesis in a hexapod robot. In Y.F. Zheng, editor, *Recent trends in mobile robots*. World Scientific, 1993.

[111] M. A. Lewis, A. H. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network control of a walking robot. In *Proceedings of the International Conference on Robotics and Automation*, pages 2618–2623. Electronica Bks, 1992.

[112] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[113] S. Margetts and A. J. Jones. An adaptive mapping for developmental genetic programming. In J. F. Miller, M. Tomassini, P. Luca Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Proceedings of the fourth European Conference on Genetic Programming (EuroGP)*, volume 2038 of *Lecture Notes in Computer Science*, pages 97–107. Springer, Berlin, 2001, 2001.

[114] O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

[115] S. McMillan. *DynaMechs (Dynamics of Mechanisms): A Multibody Dynamic Simulation Library*. http://dynamechs.sourceforge.net. WWW-Document.

[116] S. Meyret, A. Muller, A. Siadat, and G. Abba. Adaptive neuro-fuzzy control of the rabbit biped robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 131–138. Professional Engineering Publishing, 2002.

[117] O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434, 1995.

[118] B. Mirtich. *Multibody Dynamics*. Mitsubishi Electric Research Laboratories, www.merl.com/projects/rigidBodySim/multibodyDyn. WWW-Document.

[119] F. Mondada and D. Floreano. Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16:183–195, 1995.

[120] S. Mondal, T. P. Sattar, and B. Bridge. TOFD inspection of V-groove butt welds on the hull of a container ship with a magnetically adhering wall climbing robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 955–961. Professional Engineering Publishing, 2002.

[121] N. Mori, S. Imanishi, H. Kita, and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 299–306. Morgan Kaufmann, San Francisco, CA, 1997.

[122] L. N. Moussi, R. R. Gudwin, F. J. Von Zuben, and M-K. Madrid. *Advances in Signal Processing, Robotics and Communications*, chapter Neural Networks in Classifier Systems (NNCS): An Application to Autonomous Navigation, pages 256–262. Electrical and Computer Engineering Series. WSES Press, 2001.

[123] L. N. Moussi, F. J. Von Zuben, R. R. Gudwin, and M-K. Madrid. A simulator using classifier systems with neural networks for autonomous robot navigation. In *Proceedings of the World Conferece on Computational Intelligence (WCCI)*. IEEE, 2002.

[124] R. Nakano, Y. Davidor, and T. Yamada. Optimal population size under constant computation cost. In Y. Davidor et al., editor, *Parallel Problem Solving from Nature – PPSN III*, pages 130–138. Springer, Berlin, 1994.

[125] J. Niehaus and W. Banzhaf. Adaption of operator probabilities in genetic programming. In J. F. Miller, M. Tomassini, P. Luca Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *Lecture Notes in Computer Science*, pages 325–336. Springer, Berlin, 2001.

[126] S. B. Niku. *Introduction to Robotics - Analysis, Systems, Aplications*. Prentice Hall, Upper Saddle River, NJ, 2001.

[127] S Nolfi and D. Floreano. *Evolutionary Robotics*. MIT Press, Cambridge, MA, 2001.

[128] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In R.A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Conference on Artificial Life*, pages 190–197. MIT Press/Bradford Books, 1994.

[129] P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press, Cambridge, MA., 1994.

[130] P. Nordin and W. Banzhaf. Genetic programming controlling a miniature robot. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67, MIT, Cambridge, MA, 10–12 November 1995. AAAI, Menlo Park, CA.

[131] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5:107–140, 1997.

[132] P. Nordin and W. Banzhaf. Real time control of a Khepera robot using genetic programming. *Cybernetics and Control*, 26(3):533–561, 1997.

[133] P. Nordin, W. Banzhaf, and M. Brameier. Evolution of a world model for a miniature robot using genetic programming. *Technical Report SysReport 4/95, Univ. of Dortmund, Dept. of Computer Science*, 1995.

[134] M. Olmer, W. Banzhaf, and P. Nordin. Evolving real-time behavior modules for a real robot with genetic programming. In M. Jamshidi, F. Pin, and P. Dauchez, editors, *Proceedings of the International Symposium on Robotics and Manufacturing (ISRAM-96)*, Robotics and Manufacturing, pages 675 – 680. Asme Press, New York, 1996.

[135] G. B. Parker and J. W. Mills. Adaptive hexapod gait control using anytime learning with fitness biasing. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 519–524. Morgan Kaufmann, 1999.

[136] J. M. Porta and E. Celaya. Efficient gait generation using reinforcement learning. In K. Berns and R. Dillmann, editors, *Proceedings of the Fourth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 411–418. Professional Engineering Publishing, 2001.

[137] D. K. Pratihar, K. Deb, and A. Ghosh. Design of a genetic-fuzzy system for planning optimal path and gait simultaneously of a six-legged robot. In W. Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1678–1684. Morgan Kaufmann, 1999.

[138] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 2 edition, 1993.

[139] M. H. Raibert. *Legged robots that balance*. MIT Press, Cambridge, MA, 1986.

[140] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzbog, Stuttgart, 1973.

[141] C.W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[142] M. Risler. *InvKinWalkingEngine - Kurze Beschreibung der Parameter*. Technische Universität Darmstadt, GermanTeam 2002, 2002.

[143] R. E. Roberson and R. Schwertassek. *Dynamics of Multibody Systems*. Springer Berlin, 1988.

[144] J. P. Rosca and D. H. Ballard. Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, 1994.

[145] J. P. Rosca and D. H. Ballard. Learning by adapting representations in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Orlando, Florida, USA*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[146] S. Saripalli, D. J. Naffin, and G. S. Sukhatme. Autonomous flying vehicle research at the university of southern california. In *Proceedings of First International Workshop on Multi-Robot Systems*, pages 1–8, 2002.

[147] K. Sastry, D.E. Goldberg, and M. Pelikan. Don't evaluate, inherit. In L. Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 551–558. Morgan Kaufmann, 2001.

[148] T. P. Sattar, Z. Zhao, J. Feng, B. Bridge, S. Mondal, and S. Chen. Internal in-service inspection of the floor and walls of oil, petroleum, and chemical storage tanks with a mobile robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 947–954. Professional Engineering Publishing, 2002.

[149] A. Schults and J.J. Grefenstette. Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation and Control Conference*, pages 739–749. AIAA, 1992.

[150] H.-P. Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. Diplomarbeit, Technische Universität Berlin, Hermann Föttinger–Institut für Strömungstechnik, März 1965.

[151] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, Inc., New York, 1995.

[152] C. G. Shaefer. The ARGOT strategy: Adaptive representation genetic optimizer technique. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–58. Lawrence Erlbaum Associates, Mahwah, NJ, 1987.

[153] K. Sims. Artificial evolution for computer graphics. *ACM Computer Graphics*, 25(4):319–328, July 1991. SIGGRAPH '91 Proceedings.

[154] K. Sims. Interactive evolution of dynamical systems. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 171–178. MIT Press, 1992.

[155] K. Sims. Evolving 3D morphology and behavior by competition. In R. A. Brooks and P. Maes, editors, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV*, pages 28–39. MIT Press, Cambridge, MA, July 1994.

[156] K. Sims. Evolving virtual creatures. In A. Glassner, editor, *Proceedings of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22. ACM SIGGRAPH, ACM Press, 1994.

[157] M. Sinclair. Operator-probability adaptation in a genetic-algorithm / heuristic hybrid for optical network wavelength allocation. In *Proceedings of the International Conference on Evolutionary Computation (ICEC)*, pages 840–845. IEEE Press, New York, 1998.

[158] J. E. Smith and T. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

[159] J. E. Smith and T. C. Fogarty. Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 441–450. Springer, Berlin, 1996.

[160] J. E. Smith and T. C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the International Conference on Evolutionary Computation (ICEC)*, pages 318–323. IEEE Press, New York, 1996.

[161] G. Spencer. Automatic generation of programs for crawling and walking. In K. E. Kinnear, editor, *Advances in Genetic Programming*, Complex Adaptive Systems, pages 335–354, Cambridge, 1994. MIT Press.

[162] M. Srinivas and L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-24(4):656–667, 1994.

[163] L. Steels. When are robots intelligent agents? *Robotics and Autonomous Systems*, 15(1–2):3–9, 1995. Elsevier Science Publishers (North-Holland), Amsterdam, The Netherlands.

[164] M. Svinin, S. Hosoe, and K. Ueda. Optimal decentralization of reinforcement learning schemes in acquiring gait patterns by walking machines. In K. Berns and R. Dillmann, editors, *Proceedings of the Fourth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 427–434. Professional Engineering Publishing, 2001.

[165] H. Tagaki. Interactive evolutionary computation: System optimization based on human subjective evaluation. In *IEEE International Conference on Intelligent Engineering Systems (INES)*, pages 1–6. IEEE Press, 1998.

[166] K. Tanie, 2001. Personal communication.

[167] R. Tedrake and H. S. Seung. Improved dynamic stability using reinforcement learning. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 341–348. Professional Engineering Publishing, 2002.

[168] A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. *Symbolic Visual Learning*, pages 81–116, 1996. Oxford University Press, Oxford, UK.

[169] J. Toner and Y. Tu. Flocks, herds, and schools: a quantitative theory of flocking. *Physical Review E*, 58(4):4828–4858, 1998.

[170] A. Tuson and P. Ross. Co-evolution of operator settings in genetic algorithms. In T. C. Fogarty, editor, *Proceedings of the Third AISB Workshop on Evolutionary Computing*, pages 286–296. Springer, Berlin, 1996.

[171] A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.

[172] H. Uchida and K. Nonami. Biological walking of a mine detecting six-legged robot using rhythm generation and fuzzy control. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the Fifth International Conference on Climbing and Walking Robots (CLAWAR)*, pages 741–748. Professional Engineering Publishing, 2002.

[173] M. Veloso, W. Uther, M. Fujita, M. Asada, and H. Kitano. Playing soccer with legged robots. In *Proceedings of the Intelligent Robots and Systems Conference (IROS)*, pages 437–442. IEEE Press, 1998.

[174] S.W. Wilson and D.E. Goldberg. A critical rewiew of classifier systems. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their applications*. Morgan Kaufmann, San Francisco, CA, 1989.

[175] H. Witte. Hints for the construction of anthropomorphic robots based on the functional morphology of human walking. *Journal of the Robotic Society of Japan*, 20(3):247 – 254, 2002.

[176] H. Witte, R. Hackert, K.E. Lilje, N. Schilling, D. Voges, G. Klauer, W. Ilg, J. Albiez, A. Seyfarth, D. Germann, M. Hiller, R. Dillmann, and M.S. Fischer. Transfer of biological principles into the construction of quadruped walking machines. In *Proceedings of the Second International Workshop on Robot Motion and Control (RoMoCo)*, pages 245 – 250. IEEE, 2001.

[177] K. Wolff and P. Nordin. Evolution of efficient gait with autonomous biped robot using visual feedback. In D. Goodman, editor, *Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 482–489. Morgan Kaufmann, 2001.

[178] A. Zell. *Simulation Neuronaler Netze*. Oldenbourg, 2000.

Teil IV

Anhänge

Anhang A

Voruntersuchungen

A.1 Vorzeitiger Abbruch

Parameter	**Wert**
Ziel:	Evolviere die Bewegung eines Beines entlang einer definierten Trajektorie.
Terminalmenge	$\mathbb{R}$
Funktionsmenge	Kräfte und Wirkzeiten
Selektionsschema	Turnierselektion, Größe 4
Populationsgröße	600
Crossoverwahrscheinlichkeit	0.3 für jeden Crossoveroperator
Mutationswahrscheinlichkeit	0.45 für jeden Mutationsoperator
Abbruchbedingung	Max. Anzahl Turniere
Maximale Länge der Programme	unbegrenzt
Initialisierung	zufällig

Tabelle A.1: Parametereinstellungen für das Experiment zum vorzeitigen Abbruch von Auswertungen.

A.2 Adaptive Operatorwahrscheinlichkeiten

Parameter	**Testproblem**		
Zielfunktion	Ringe	Sinus	Hexapod
Repräsentation	linear	linear	linear
Terminalmenge	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{N}$
Funktionsmenge	{+,-,mul,div}	{+,-,mul,div,sin,cos}	{wait,stance,swing}
Selektionsschema	Turnier	Turnier	Turnier
Populationsgröße	30	50	50
Crossoverwahrscheinlichkeit	siehe Experiment		
Mutationswahrscheinlichkeit	siehe Experiment		
adaptation window	10 Generationen		
Abbruch nach (Generationen)	50	100	1000
Maximale Länge der Programme	unbegrenzt	unbegrenzt	unbegrenzt
Initialisierungsmethode	*half and half*	*half and half*	*half and half*

Tabelle A.2: Koza-Tableau mit allen Parametern des SYSGP-Systems für jedes der drei Benchmarkprobleme.

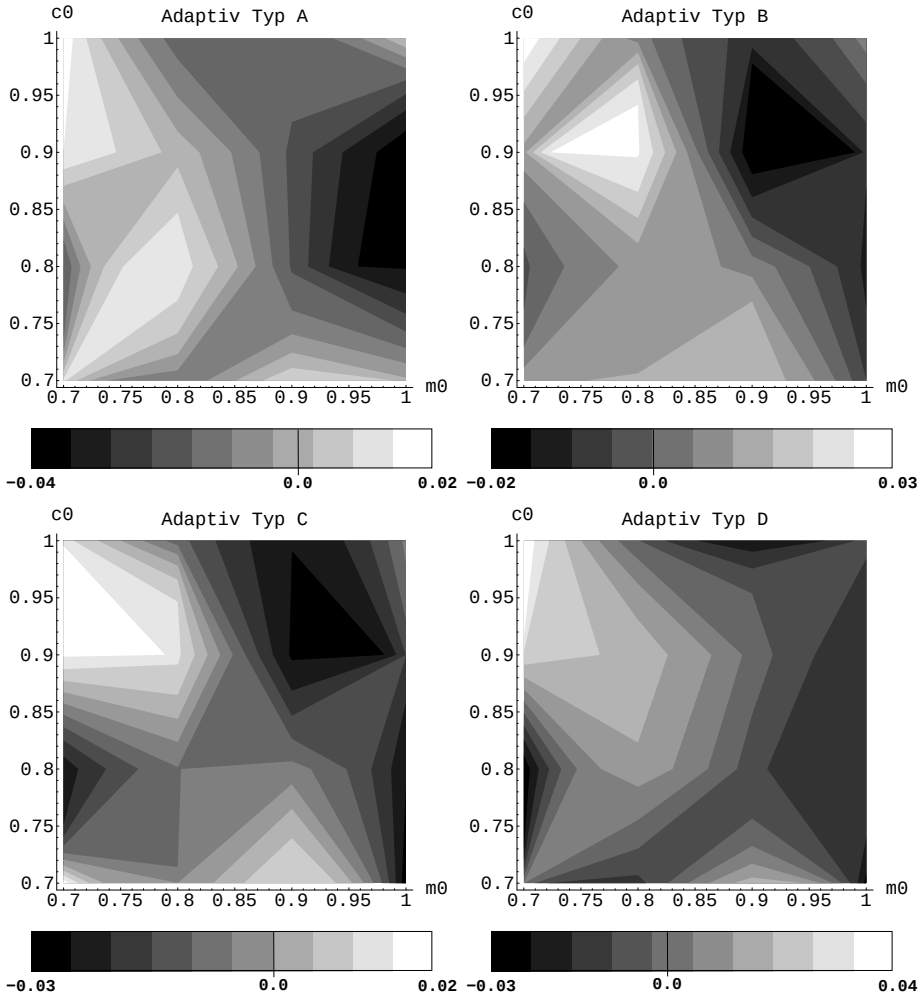


Abbildung A.1: **Sinus:** Die durchschnittliche Fitness von 200 Experimenten mit adaptiver Anpassung Typ A-D im Verhältnis zur statischen Variante. Deutlich zu erkennen ist, dass es nahezu keine Verbesserungen gibt (kleine helle Flächen).

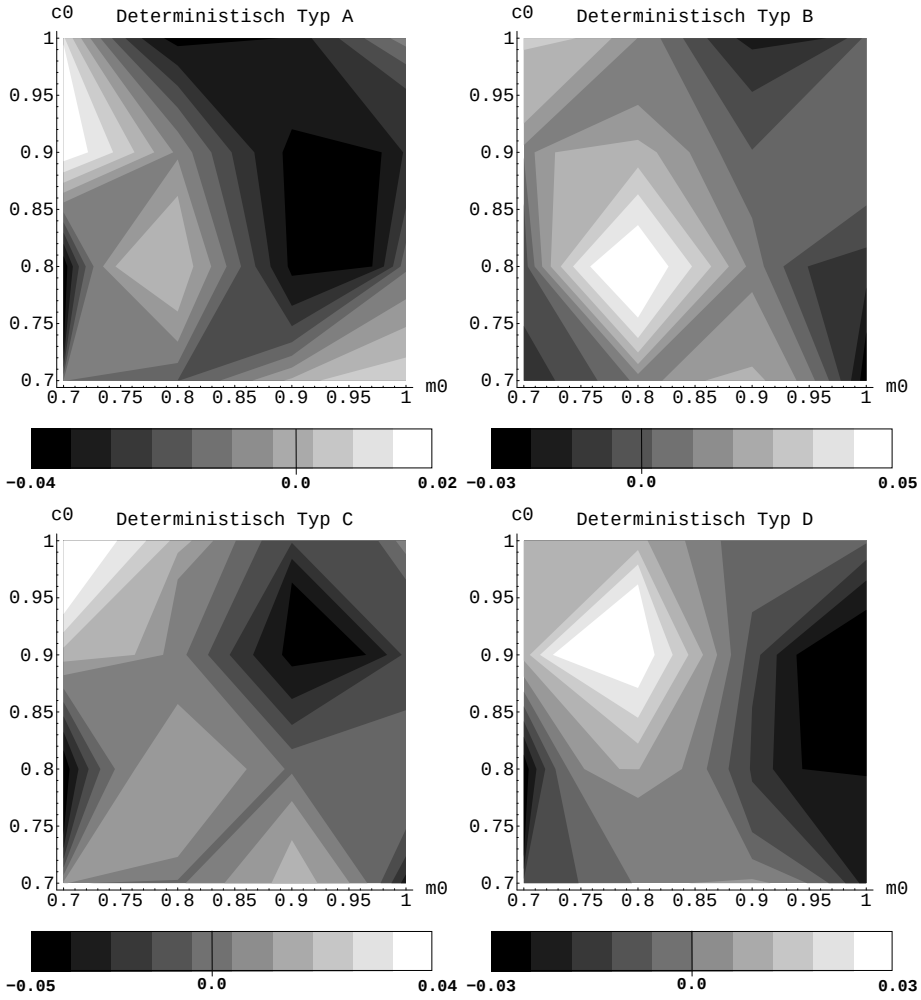


Abbildung A.2: **Sinus:** Die durchschnittliche Fitness von 200 Experimenten mit deterministischer Anpassung Typ A-D im Verhältnis zur statischen Variante. Auch hier ist keine wirkliche Verbesserung zu erkennen.

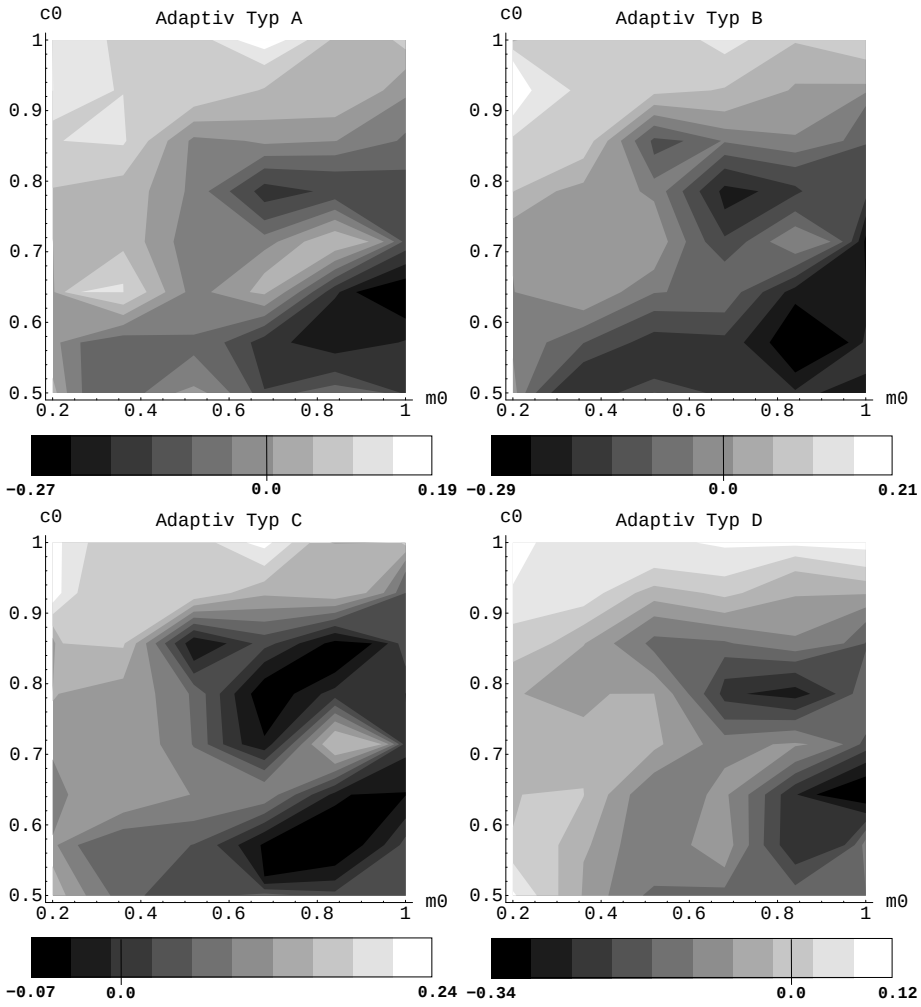


Abbildung A.3: **Hexapod:** Die durchschnittliche Fitness von 200 Experimenten mit adaptiver Anpassung Typ A-D im Verhältnis zur statischen Variante.

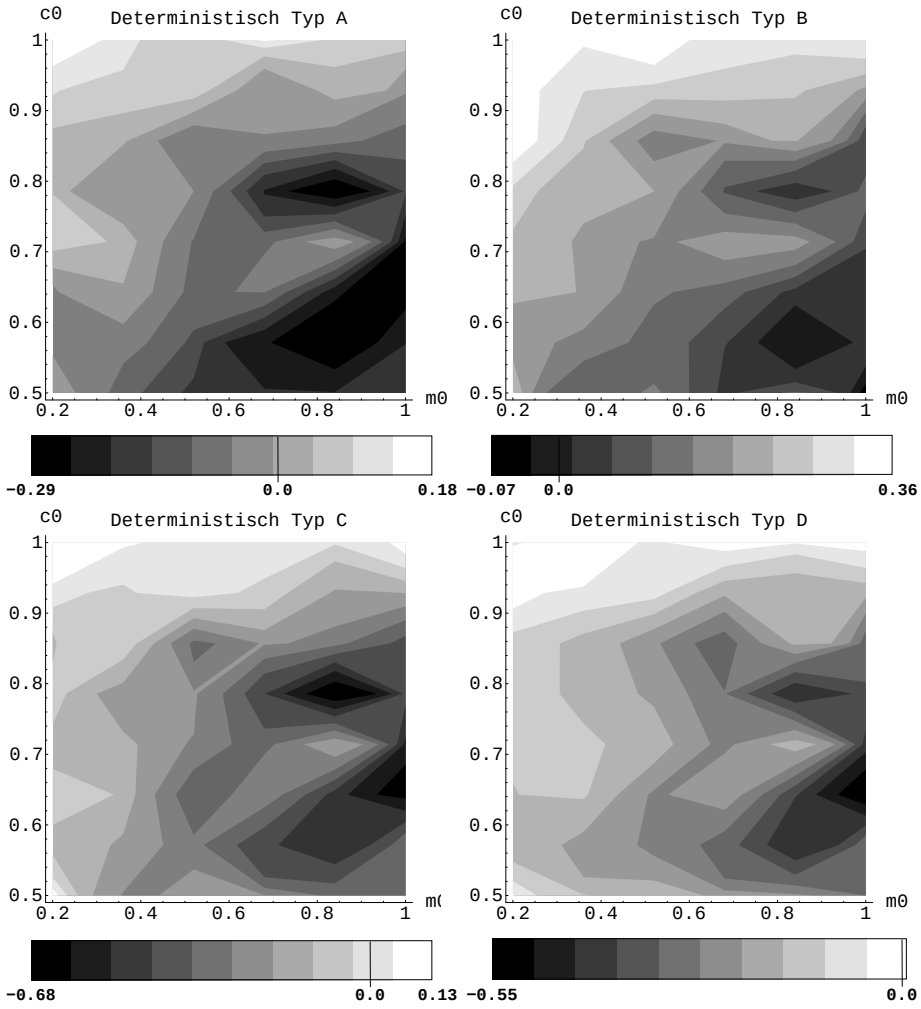


Abbildung A.4: **Hexapod:** Die durchschnittliche Fitness von 200 Experimenten mit deterministischer Anpassung Typ A-D im Verhältnis zur statischen Variante. Variante B zeigt die beste Performance (starke Verbesserung, nahezu keine Verschlechterung über den gesamten Parameterbereich)

A.3 Meta-Modell

Parameter	**Wert**
Fitnessfunktion	Maximiere Gleichung (7.21) bzw. (7.22)
Anzahl der Läufe	100
Populationsgröße	100
Länge der Individuen	10 Bit
Crossoverwahrscheinlichkeit	0.3
Mutationswahrscheinlichkeit	0.7
Initialisierung	random
Selektionsschema	Turnierselektion (Größe = 4)
Abbruchbedingung	Durchschn. Fitness der Population $>$ 80% der max. Fitness

Tabelle A.3: Evolution mit Klassifizierung von Turnieren. Parameter des Genetischen Algorithmus.

Parameter	**Wert**
Laufzeit von C (t_K)	0.1 s
Speed factor s	$[1,\ldots,1000]$
Laufzeit einer Auswertung (t_A)	$t_A = s \cdot t_K$
Konfidenz-Sollniveau k	$[0.7,\ldots,1.0]$
Fehlerrate p_e	$\{0.05, 0.1, 0.2, 0.3\}$
Dauer der Startphase n_S (ca. 25% der Standardlaufzeit)	50 (Gleichung (7.21)) bzw. 400 (Gleichung. (7.22))
Standardlaufzeit (N)	200 Turniere (nach Gleichung (7.21)), 1528 Turniere (Gleichung (7.22))
Lernzeit t_L des Klassifizierers C'	10.0s

Tabelle A.4: Parameter für die Berechnung der Laufzeit.

Anhang B

Parameter der Experimente mit SIGEL

B.1 Der minimale Roboter

Parameter	**Wert**	**Parameter**	**Wert**
Drehgelenke	1	davon motorisiert	1
Schubgelenke	0	davon motorisiert	0
Freiheitsgrade (DOF)	1		
Sensoren	1		
Beine	0		

Tabelle B.1: Kinematische Details des einfachen Robotermodells in Abbildung 8.1.

Parameter	**Wert**	**Parameter**	**Wert**
Bibliothek	DynaMechs	Gelenk-Begrenzungs-Federkonstante	25.000
Simulierte Zeit	30s	Random Seed	0
Numerisches Verfahren	Runge-Kutta	Gelenkreibungskonstante	0.35
Schrittweite	0.01	Gelenk-Begrenzungs-Dämpferkonstante	2500

Tabelle B.2: Simulationsparameter des Experimentes mit dem minimalen Roboter in Abbildung 8.1.

Parameter	**Wert**
Gravitation	9.81 $\frac{m}{s^2}$
Plan. Federkonstante	5500
Norm. Federkonstante	7500
Plan. Dämpferkonstante	50
Norm. Dämpferkonstante	50
Stat. Reibkoeffizient	1.5
Kin. Reibkoeffizient	1.0

Tabelle B.3: Umgebungsparameter des Experimentes mit dem minimalen Roboter in Abbildung 8.1.

Befehl	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**
ADD	0.001	0.077	JMP	0.0	0.0	MOVE	0.01	0.077
CMP	0.001	0.077	LOAD	0.001	0.077	MUL	0.001	0.077
COPY	0.001	0.077	MAX	0.001	0.077	NOP	0.0	0.0
DELAY	0.001	0.077	MIN	0.001	0.077	SENSE	0.001	0.077
DIV	0.002	0.077	MOD	0.001	0.077	SUB	0.001	0.077

Tabelle B.4: Die Sprachparameter des Experimentes mit dem minimalen Roboter in Abbildung 8.1.

Parameter	**Wert**
Fitnessfunktion	SimpleFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.67
Mutationswahrscheinlichkeit	0.05
Reproduktionswahrscheinlichkeit	0.28
Selektionsschema	Turnierselektion
Min. Programmlänge	10
Max. Programmlänge	1024
Maximales Alter	50
Random Seed	0
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.5: Die GP-Parameter des Experimentes mit dem minimalen Roboter in Abbildung 8.1.

B.2 Der lineare Roboter

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Drehgelenke	4	davon motorisiert	1
Schubgelenke	0	davon motorisiert	0
Freiheitsgrade (DOF)	4		
Sensoren	0		
Beine	0		

Tabelle B.6: Kinematische Details des linearen Roboters in Abbildung 8.2.

Befehl	Dauer	Prob.	Befehl	Dauer	Prob.	Befehl	Dauer	Prob.
ADD	0.001	0.082	JMP	0.0	0.0	MOVE	0.1	0.096
CMP	0.001	0.082	LOAD	0.001	0.082	MUL	0.001	0.082
COPY	0.001	0.082	MAX	0.001	0.082	NOP	0.0	0.0
DELAY	0.001	0.082	MIN	0.001	0.082	SENSE	0.001	0.005
DIV	0.002	0.082	MOD	0.001	0.082	SUB	0.001	0.082

Tabelle B.7: Die Sprachparameter des Experimentes mit dem linearen Roboter in Abbildung 8.2.

Parameter	Wert
Fitnessfunktion	NiceWalking Fitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.3
Mutationswahrscheinlichkeit	0.7
Reproduktionswahrscheinlichkeit	0.0
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	300
Maximales Alter	50
Random Seed	1
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.8: Die GP-Parameter des Experimentes mit dem linearen Roboter in Abbildung 8.2.

B.3 Der zweibeinige Roboter

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Drehgelenke	6	davon angetrieben	6
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	6		
Sensoren	6		
Beine	2		

Tabelle B.9: Kinematische Details des Zweibeiners in Abbildung 8.3.

Befehl	Dauer	Prob.	Befehl	Dauer	Prob.	Befehl	Dauer	Prob.
ADD	0.001	0.082	JMP	0.0	0.0	MOVE	0.1	0.096
CMP	0.001	0.082	LOAD	0.001	0.082	MUL	0.001	0.082
COPY	0.001	0.082	MAX	0.001	0.082	NOP	0.0	0.0
DELAY	0.001	0.082	MIN	0.001	0.082	SENSE	0.001	0.005
DIV	0.002	0.082	MOD	0.001	0.082	SUB	0.001	0.082

Tabelle B.10: Die Sprachparameter des Experimentes mit dem Zweibeiner (Abbildung 8.3).

Parameter	Wert
Fitnessfunktion	NiceWalkingFitness/SimpleFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.3
Mutationswahrscheinlichkeit	0.7
Reproduktionswahrscheinlichkeit	0.0
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	1
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.11: Die GP-Parameter des Zweibeiner-Experimentes.

B.4 Der Dreibeiner

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Drehgelenke	9	davon angetrieben	9
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	9		
Sensoren	9		
Beine	3		

Tabelle B.12: Kinematische Details des Dreibeiners in Abbildung 8.5.

Parameter	Wert	Parameter	Wert
Bibliothek	DynaMechs	Gelenk-Begrenzungs-Federkonstante	25.000
Simulierte Zeit	180s	Random Seed	0
Numerisches Verfahren	Runge-Kutta	Gelenkreibungskonstante	0.35
Schrittweite	0.01	Gelenk-Begrenzungs-Dämpferkonstante	5

Tabelle B.13: Simulationsparameter des Experimentes mit dem Dreibeiner in Abbildung 8.5.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness/SimpleFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.3
Mutationswahrscheinlichkeit	0.7
Reproduktionswahrscheinlichkeit	0.0
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	300
Maximales Alter	50
Random Seed	1
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.14: Die GP-Parameter des Experimentes mit dem Dreibeiner in Abbildung 8.5.

B.5 Der vierbeinige Roboter

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Drehgelenke	12	davon angetrieben	12
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	12		
Sensoren	12		
Beine	4		

Tabelle B.15: Kinematische Details des Vierbeiners in Abbildung 8.7.

Parameter	**Wert**	**Parameter**	**Wert**
Bibliothek	DynaMechs	Gelenk-Begrenzungs-Federkonstante	500
Simulierte Zeit	60s	Random Seed	1234
Numerisches Verfahren	Runge-Kutta	Gelenkreibungskonstante	0.35
Schrittweite	0.02	Gelenk-Begrenzungs-Dämpferkonstante	5

Tabelle B.16: Simulationsparameter des Experimentes mit dem Vierbeiner in Abbildung 8.7.

Parameter	**Wert**
Gravitation	9.81 $\frac{m}{s^2}$
Plan. Federkonstante	5500
Norm. Federkonstante	7500
Plan. Dämpferkonstante	50
Norm. Dämpferkonstante	50
Stat. Reibkoeffizient	1.5
Kin. Reibkoeffizient	1.5

Tabelle B.17: Umgebungsparameter des Experimentes mit dem Vierbeiner in Abbildung 8.7.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.05
Mutationswahrscheinlichkeit	0.8
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	1
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.18: Die GP-Parameter des Vierbeiner-Experimentes.

B.6 Sechsbeiner

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Drehgelenke	18	davon angetrieben	18
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	18		
Sensoren	18		
Beine	6		

Tabelle B.19: Kinematische Details des Sechsbeiners in Abbildung 8.8.

Parameter	Wert	Parameter	Wert
Bibliothek	DynaMechs	Gelenk-Begrenzungs-Federkonstante	500
Simulierte Zeit	60s	Random Seed	1234
Numerisches Verfahren	Runge-Kutta	Gelenkreibungskonstante	0.35
Schrittweite	0.01	Gelenk-Begrenzungs-Dämpferkonstante	1

Tabelle B.20: Simulationsparameter des Experimentes mit dem Sechsbeiners in Abbildung 8.8.

Parameter	Wert
Gravitation	9.81 $\frac{m}{s^2}$
Plan. Federkonstante	5500
Norm. Federkonstante	7500
Plan. Dämpferkonstante	1.0
Norm. Dämpferkonstante	1.0
Stat. Reibkoeffizient	3.0
Kin. Reibkoeffizient	2.9

Tabelle B.21: Umgebungsparameter des Experimentes mit dem Sechsbeiners in Abbildung 8.8.

B.7 Vierbeiner, hohe Mutationswahrscheinlichkeit

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.5 für den *Vierbeiner* aufgeführten Werte.

Parameter	Wert
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.05
Mutationswahrscheinlichkeit	0.8
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	Time
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.22: Die GP-Parameter des Vierbeiner-Experimentes mit hoher Mutations- und niedriger Crossoverwahrscheinlichkeit. Alle anderen Einstellungen sind in den Tabellen B.15, B.16, B.17 und B.18 aufgeführt.

B.8 Vierbeiner, hohe Crossoverwahrscheinlichkeit

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.5 für den *Vierbeiner* aufgeführten Werte.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.8
Mutationswahrscheinlichkeit	0.05
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	Time
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.23: Die GP-Parameter des Vierbeiner-Experimentes mit niedriger Mutations- und hoher Crossoverwahrscheinlichkeit. Alle anderen Einstellungen sind in den Tabellen B.15, B.16, B.17 und B.18 aufgeführt.

B.9 Vierbeiner, reduzierter Befehlssatz

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.5 für den *Vierbeiner* aufgeführten Werte.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.05
Mutationswahrscheinlichkeit	0.8
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	Time
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.24: Die GP-Parameter des Vierbeiner-Experimentes mit hoher Mutations- und niedriger Crossoverwahrscheinlichkeit. Alle anderen Einstellungen sind in den Tabellen B.15, B.16, B.17 und B.18 aufgeführt.

Befehl	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**
ADD	0.001	0.24	JMP	0.0	0.0	MOVE	0.2	0.28
CMP	0.0	0.0	LOAD	0.001	0.24	MUL	0.0	0.0
COPY	0.001	0.24	MAX	0.0	0.0	NOP	0.0	0.0
DELAY	0.0	0.0	MIN	0.0	0.0	SENSE	0.001	0.14
DIV	0.0	0.0	MOD	0.0	0.0	SUB	0.0	0.0

Tabelle B.25: Reduzierter Befehlssatz des Experimentes mit dem Vierbeiner

B.10 Vierbeiner mit versteiftem Bein, Neustart

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.5 für den *Vierbeiner* aufgeführten Werte.

Drehgelenke	12	davon angetrieben	9
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	9		
Sensoren	12		
Beine	4		

Tabelle B.26: Kinematische Details des Vierbeiners mit versteiftem Bein in Abbildung 8.10.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.05
Mutationswahrscheinlichkeit	0.8
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	Time
Initialisierung	Random
Populationsgröße	100
Interaktive Evolution	Nein

Tabelle B.27: Die GP-Parameter des Vierbeiner-Experimentes mit versteiftem Bein und kompletter Neu-Evolution. Alle anderen Einstellungen sind in den Tabellen B.15, B.16, B.17 und B.18 aufgeführt.

B.11 Vierbeiner mit versteiftem Bein, 50% zufällig initialisiert, 50% aus vorhergegenden Experimenten

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.5 für den *Vierbeiner* aufgeführten Werte.

Drehgelenke	12	davon angetrieben	9
Schubgelenke	0	davon angetrieben	0
Freiheitsgrade (DOF)	9		
Sensoren	12		
Beine	4		

Tabelle B.28: Kinematische Details des Vierbeiners mit versteiftem Bein in Abbildung 8.10.

Parameter	**Wert**
Fitnessfunktion	NiceWalkingFitness
Terminalmenge	$\mathbb{N}$
Crossoverwahrscheinlichkeit	0.05
Mutationswahrscheinlichkeit	0.8
Reproduktionswahrscheinlichkeit	0.15
Selektionsschema	Turnierselektion
Min. Programmlänge	100
Max. Programmlänge	1000
Maximales Alter	50
Random Seed	Time
Initialisierung	50%Random, 50% Wiederverwendung
Populationsgröße	100
Interaktive Evolution	Ja

Tabelle B.29: Die GP-Parameter des Vierbeiner-Experimentes mit versteiftem Bein und Wiederverwendung von Individuen. Alle anderen Einstellungen sind in den Tabellen B.15, B.16, B.17 und B.18 aufgeführt.

Anhang C

Parameter der Experimente mit ZORC

C.1 Experimente in SIGEL

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Parameter	**Wert**	**Parameter**	**Wert**
Drehgelenke	13	davon motorisiert	13
Schubgelenke	0	davon motorisiert	0
Freiheitsgrade (DOF)	13		
Sensoren	15		
Beine	2		

Tabelle C.1: Kinematische Details des Robotermodells in Abbildung 9.6.

Parameter	**Wert**	**Parameter**	**Wert**
Bibliothek	DynaMechs	Gelenk-Begrenzungs-Federkonstante	25.000
Simulierte Zeit	10 s	Random Seed	0
Numerisches Verfahren	Runge-Kutta	Gelenkreibungskonstante	0.38
Schrittweite	0.00005	Gelenk-Begrenzungs-Dämpferkonstante	0

Tabelle C.2: Simulationsparameter des Experimentes mit ZORC .

Parameter	**Wert**
Gravitation	9.81 $\frac{m}{s^2}$
Plan. Federkonstante	5500
Norm. Federkonstante	7500
Plan. Dämpferkonstante	50
Norm. Dämpferkonstante	50
Stat. Reibkoeffizient	1.5
Kin. Reibkoeffizient	1.0

Tabelle C.3: Umgebungsparameter des Experimentes mit ZORC .

Befehl	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**
ADD	0.001	0.077	JMP	0.001	0.077	MOVE	0.01	0.077
CMP	0.001	0.077	LOAD	0.001	0.077	MUL	0.001	0.077
COPY	0.001	0.077	MAX	0.001	0.077	NOP	0.001	0.077
DELAY	0.001	0.077	MIN	0.001	0.077	SENSE	0.001	0.077
DIV	0.001	0.077	MOD	0.001	0.077	SUB	0.001	0.077

Tabelle C.4: Die Sprachparameter des Experimentes mit ZORC .

Parameter	**Wert**
Fitnessfunktion	Nach Gleichung 9.1
Terminalmenge	$[-2^{31}, 2^{31}-1]$
Crossoverwahrscheinlichkeit	0.25
Mutationswahrscheinlichkeit	0.73
Reproduktionswahrscheinlichkeit	0.02
Selektionsschema	Turnierselektion
Min. Programmlänge	3
Max. Programmlänge	1024
Maximales Alter	100
Random Seed	0
Initialisierung	Random
Populationsgröße	75
Interaktive Evolution	Nein
Register	8 zu je 32 Bit

Tabelle C.5: Die GP-Parameter des Experimentes mit ZORC . Alle anderen Einstellungen sind in den Tabellen C.1 - C.4 aufgeführt.

C.2 Experimente mit dem realen Roboter

Wenn keine anderen Daten angegeben sind, gelten die in den Tabellen in Abschnitt B.1 für den minimalen Roboter aufgeführten Werte.

Befehl	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**	**Befehl**	**Dauer**	**Prob.**
ADD	0.01	0.07	JMP	0.01	0.07	MOVE	0.05	0.07
CMP	0.01	0.07	LOAD	0.01	0.07	MUL	0.01	0.07
COPY	0.01	0.07	MAX	0.01	0.07	NOP	0.01	0.077
DELAY	0.01	0.07	MIN	0.01	0.07	SENSE	0.05	0.02
DIV	0.01	0.07	MOD	0.01	0.07	SUB	0.01	0.07

Tabelle C.6: Die Sprachparameter des Experimentes mit dem realen ZORC .

Parameter	**Wert**
Fitnessfunktion	Manuelle Bewertung
Terminalmenge	$[-2^{31}, 2^{31}-1]$
Crossoverwahrscheinlichkeit	0.25
Mutationswahrscheinlichkeit	0.75
Reproduktionswahrscheinlichkeit	0.0
Selektionsschema	Turnierselektion
Min. Programmlänge	3
Max. Programmlänge	1024
Maximales Alter	100
Random Seed	697
Initialisierung	Aus früheren Experimenten
Populationsgröße	20
Interaktive Evolution	Ja
Register	16 zu je 16 Bit

Tabelle C.7: Die GP-Parameter des Experimentes mit dem realen ZORC . Alle anderen Einstellungen sind für das Experiment mit dem realen Roboter irrelevant, da sie die Simulation betreffen.

Anhang D

Parameter der Experimente mit dem Sony Aibo

D.1 Technische Daten des Sony Aibo

Parameter	**Wert**
Motoren	20
davon für die Beinbewegungen	12
Lautsprecher	1
Gelenksensoren	20
Druckschalter	7 (1 pro Fuß, 1 am Rücken, 2 am Kopf)
Beschleunigungssensoren	3
Kamera (Farb-CCD)	1
Mikrofon (8KHz, 8 Bit)	2 (li./re.)
PCMCIA-II Steckplatz	1 (WLAN)
Operationszeit (Normal/Fußball)	2h/10min.

Tabelle D.1: Technische Daten des vierbeinigen Laufroboters Aibo

D.2 Experimente mit der Inversen Kinematik

Parameter	Anzahl	Datentyp
Bewegungsmodus (kreisförmig, polygonal)	1	binär
Vorderbein-Ruheposition (relativ zum Ursprung des Roboterkoordinatensystems)	3	reell
Hinterbein-Ruheposition	3	reell
Neigungswinkel der Fußbewegungen (Vorder- und Hinterbein) zur y-Achse	2	reell
Maximaler Abstand des Fußes zum Boden (Vorder- und Hinterbein)	2	reell
Schrittlängenverhältnis zwischen Vorder- und Hinterbein	3	reell
Schrittlänge	2	reell
Frequenz der Teilbewegungen	3	reell
Neigungswinkel der überlagerten Rotation zur z-Achse	1	reell
Dauer eines Schrittes (in Taktraten zu je 8 ms)	1	reell
Anteil der Schwing- und Stehphasen an der Gesamtdauer eines Schrittes	2	reell
Neigungswinkel (Tilt, Pan, Roll) des Kopfes	3	reell
Status des Mundes (geöffnet, geschlossen)	1	binär

Tabelle D.2: Beschreibung der 27 Parameter der InvKinWalkingEngine [142].

Parameter	Wert
Bewegungsmodus	0 (polygonal)
Vorderbein-Ruheposition	v_0, v_1, v_2
Hinterbein-Ruheposition	v_3, v_4, v_5
Neigungswinkel der Fußbewegungen	0,0
Maximaler Abstand des Fußes zum Boden	v_6, v_7
Schrittlängenverhältnis zwischen Vorder- und Hinterbein	$\frac{v_8}{100}, \frac{v_9}{100}, \frac{v_{10}}{100}$
Schrittlänge	v_{11}, v_{12}
Frequenz der Teilbewegungen	$2.0, 2.0, 0.2$
Neigungswinkel der überlagerten Rotation zur z-Achse	0.25
Dauer eines Schrittes (in Taktraten zu je 8ms)	v_{13}
Anteil der Schwing- und Stehphasen an der Gesamtdauer eines Schrittes	$\frac{v_{14}}{100}, \frac{v_{15}}{100}$
Neigungswinkel (Tilt, Pan, Roll) des Kopfes	irrelevant
Status des Mundes (geöffnet, geschlossen)	irrelevant

Tabelle D.3: Wertebelegung der 27 Parameter der InvKinWalkingEngine mit den Daten eines Individuums $i_x = \{v_0, \ldots, v_{15}\}$.

Parameter	**Wert**
Fitnessfunktion	Geradeauslauf mit maximaler Geschwindigkeit
Terminalmenge	$\mathbb{R}$
Crossoverwahrscheinlichkeit	0.5
Mutationswahrscheinlichkeit	0.2
Reproduktionswahrscheinlichkeit	0.3
Selektionsschema	Turnierselektion (synchrones Update der Population)
Initialisierung	Standardparameter (GT2002)
Populationsgröße	26
Interaktive Evolution	Ja

Tabelle D.4: Parametereinstellungen des GA für die manuelle Evolution von Gangmustern mit dem Sony ERS-2100.

Parameter	**Wert**
Fitnessfunktion	Geradeauslauf mit maximaler Geschwindigkeit
Terminalmenge	$\mathbb{R}$
Crossoverwahrscheinlichkeit	0.4
Mutationswahrscheinlichkeit	0.125
Reproduktionswahrscheinlichkeit	0.0
Selektionsschema	Turnierselektion (synchrones Update der Population)
Initialisierung	Standardparameter (GT2002)
Populationsgröße	50
Interaktive Evolution	Ja (Meta-Evolution)

Tabelle D.5: Parametereinstellungen des GA für die manuelle Evolution von Gangmustern mit dem Sony ERS-2100.

Index

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z